OPC Data Client User's Guide

OPC Data Client User's Guide

Version 2020.2

Table of Contents

1 Edition Notice

OPC Data Client User's Guide

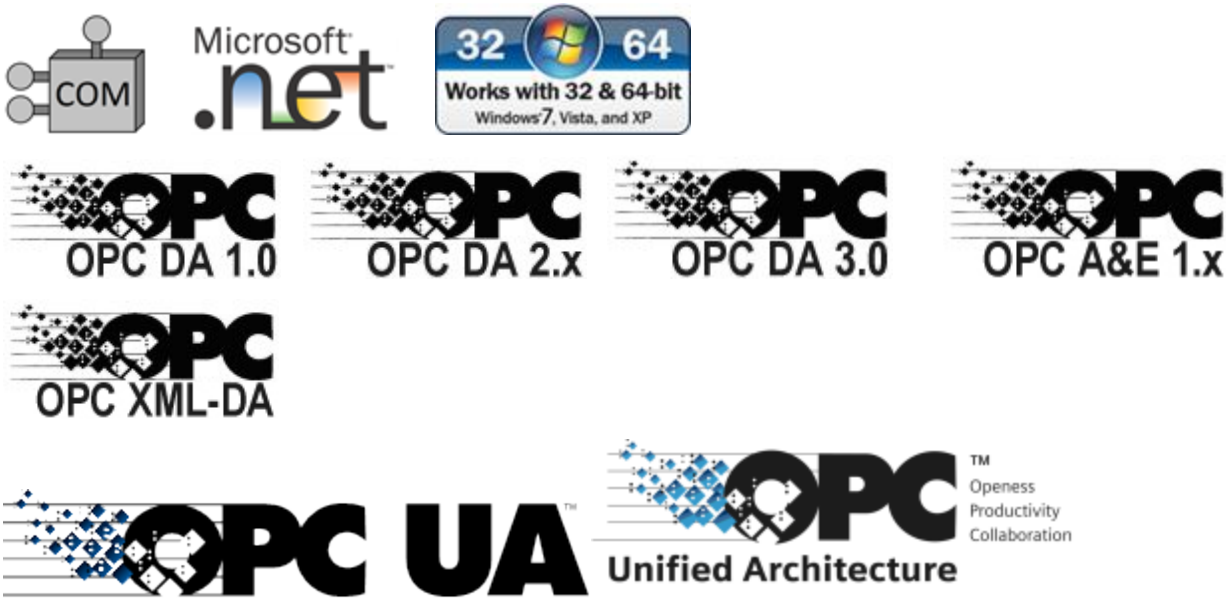
Version 2020.2

2 Introduction

Are you having difficulties incorporating the OPC data into your solution? Need to do it quickly and in quality? If so, OPC Data Client comes to the rescue.

OPC Data Client is a radically new approach to access OPC data. Traditionally, OPC programming required complicated code, no matter whether you use OPC custom or automation interfaces, or OPC Foundation SDKs. In OPC "Classic" world (COM/DCOM based), OPC Server objects must be instantiated, OPC Group objects must be created and manipulated, OPC Items must be added and managed properly, and subscriptions must be established and maintained. Other hurdles exist in OPC Unified Architecture (OPC-UA), such as proper server endpoint discovery, security policy negotiation, etc. Too many lines of error-prone code must be written to achieve a simple goal – reading or writing a value, or subscribing to value changes.

OPC Data Client is a set of components that simplify the task of integrating OPC into applications. Reading a value from an OPC server, writing a data value, or subscribing to data changes can be achieved in just one or two lines of code! Receiving alarms from OPC Alarms and Events server is also easy.



Notes:

- This documentation is for OPC Data Client version 2020.2. This is known as the *external version* of the product. The *internal version* number is 5.58.

2.1 Terminology

OPC Data Client is a name for the product as a whole. It can be used from various languages and environments.

Parts of the product can be differentiated by the kind of OPC specifications they deal with. OPC Data Client-Classic refers to parts of the product that deal with "Classic" OPC specifications, based on COM/DCOM (or OPC XML). It can be used

from COM tools or from .NET. Similarly, the OPC Data Client-UA referts to parts of the product that deal with OPC Unified Architecture (OPC UA) specifications, and it also supports both .NET and COM development targets.

The other way to differentiate the parts of the product is by the principal development technology they are aiming at: Either Microsoft .NET, or Microsoft .COM (so-called OLE Automation). When we need such disctinction, we refer to there product parts as OPC Data Client.NET or OPC Data Client-COM, respectively. When necessary, the differing text is marked with corresponding COM or .NET icon.

The following table explains the product part names, and how they relate to OPC specifications they cover, and to the development tools platform that can be used with them.

Development Tools/Target Platform:		**Microsoft .NET**	**Microsoft COM**
OPC Specifications			
OPC "Classic" (DCOM based)	OPC Data Access (OPC-DA)	OPC Data Client-Classic OPC Data Client.NET	OPC Data Client-Classic OPC Data Client-COM
	OPC Alarms and Events (OPC-A&E)		
OPC XML-DA (XML Data Access)			
OPC Unified Architecture (OPC-UA), for Data Access, Alarms & Conditions		OPC Data Client-UA OPC Data Client.NET	OPC Data Client-UA OPC Data Client-COM
OPC UA PubSub		OPC Data Client-UA OPC Data Client.NET	OPC Data Client-UA OPC Data Client-COM

The OPC XML-DA support in OPC Data Client.NET is transparently integrated into the component. Wherever this document refers to "OPC Data Access", it also usually applies to OPC XML-DA.

All products/product parts are delivered in a single installation package.

The main component of OPC Data Client is called EasyOPC, and it allows programmatic client access to OPC servers, Other parts of OPC Data Client are user interface objects (dialogs and controls), extensions, tools, and so on.

2.2 Product Editions

Following table will help you to understand the features of different OPC Data Client editions, and assist in selecting the edition that is best for you.

Product Edition:		**Standard**	**Professional**	**Enterprise** (new in 2019.2)	**Ultimate**	**Trial (free)**
Unlimited run time	The trial version delivers valid data for **30 minutes** since you have started your application. You then need to re-	✓	✓	✓	✓	×

	start the process in order to receive valid data again, and so on. Paid editions have **no such limitation**. *Contact us if you need a longer run time for your evaluation.*					
OPC Specifications						
OPC Data Access (4)	OPC-DA is the original specification. It is commonly used to move real-time data from PLCs, DCSs, and other control devices to HMIs and other clients, for display purposes and further processing.	✓	✓	✓	✓	✓
OPC XML-DA	OPC XML-DA specification builds on the OPC-DA specifications to communicate data in XML. It incorporates SOAP and Web services.	×	✓	✓	✓	✓
OPC Alarms and Events (4)	OPC-A&E provides alarm and event notifications on demand (in contrast to the continuous data flow of Data Access). These include process alarms, operator actions, informational messages, and tracking/auditing	×	✓	✓	✓	✓

	messages.					
OPC Unified Architecture (Generic/Data)	OPC-UA is the next generation of OPC. It brings the existing specifications together with a unified data model and uses Web services, rather than Microsoft COM, for messaging.	✓	✓	✓	✓	✓
OPC Unified Architecture (Methods)	The Method Service Set in OPC Unified Architecture allows the client to make function calls on OPC UA objects.	×	✓	✓	✓	✓
OPC Unified Architecture (Alarms & Conditions)	This standard deals with event subscriptions and notifications, Conditions, Dialog Conditions, and Alarms, including acknowledgement capabilities. Also includes specialized OPC UA Alarms and Conditions client object.	×	✓	✓	✓	✓
OPC UA PubSub	Subscriber functionality for OPC UA dataset messages distributed by message-oriented middleware, with support for resolution of parameters from the PubSub information model.	×	×	×	✓	✓

Development Models						
Imperative Programming Model (Section 5.1)	This is the traditional programming. You write code to instantiate objects, and call methods to perform the OPC operations.	✓	✓	✓	✓	✓
Live Binding Model (Section 5.3) [2]	Simply use the Visual Studio Designer to configure bindings between properties of visual or non-visual components, and OPC data. For Windows Forms and WPF, code-less.	×	✓	✓	✓	✓
Live Mapping Model (Section 5.2) [1]	Allows you to write objects that correspond logically to a functionality provided by whatever is "behind" the OPC data. You map your objects and members to OPC by annotating them with attributes.	×	×	✓	✓	✓
Reactive Programming Model (Section 5.4) [1]	High-level abstraction that allows you to compose asynchronous and event-based programs using data streams and LINQ-style query operators. Uses	×	×	×	✓	✓

	Microsoft Reactive Extensions (Rx) for .NET.					
Features						
Browsing Dialogs (Section 6.1.1)	Pre-made, highly configurable OPC browsing dialogs for Windows Forms (callable from WPF or COM applications as well). Provide interactive viewing and selection features.	×	✓	✓	✓	✓
Browsing Controls (Section 6.1.2) [(2)]	Configurable OPC browsing controls for Windows Forms. Can be combined with other controls on the form to achieve the desired functionality.	×	✓	✓	✓	✓
OPC UA Discovery (Section 5.1.2.3.5)	Local Discovery (LDS)	✓	✓	✓	✓	✓
	Network Discovery (LDS-ME)	×	✓	✓	✓	✓
	Global Discovery (GDS)	×	×	✓	✓	✓
Specialized Client Objects (Section 6.4)	- OPC UA Certificate Management Client (new in 2018.3) - OPC UA Global Discovery Client (new in 2018.3)	×	×	✓	✓	✓
	- OPC UA Publish-Subscribe Client. Provides access to PubSub information model	×	×	×	✓	✓

	in an OPC UA client, or to PubSub configuration stored in a file. (new in 2019.2)					
Extensions						
OPC UA Complex Data (Section 7.2.1)	Encodes and decodes OPC UA extension objects, according to data type dictionaries in the OPC UA information model.	×	×	✓	✓	✓
Development Platforms						
COM (3)	For languages or tools that supports Microsoft OLE Automation, e.g. Delphi, VB6, VBScript (such as in ASP), JScript, Perl, PHP, Python, VBA (such as in Excel), Visual FoxPro, REALbasic, Xbase, T-SQL, and many others. Also works from native C/C++.	×	✓	✓	✓	✓
.NET Framework (3)	For languages and tools based on Microsoft .NET Framework on Windows. Typical usage is from Visual Basic (VB.NET) or C#, but other languages can be used as well (e.g. PowerShell, managed C++, or F#).	✓	✓	✓	✓	✓

.NET Standard (new in 2018.3)	Multi-platform abstract API specification. Allows development in .NET languages for the .NET Core runtime on Windows or Linux.	×	✓	✓	✓	✓
Product Options (must be purchased as additional items)						
Excel Option (Section 12.1) (3)	Allows you to set up a communication link between Excel and any OPC server. It is possible to subscribe to and view real-time data, and also write the data back. No programming, macros or add-ins are necessary. With just drag-and-drop or copy-and-paste, you can create Microsoft Excel sheets with live, animated OPC data.	✓	✓	✓	✓	✓
StreamInsight Option (Section 12.2) (3)	Allows you to bring in streaming data from OPC sources into StreamInsight, analyze them and process them further, and even feed the results back to OPC servers. Data from OPC can be combined with data from multiple	×	×	✓	✓	✓

	other sources. You can monitor the data for meaningful patterns, trends and exceptions. Streaming OPC data can be analyzed and correlated while they are in-flight.					
Tools						
Connectivity Explorer (Section 11.2) (3)	Allows to navigate through hierarchy of OPC Data Access and OPC Unified Architecture servers and data nodes. You can subscribe to data changes, and view the results live.	✓	✓	✓	✓	✓
other tools	Launcher(3), License Manager(3) (GUI and Console), OPC UA Demo Publisher, OpcCmd Utility, UA Configuration Tool(3).	✓	✓	✓	✓	✓

Functionality of some Tools might be affected (limited) by the product edition in use.

Note (1): Only available under .NET Framework or .NET Standard development platforms.

Note (2): Only available under .NET Framework development platform (Windows).

Note (3): Only on Windows.

Note (4): Only available under COM or .NET Framework development platforms (Windows).

> ⚠ Licensing changes affecting the capabilities of various editions occur from time to time with releases of new OPC Data Client versions. Please re-check the edition capabilities if you are upgrading, or extending the upgrade assurance term.

The installation package and/or product packages or asemblies are the same, regardless of the product edition; the

enabled/allowed product capabilities are distinguished in the license key.

2.3 Development Platforms

OPC Data Client supports multiple development platforms. Not all functionality is available on every platform, however. The following table summarizes which functionalities are available where.

Development Platform:	**COM**	**.NET Framework**	**.NET Standard** (new in 2018.3)	**Excel** (with **Excel Option (Section 12.1)**)
Runtimes				
.NET Framework	✓	✓	✓	✓
.NET Core (new in 2018.3)	×	×	✓	×
Operating Systems				
Windows	✓	✓	✓	✓
Linux (new in 2018.3)	×	×	✓	×
OPC Specifications				
OPC Data Access	✓	✓	×	✓
OPC Alarms and Events	✓	✓	×	×
OPC XML-DA	✓	✓	✓	✓
OPC Unified Architecture - Generic/Data	✓	✓	✓	✓
OPC Unified Architecture - Alarms & Conditions	✓	✓	✓	×
OPC UA PubSub (new in 2019.2)	✓	✓	✓	×
Specialized OPC UA Facets				
Methods (Call service)	✓	✓	✓	× (1)
OPC UA Encodings (Client-Server)				
Binary	✓	✓	✓	✓
XML	✓	✓	×	✓
OPC UA Protocols (Client-Server)				
TCP (opc.tcp)	✓	✓	✓	✓
HTTP	✓	✓	×	✓
HTTPS	✓	✓	✓	✓
OPC UA PubSub Message Mappings				

UADP	✓	✓	✓	×
OPC UA PubSub Transport Protocol Mappings				
UDP (opc.udp)	✓	✓	✓	×
Ethernet (opc.eth)	✓	✓	✓	×
Development Models				
Imperative Programming Model (Section 5.1)	✓	✓	✓	× (1)
Live Binding Model (Section 5.3) (Windows Forms, WPF)	×	✓	×	×
Live Mapping Model (Section 5.2)	×	✓	✓	×
Reactive Programming Model (Section 5.4)	×	✓	✓	×
Real-time Data Spreadsheet	×	×	×	✓
Features				
Browsing Dialogs (Section 6.1.1)	✓	✓	×	× (1)
Browsing Controls (Section 6.1.2) (Windows Forms)	×	✓	×	×
Extensions				
OPC UA Complex Data (Section 7.2.1)	✓	✓	✓	✓
Distribution Formats				
Standalone Installer (Windows)	✓	✓	×	✓
NuGet Packages	×	✓	✓	×
Example Archives (new in 2018.3) (TGZ, ZIP formats)	×	✓	✓	×
Licensing				
License Location	See **Licensing Elements (Section 8.5)**			

Notes: (1) This functionality is not directly available on the development platform, but can be achieved by combining with VBA code (through COM platform).

OPC Data Client attempts to hide the differences caused by different implementations on various platforms and runtimes from the developer who uses it. It is not, however, always fully possible.

Throughout the documentation, we also try to pinpoint specific areas which are not available under some development platform or may work differently. There may be other, subtle differences under each platform as well. For example, the types of inner exceptions generated by the component may differ under various .NET Standard runtimes versus the code running on .NET Framework, unless the type of exception is documented. The error messages and other details of exceptions may differ as well.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

'Support' means that OPC Data Client is designed to work and tested on the environment, and vendor's technical support may be contacted for assistance with OPC Data Client on the environment. Support may be not available for operating systems, .NET runtimes and other dependencies that are past their active lifecycle at the time of support request, even if they were supported at the time of product release.

> .NET Standard is basically just an abstract API specification, with multiple runtimes (such as .NET Core) available that fulfill the specification. .NET Framework, on the contrary, is both an API specification and a concrete runtime. Listing them here side by side as two distinct target development platforms might seem improper, but it is not a misunderstanding of the concepts. It is intentional and appropriate, as this is how the OPC Data Client usages needs to be distinguished in terms of development and deployment.

2.3.1 Transitioning from .NET Framework to .NET Standard

If you have previously developed with OPC Data Client for .NET Framework and are now transitioning to .NET Standard, you may continue to use your existing code, as there are practically no coding differences between the two. There are, however, differences in other aspects, such as the supported functionality, tooling, installation, security, user interface, deployment, use of examples, etc.

For easier transition, we recommend that you read the following articles, which cover the areas that specifically need your attention.

(Section 2.3)

- **Development Platforms (Section 2.3)**: Overview of the development platforms, and how OPC Data Client differs on them.
- **.NET Runtimes (Section 2.4.1)**
- **Operating Systems (Section 2.4.2)**
- **Development Tools (Section 2.5)**
- **Installing NuGet Packages in JetBrains Rider (Section 3.1.2.3)**
- **Getting Started under .NET Standard (Section 3.3)**
- Security: **OPC UA Directory Certificate Stores (Section 4.13.1.4.1)**
- User Interface: **Console Interaction (Section 6.1.4.1)**
- Deployment: **Automated deployment, roll your own (Section 8.1.2)**; **Deployment Prerequisites (Section 8.2)**; **Installing Assemblies (.NET) (Section 8.3.1)**
- **Managed Resource License Store (Section 8.5.1)**
- **Supported OPC Specifications (Section 9.1)**; **Supported OPC UA Profiles (Section 9.1.1)**
- **.NET Standard Examples (Section 13.1.3)**

2.4 Requirements

Topics in this part describe OPC Data Client requirement in terms of:

- **.NET Runtimes (Section 2.4.1)**

- **Operating Systems (Section 2.4.2)**
- **Hardware (Section 2.4.3)**

Related to this, but not considered a requirement per se, are **Development Tools (Section 2.5)**. This is because OPC Data Client can be used from wide range of tools, and none is strictly required.

2.4.1 .NET Runtimes

.NET Framework, COM, and Excel Development

If you are developing under .NET Framework or COM development platform, and also for Excel development with the **Excel Option (Section 12.1)**, (at least) one of the following .NET runtimes must be installed on your development and production computers:

- .NET Framework 4.7
- .NET Framework 4.7.1
- .NET Framework 4.7.2
- .NET Framework 4.8

In This Topic

Note that

- .NET Framework 4.7 is included with Windows 10 version 1703 (Creators Update).
- .NET Framework 4.7.1 is included with Windows 10 version 1709 (Fall Creators update).
- .NET Framework 4.7.2 is included with Windows 10 version 1803 (April 2018 Update), version 1809 (October 2018 Update), Windows Server versions 1803 and 1809, and Windows Server 2019.
- .NET Framework 4.8 is included with Windows 10 version 1903 (May 2019 Update).

.NET Standard Development

If you are developing under .NET Standard development platform, (at least) one of the following .NET runtimes must be installed on your development computer:

- .NET Core 2.1.2 - 2.1.14
- .NET Core 3.1.0 - 3.1.1

On the production computers, you can either use framework-dependent deployment (FDD), and in such case one of the above runtimes must be installed on the production computer, or you can use a self-contained deplyment (SCD), in which case the .NET runtime you have selected does not have to be preinstalled on the production computer (but it is deployed along with your application). See .NET Core application deployment for more information.

.NET Core

On Linux, you may find Prerequisites for .NET Core on Linux page useful, in order to check the supported Linux distributions/versions, and what other dependencies exist. Related to this, you may also want to review:

- .NET Core versioning article
- .NET Core Supported OS Lifecycle Policy article
- Microsoft Support for .NET Core article

If you want to verify the version of .NET Core on your computer, use the ***dotnet --info*** command in the command prompt, and look for the **Version** field under **"Microsoft .NET Core Shared Framework Host"**, or (in newer versions) under **".NET Core runtimes installed:"** in the generated output. Do not use ***dotnet --version***, because this command only returns the version of the .NET Core command-line tools.

Other Runtimes for .NET Standard

Other .NET runtimes for the .NET Standard that are not listed here (Mono, Xamarin, UWP, .NET Framework) are not supported. It is possible that OPC Data Client will work on such .NET runtimes as well, but if you decide to use them, you are on your own... OPC Data Client may support them in the future.

Remarks

For a (incomplete) list of .NET Runtimes OPC Data Client has been tested with, consult Knowledge Base article QuickOPC Compatibility Test List.

The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

'Support' means that OPC Data Client is designed to work and tested on the environment, and vendor's technical support may be contacted for assistance with OPC Data Client on the environment. Support may be not available for operating systems, .NET runtimes and other dependencies that are past their active lifecycle at the time of support request, even if they were supported at the time of product release.

2.4.2 Operating Systems

.NET Framework, COM, and Excel Development

In This Topic

This chapter describes the operating systems supported if you are developing under .NET Framework or COM development platform, and also for Excel development with the **Excel Option (Section 12.1)**.

The product is supported on following **client operating systems**:

- Microsoft Windows 7 (x86 or x64) with SP1
- Microsoft Windows 10 (x86 or x64) Fall Creators Update (version 1709) or later

The product is supported on following **server operating systems**:

- Microsoft Windows Server 2012 (x64)
- Microsoft Windows Server 2012 R2 (x64)
- Microsoft Windows Server 2016 (x64)
- Microsoft Windows Server 2016 Core (x64) *(see **Server Core Systems (Section 2.4.2.1)** for limitations on this type of system)*

- Microsoft Windows Server 2019 (x64)
- Microsoft Windows Server 2019 Core (x64) *(see **Server Core Systems (Section 2.4.2.1)** for limitations on this type of system)*

On x64 platforms, OPC Data Client can run in 32-bit or 64-bit mode.

.NET Standard Development

This chapter describes the operating systems supported if you are developing under .NET Standard.

With **.NET Core** runtime, OPC Data Client supports following operating systems in general:

- Linux
- Microsoft Windows

The precise versions supported depend on the .NET runtime you have chosen, and on its version. Refer to the documentation of the particular .NET runtime for more information.

Linux

For .NET Core: See Microsoft articles .NET Core Supported OS Lifecycle Policy, and Linux Prerequisites.

Microsoft Windows

For .NET Core: See Microsoft articles .NET Core Supported OS Lifecycle Policy, and Windows Prerequisites.

Other Operating Systems for .NET Standard

Other operating systems on which .NET Core can run but are not listed here (macOS) are not supported. It is possible that OPC Data Client will work on such operating systems as well, but if you decide to use them, your are on your own... OPC Data Client may support them in the future.

Remarks

For a (incomplete) list of operating systems OPC Data Client has been tested with, consult Knowledge Base article QuickOPC Compatibility Test List.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

'Support' means that OPC Data Client is designed to work and tested on the environment, and vendor's technical support may be contacted for assistance with OPC Data Client on the environment. Support may be not available for operating systems, .NET runtimes and other dependencies that are past their active lifecycle at the time of support request, even if they were supported at the time of product release.

2.4.2.1 Server Core Systems

Some Microsoft Windows Server systems are categorized or can be installed as **Server Core** systems. They have no graphical user interface for management, and have some other limitations too.

OPC Data Client does not support development work on Server Core systems, but you can run programs created with OPC Data Client on them.

You should use the Production Installer (or your own installation method for the application your are creating) to install OPC Data Client on the Server Core system. If you attempt to run the Full Installer on the Server Core, it will show a warning message. If you proceed with the installation using the Full Installer anyway, the product will install, and will work well for production purposes (runtime), but some features will not work. For example, no links installed by the Setup program will be functional, because they rely on the shell functionality that is not present in Server Core. No shortcuts are installed onto desktop, and to the Start menu (as there is no such thing on Server Core).

The Server Core systems are meant for programs with no graphical user interface. If you run a GUI program developed with OPC Data Client on a Server Core system, it will most likely work - but that's not what the system was meant to be for. Ideally, your programs for Server Core should be services, Web apps, console applications, and such. There are also some limitations to what the GUI parts of OPC Data Client can do on Server Core. For example:

- Launching a browser by clicking on a URL in the controls or dialogs provided by OPC Data Client will not work.
- The ComputerBrowserDialog dialog (using Windows Shell) does not work.

> If you are scratching your head trying to get the installation files onto the Server Core box, see e.g. Getting installation files onto Server Core.

'Support' means that OPC Data Client is designed to work and tested on the environment, and vendor's technical support may be contacted for assistance with OPC Data Client on the environment. Support may be not available for operating systems, .NET runtimes and other dependencies that are past their active lifecycle at the time of support request, even if they were supported at the time of product release.

2.4.3 Hardware

OPC Data Client minimal hardware requirements are the same as those for the operating system and the development tools you are using.

For installation on Windows using the Full Installer, you should have at least 680 MB of free hard disk space before installing OPC Data Client.

2.5 Development Tools

In This Topic

Development Tools under .NET Framework

The development tools we have targeted primarily are:

- Microsoft Visual Studio 2017 (all editions, including Community).
- Microsoft Visual Studio 2019 (all editions, including Community).

Microsoft Visual Studio 2013 and 2015 may still work for many tasks, but we do not test for that.

The .NET Framework projects you create need to target .NET Framework 4.7 or higher.

In OPC Data Client.NET and OPC Data Client-UA for .NET, the available examples show how the components can be used from C#, Visual Basic.NET, and managed C++. Windows Forms, ASP.NET pages, console applications, and WPF applications are all supported.

Development Tools under .NET Standard

The development tools we have targeted primarily are:

- .NET Core CLI Tools
- JetBrains Rider 2019.1
- Visual Studio 2019
- Visual Studio Code (with the C# extension)

Other tools are likely to work as well. Also, some older versions of the development tools may work; we do not, however, test with them or support them.

When targeting the .NET Core runtime, .NET Core SDK 2.1.400 or higher is needed for development.

Development Tools under COM

In OPC Data Client-COM and OPC Data Client-UA for COM, the available examples show how the components can be used from Visual Basic (VB), C++, VBScript (e.g. in ASP, or Windows Script Host), JScript, Object Pascal (in Delphi), PHP, Visual Basic for Applications (VBA, e.g. in Excel), Visual FoxPro (VFP), and other tools. Any tool or language that supports COM Automation is supported.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

2.6 Licensing

OPC Data Client is a licensed product. You must obtain a license to use it in development or production environment. The legal aspect of the OPC Data Client licensing requires you to comply with the terms of its End User License Agreement (EULA). The technical aspect of the OPC Data Client licensing requires that for both the development and runtime purposes, you install or or otherwise provide a license key file that you have obtained with the OPC Data Client purchase (unless you are evaluating the product, in which case the built-in trial license can be used).

The license key is in a file, typically with .BIN or .TXT extension.

When you develop for .NET Framework, COM or Excel platform, the licenses are installed and managed using a **License Manager (Section 11.1)** utility (GUI, or console-based **LMConsole Utility (License Manager Console) (Section 11.1.1)**), described further in this document.

When developing for .NET Framework or .NET Standard platform, you can embed the license key in a managed resource directly in your program. For details, see **Managed Resource License Store (Section 8.5.1)**.

Trial License

For evaluation purposes, you are granted a trial license, which is in effect if no other license is available. The OPC Data Client.NET, OPC Data Client-UA and OPC Data Client-COM parts are licensed together.

With the trial license, the components only provide valid OPC data for 30 minutes since the application was started. After this period elapses, performing OPC operations will return an error. Restarting the application gives you additional 30 minutes, and so on. If you need to evaluate the product but the default trial license is not sufficient for your purposes, please contact the vendor or producer, describe your needs, and a special evaluation license may be provided to you.

Other Licensing Considerations

When using OPC Data Client, you also need to comply with **Licenses for 3rd-party Material (Section 2.6.1)** included with the product.

License to code from OPC Foundation

OPC Data Client includes code from OPC Foundation, such as OPC UA stack. Some of that code is released under dual-license model: RCL (Reciprocal Community License) applies to OPC Foundation corporate members, while GPL 2.0 applies to other users. Since OPC Data Client has been developed and is distributed by an OPC Foundation corporate member, GPL 2.0 does not apply to users of OPC Data Client.

3rd-party licenses when targeting .NET Standard

> ⚠ When you target **.NET Standard** development platform (see **Development Platforms (Section 2.3)**), OPC Data Client assemblies depend on (but do not ship with or include) various 3rd-party NuGet packages (mainly from Microsoft, and OPC Foundation). When you reference any of the OPC Data Client NuGet packages, you also need to consent to licensing conditions of the dependent packages (and their dependencies, recursively). As with the material directly included with OPC Data Client, we have limited our choice of dependent packages to those with **highly permissive licenses** that will not limit you unnecessarily. It always remains your responsibility to review the precise license terms and comply with them.

Optional 3rd-party libraries under LGPL or other licenses

OPC Data Client includes some libraries licensed under GNU Lesser General Public License (LGPL) or other licenses. They are only needed in special cases described in the documentation. You do not have to include them in your deployment unless you need to actually use the specific features they support. When you do not include them, LGPL does not apply.

For example, the PacketDotNet and SharpPcap libraries are only necessary if you want to use OPC UA PubSub Ethernet mapping, or work with Wireshark capture files.

2.6.1 Licenses for 3rd-party Material

OPC Data Client uses 3rd-party licensed material from different sources, and this material is redistributed with OPC Data Client. By using OPC Data Client, you are also bound by the license conditions of any material redistributed with OPC Data Client. The licenses that apply are listed here.

We always consider the effects of including such material with the product, and try to minimize the use of 3rd party if possible. We attempt to select **highly permissive licenses** that will not limit you unnecessarily. We do not use copyleft licenses such as GPL that would restrict your work to the same license terms as they have.

There are two typical usage scenarios:

1. You install OPC Data Client and use it to develop software. In this case, the usage is covered by OPC Data Client and its documentation, and you normally do not have to do anything special to fulfill the license conditions for the material redistributed with OPC Data Client .

2. You redistribute and/or deploy the software that you have developed with OPC Data Client. In this case, the licenses for the redistributed material (below) typically do not require you to do more than to make a corresponding attribution in the software and/or its documentation.

   Some of the material redistributed with OPC Data Client is **never redistributed** with your own application, and some **may not be redistributed in your particular case**, depending on which elements of OPC Data Client you use in your application. In such cases you may skip the licenses that do not apply to your redistribution. For details on this, see the **Application Deployment (Section 8)** section (**Licenses for Redistributed Material (Section 8.5.3)**).

It always remains your responsibility to comply with the precise license conditions of OPC Data Client EULA and the material listed below.

⚠ Do not use the list below to assess which license conditions will apply to the project you develop with OPC Data Client. Material listed below applies to OPC Data Client as a whole, and contains many pieces that are only used in supporting, non-redistributable tools and utilities, or during development time. Such licenses do not apply to your project you develop. See **Licenses for Redistributed Material (Section 8.5.3)** for the list that applies to the projects developed with OPC Data Client..

In This Topic

16x16 Free Application Icons

Original Author: Aha-Soft (http://www.small-icons.com/)

License URI: https://creativecommons.org/licenses/by-sa/3.0/legalcode

CommandLineUtils

https://www.nuget.org/packages/McMaster.Extensions.CommandLineUtils/

Apache License 2.0

Diagone Icons

Original Author: Yusuke Kamiyamane (http://p.yusukekamiyamane.com/)

License URI: https://creativecommons.org/licenses/by/3.0/legalcode

Discovery Icon Theme

Original Author: Hylke Bons

License URI: https://creativecommons.org/licenses/by-sa/3.0/legalcode

Fugue 16px Additional Icons

Original Author: Yusuke Kamiyamane (http://p.yusukekamiyamane.com/)

License URI: https://creativecommons.org/licenses/by/3.0/legalcode

Fugue 16px Additional Icons 2

Original Author: Yusuke Kamiyamane (http://p.yusukekamiyamane.com/)

License URI: https://creativecommons.org/licenses/by/3.0/legalcode

The original work has been modified.

Fugue 16px Icons

Original Author: Yusuke Kamiyamane (http://p.yusukekamiyamane.com/)

License URI: https://creativecommons.org/licenses/by/3.0/legalcode

Iconza Black Icons

Original Author: Turbomilk (http://turbomilk.com/)

License URI: https://creativecommons.org/licenses/by/3.0/legalcode

Krypton software

```
BSD License

For Krypton software

Copyright (c) 2006-2015, Component Factory Pty Ltd
```

Microsoft Automatic Graph Layout, MSAGL

MQTTnet

MIT License

MQTTnet Copyright (c) 2016-2019 Christian Kratky

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Newtonsoft.Json

Newtonsoft.Json

The MIT License (MIT)

Copyright (c) 2007 James Newton-King

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PacketDotNet

PacketDotNet

```
PacketDotNet is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

PacketDotNet is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with PacketDotNet.  If not, see .

Copyright 2009 Chris Morgan

GNU Lesser General Public License Version 3: https://www.gnu.org/licenses/lgpl-3.0.txt
GNU General Public License Version 3: https://www.gnu.org/licenses/gpl-3.0.txt
```

Pixelbox Icons

Author: http://www.icojam.com/

Freeware.

SharpPcap

```
SharpPcap

SharpPcap is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

SharpPcap is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with SharpPcap.  If not, see .

Copyright 2008-2009 Chris Morgan

GNU Lesser General Public License Version 3: https://www.gnu.org/licenses/lgpl-3.0.txt
GNU General Public License Version 3: https://www.gnu.org/licenses/gpl-3.0.txt
```

Ravenna 3D Icons

Original Author: Double-J Design (http://www.doublejdesign.co.uk/)

License URI: https://creativecommons.org/licenses/by/3.0/legalcode

System.Net.Mqtt

UA Configuration Tool

Web Mini Icons

Original Author: Axialis Team (http://www.axialis.com/)

License URI: https://creativecommons.org/licenses/by/2.5/

3 Getting Started

This section of the documentation consists of following parts:

- **Installation (Section 3.1)**
- **Getting Started under .NET Framework (Section 3.2)**,
- **Getting Started under .NET Standard (Section 3.3)**, and
- **Getting Started under COM (Section 3.5)**.

Depending on your development platform, please refer to the corresponding part.

For Getting Started related to OPC UA PubSub, see

- **Getting Started with OPC UA PubSub under .NET (Section 3.4)**

For Getting Started articles for various product options, see

- for Excel Option: **Getting Started under Excel (Section 12.1.2)**, and
- for StreamInsight Option: **StreamInsight Option Installation and Getting Started (Section 12.2.2)**.

3.1 Installation

OPC Data Client comes in two *distribution formats*:

- **Setup Program (Section 3.1.1)**. This is what you get when you directly download the software from the Web site, or obtain it on a distribution media such as CD or flash disk. The Setup program is only available for Windows. Use the Setup program mainly if you are targeting .NET Framework, COM, or Excel development platform.
- **NuGet Packages (Section 3.1.2)** (NuGet is a package manager for development tools such as Visual Studio). The packages are hosted on www.nuget.org, and you need to use a NuGet client in order to access them. Use the OPC Data Client NuGet packages when you are targeting the .NET Standard as a development platform. In the absence of a setup program, OPC Data Client provides several smaller downloads that complement the NuGet packages - such as archive files with examples, etc.

The distribution formats are described further below in this chapter.

Included Software

The OPC Data Client setup program does not install other software on the target system.

Related products

Additional products exist to complement the base OPC Data Client offering. Check the options available with your vendor.

3.1.1 Setup Program

The Setup program is only available for Windows. Use the Setup program mainly if you are targeting .NET Framework, COM, or Excel development platform.

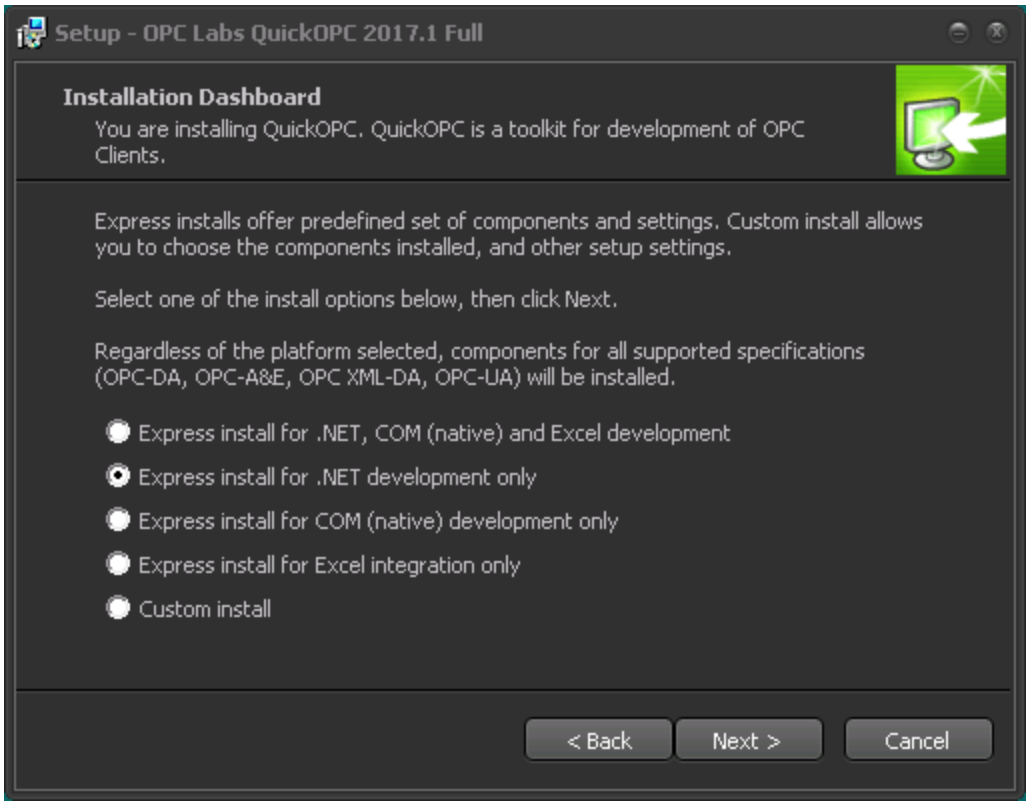
3.1.1.1 Running the Setup

The installation can be started by running the setup program. Just follow the on-screen instructions. The installation program requires that you have administrative privileges to the system.

The installation package file is code-signed, increasing security, improving trust, and providing better download experience from the Web. We use “COMODO RSA Code Signing CA” certificate. The name of signer is “CODE Consulting and Development, s.r.o.”. Certain other items in the product (such as the executables of the License Manager and OPC Kit Server, and the Help files, are also code-signed, using the same certificate).

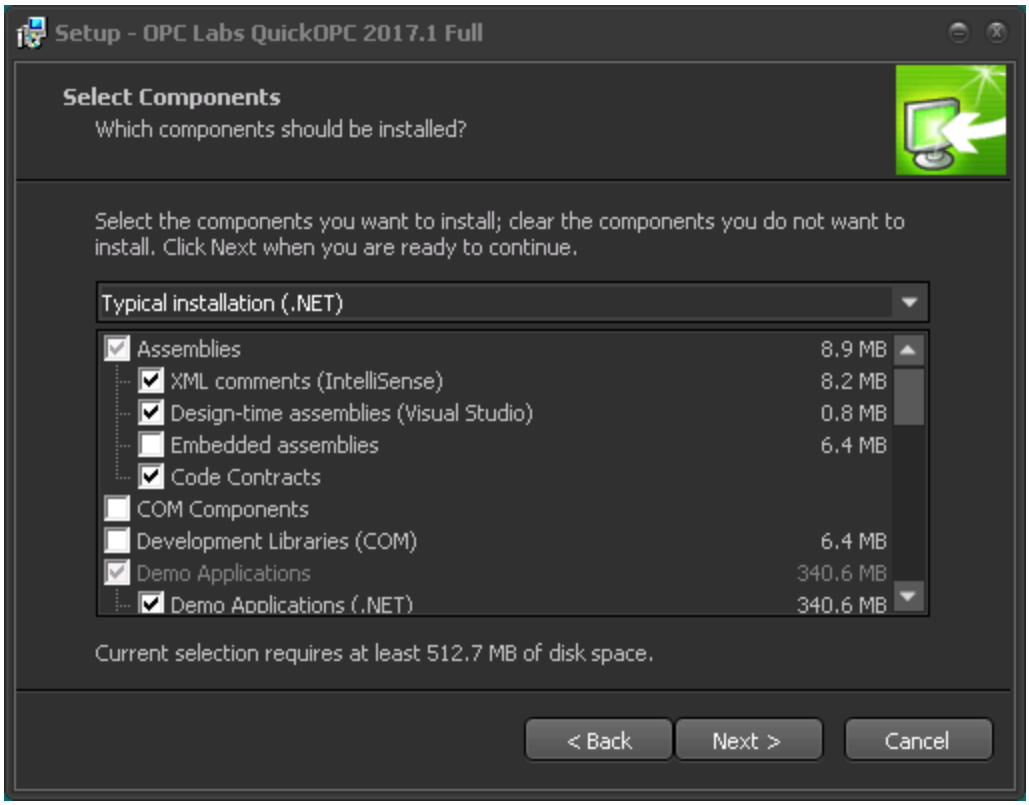
The installation wizard start with an introductory screen:

After an introductory screen, the setup wizard offers you the basic installation options:



For start, simply choose one of the “express” install options. If you decide to select “custom install”, the installation program then offers you several installation types, and also allows you to choose specifically which part of the product to install. In addition, with the “custom install”, you can also influence additional settings such as the destination location, and whether to automatically launch the License Manager utility.

For “custom install”, the component selection wizard page looks like this:



After the installation, the Start menu contains a number of hierarchically organized icons for various parts of the product. There is also a "Product Folder" shortcut, which opens the directory where all product files reside, making it easier to locate them.

When the installation is finished, it opens the Launcher application. This application is a switchboard that presents the same shortcuts as those available from the Start menu. It serves as a starting point for work with OPC Data Client.

The Setup program also places a Launcher application shortcut to the desktop, to the Programs group (Start menu), and to the Quick Launch bar (note that Quick Launch isn't visible by default; to enable it, see https://support.microsoft.com/en-us/kb/975784).

When the installation is finished, it also opens the Getting Started section of the documentation. You can also access the documentation and various tools from your Start menu, or the Launcher application that is installed together with the product.

When installing version OPC Data Client, it is strongly recommended that you first uninstall any version numbered between 5.00 and 5.31, instead of simply applying the installation over the previous version. The reason for this that the earlier versions installed the runtime assemblies into the Global Assembly Cache (GAC), whereas versions 5.32 and later do not do this by default. Having older assemblies in the GAC while developing with newer assemblies outside of the GAC is possible in principle, but without extra precautions, it can lead to all kinds of mix-ups and confusion.

The full standalone installer has a German localization. When the Setup programs detects that current user's UI language is German, it automatically switches to this localization.

If the needs of your organization require you to automate the installation of QuickOPC, it is possible to do so, using switches on the command line (see http://www.jrsoftware.org/ishelp/index.php?topic=setupcmdline). For additional information, see a related article in the Knowledge Base, Full installer Types, Components and Tasks. Note that for production purposes, it is not expected that you will automate the full installer in this way; you should be using the production installer instead (see the Application Deployment section).

For those who want to use the command-line interface, a "Command Prompt From Here" link is installed into the product folder. This link allows you to switch from the Windows Explorer to a CMD window with the current directory set to the chosen place, for quick experiments and exploration.

3.1.1.2 Uninstallation

The product includes an uninstall utility and registers itself as an installed application in Windows. It can therefore be removed easily from Control Panel. Alternatively, you can also use the Uninstall icon located in the product's group in the Start menu or the Launcher application.

You can uninstall OPC Data Client using following steps (*x.yz* denotes the version you have installed):

1. Go to **Windows Control Panel** and select "Uninstall a program". In the list of programs, locate "OPC Data Client *x.yz* Build *m.n*", and double-click on it. Alternatively, from **Windows Start menu**, select All programs -> OPC Labs -> OPC Data Client *x.y* -> Uninstall.

2. Press the "Yes" button when asked whether you are sure to uninstall.

These instructions are also available from the Start menu as a HTML document. To access them, select All programs -> OPC Labs -> OPC Data Client *x.y* -> Uninstall Instructions.

The uninstallation does not remove the OPC Data Client licenses present on the system in **Registry License Store (Section 8.5.2)**.

3.1.2 NuGet Packages

NuGet packages are for development on .NET Standard and .NET Framework platforms. Use the **Setup Program (Section 3.1.1)** when developing for COM or Excel platform.

In This Topic

NuGet (www.nuget.org) is the package manager for the Microsoft development platform including .NET. The NuGet client tools provide the ability to produce and consume packages. OPC Data Client NuGet packages are hosted in the NuGet Gallery (https://www.nuget.org/packages). The NuGet Gallery is the central package repository used by all package authors and consumers.

When you use NuGet to install a package, it copies the library files to your solution and automatically updates your project (add references, change config files, etc.). If you remove a package, NuGet reverses whatever changes it made so that no clutter is left.

For more details about OPC Data Client NuGet packages and instructions on how to use them, see:

- **List of Packages (Section 3.1.2.1)**
- **Installing NuGet Packages in Visual Studio (Section 3.1.2.2)**
- **Installing NuGet Packages in JetBrains Rider (Section 3.1.2.3)**
- **Installing NuGet Packages using .NET Core CLI Tools (Section 3.1.2.4)**

Limitations

Note that NuGet is primarily a tool for resolving build-time dependencies. The amount of functionality that you get through OPC Data Client NuGet packages is smaller than what OPC Data Client can actually do for you. If you want a full coverage of the features, you would be better off installing the product using the Setup program. Further below you will

find a list of differences between the two distribution forms.

The bulk of this documentation assumes that you have installed the product using the Setup program. You need to keep the limited scope of NuGet in mind when interpreting the documentation.

What is included in the NuGet packages:

- Runtime assemblies for all OPC specifications and programming models (for .NET Framework and .NET Standard target platforms).
- User interface components (OPC browsing dialogs and browsing controls for Windows Forms).
- IntelliSense support (XML comments).
- Examples in LINQPad. This means that whenever you reference one of the OPC Data Client NuGet packages in LINQPad, LINQPad will automatically add the OPC Data Client examples under its "Samples" tab.

What is only available from the Setup program or elsewhere (e.g. online):

- Support for COM development (VB6, PHP, Excel, Delphi and similar tools).
- Documentation and Help.
- Visual Studio integration, including Live Binding design-time support (codeless creation of OPC applications).
- Examples and Demo applications, Bonus.
- OPC Data Access simulation server, test tools.
- License Manager utility.
- StreamInsight Option, Excel Option.

3.1.2.1 List of Packages

Following OPC Data Client NuGet packages are available from www.nuget.org :

- **OpcLabs.QuickOpc**: OPC client components for all environments and project types (for .NET Framework and .NET Standard target platforms).
- **OpcLabs.QuickOpc.Forms**: Components that are specific for Windows Forms (can be partially used from WPF as well).

Following NuGet packages with OPC Data Client examples exist and are available from www.nuget.org :

- **OpcLabs.QuickOpc.Sample.CS**: Console-based QuickOPC examples in C# (source code).
- **OpcLabs.QuickOpc.Sample.VB**: Console-based QuickOPC examples in VB.NET (source code).

The strings in **bold** are the package IDs.

3.1.2.2 Installing NuGet Packages in Visual Studio

Note that before installing the NuGet packages, you need to create a Visual Studio project of the desired type. The NuGet packages get installed "into" the project.

In This Topic

Using Package Manager GUI

Using Package Manager Console

This text assumes that you are using Visual Studio 2017. With other tools, the procedures may be different, but should be reasonably similar.

In order to install a OPC Data Client NuGet package and reference it from your project using the **Package Manager GUI**:

1. In the Visual Studio Solution Explorer window, right-click the project, and select Manage NuGet Packages.
2. Make sure that **Package source** is set to "**nuget.org**".
3. Type ***QuickOPC*** into the Search box. A list of available OPC Data Client packages appears.
4. Click on the package you are interested in. Its details appear in the pane next (right) to it.
5. If needed, select a specific version of the package.
6. Press the Install button. A **Preview** windows appears, summarizing the changes that are about to be made.
7. Review the changes, and press the OK button.
8. The NuGet client will now update your project as needed. Among other things, it will download and copy over the OPC Data Client assemblies, and reference them.
9. A "**readme.txt**" file appears in the editor, with additional information. It is recommended that you read the file.
10. You can now start using the types from the referenced OPC Data Client assemblies.

If you do not see the packages in the NuGet Package Manager, make sure your project's framework is set to .NET Framework 4.7 or later, or .NET Core 2.1 and later.

Using Package Manager Console

This text assumes that you are using Visual Studio 2017. With other tools, the procedures may be different, but should be reasonably similar.

In order to install a OPC Data Client NuGet package and reference it from your project using **Package Manager Console**:

1. In the Visual Studio, select Tools -> NuGet Package Manager -> Package Manager Console.
2. Make sure that **Package source** is set to "**nuget.org**".
3. In the **Default project** drop-down, select the project.
4. Enter command ***Install-Package Id***, where *Id* is the package ID of the package to install (for example, **OpcLabs.QuickOpc**). Press ***Enter***.
5. The NuGet client will now update your project as needed. Among other things, it will download and copy over the OPC Data Client assemblies, and reference them.
6. A "**readme.txt**" file appears in the editor, with additional information. It is recommended that you read the file.
7. You can now start using the types from the referenced OPC Data Client assemblies.

3.1.2.3 Installing NuGet Packages in JetBrains Rider

Note that before installing the NuGet packages, you need to create a Rider project of the desired type. The NuGet packages get installed "into" the project.

This text assumes that you are using JetBrains Rider 2018.1. With other versions, the procedures may be different, but should be reasonably similar.

In order to install a OPC Data Client NuGet package and reference it from your project

1. Switch to the **NuGet** window in the Rider; if it is not visible, use View -> Tool Windows -> NuGet (***Alt+7***) command

from the menu.
2. In the **NuGet** window, switch to the **Packages** tab.
3. Make sure that package sources (the second control to the right of the search box) include "**nuget.org**".
4. Type ***QuickOpc*** into the search box.
5. Select the package you are interested in under **Available Packages** list in the left pane of the window. Its details appear in the pane next (right) to it.
6. If needed, select a specific version of the package in the upper part of the right pane.
7. In the right pane, press the green + button next to the name of your project.
8. Press the Yes button when asked to confirm the installation of the package.
9. The NuGet client will now update your project as needed. Among other things, it will download and copy over the OPC Data Client assemblies, and reference them.
10. You can now start using the types from the referenced OPC Data Client assemblies.

3.1.2.4 Installing NuGet Packages using .NET Core CLI Tools

Note that before installing the NuGet packages, you need to create a .NET Core project of the desired type. The NuGet packages get installed "into" the project.

> Use this procedure also when your development environment is Visual Studio Code.

In order to install a OPC Data Client NuGet package and reference it from your project:

1. Open a command prompt.
2. Navigate to the directory of your project.
3. Type the following: ***dotnet add package OpcLabs.QuickOpc***. This command adds OPC Data Client package reference to the project file.

   > The above command actually adds a reference to the package corresponding always to the very latest OPC Data Client version. If you want to be sure that you referencing a package for OPC Data Client 2020.2, use ***dotnet add package OpcLabs.QuickOpc --version 5.58*** command instead.

4. If you need to add more packages (see **List of Packages (Section 3.1.2.1)**), repeat the ***dotnet add package*** command, giving it the package Id as an argument.

3.2 Getting Started under .NET Framework

This section of the documentation consists of following parts:

- **Getting Started with OPC Classic under .NET Framework (Section 3.2.1)**, and
- **Getting Started with OPC UA under .NET (Section 3.2.2)**.

Depending on whether you develop OPC "Classic" or OPC Unified Architecture applications for Microsoft .NET Framework, please refer to the corresponding part.

3.2.1 Getting Started with OPC Classic under .NET Framework

OPC Data Client.NET allows you to develop using various development models. In this Getting Started chapter, we will present two of them:

- **Making a first OPC Data Client.NET application using Live Binding in Windows Forms (Section 3.2.1.1)** (without any coding), and
- **Making a first OPC Data Client.NET application using traditional coding (Section 3.2.1.2)** (imperative programming).

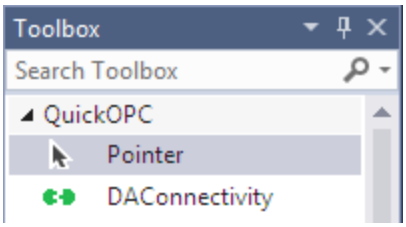
The Live Binding model also works with Windows Presentation Foundation (WPF); for a related Getting Started see **Making a first OPC UA application using traditional coding (Section 3.2.2.3)**.

Other development models available are *live mapping*, and *reactive programming (Rx)*.

3.2.1.1 Making a first OPC Classic application using Live Binding in Windows Forms

In this Getting Started procedure, we will create a Windows Forms application in C# or Visual Basic that subscribes to OPC Data Access (OPC-DA) item and continuously displays it changes. The live binding model allows achieving this functionality by simply configuring the OPC Data Client components, without any manual coding.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes .NET development.
2. Start Microsoft Visual Studio, and create new Visual C# or Visual Basic project, selecting "Windows Forms Application" template. **Make sure that in the upper part of the "New Project" dialog, the target framework is set to ".NET Framework 4.7" or later.**
3. Drag the DAConnectivity component from the "**QuickOPC**" tab of the **Toolbox** to the form's design surface.

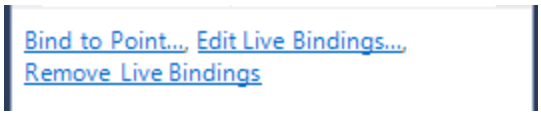


   Notice that three icons will appear below the form: One labeled "daConnectivity1", the second labeled "pointBinder1", and the third labeled "bindingExtender1".

   If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

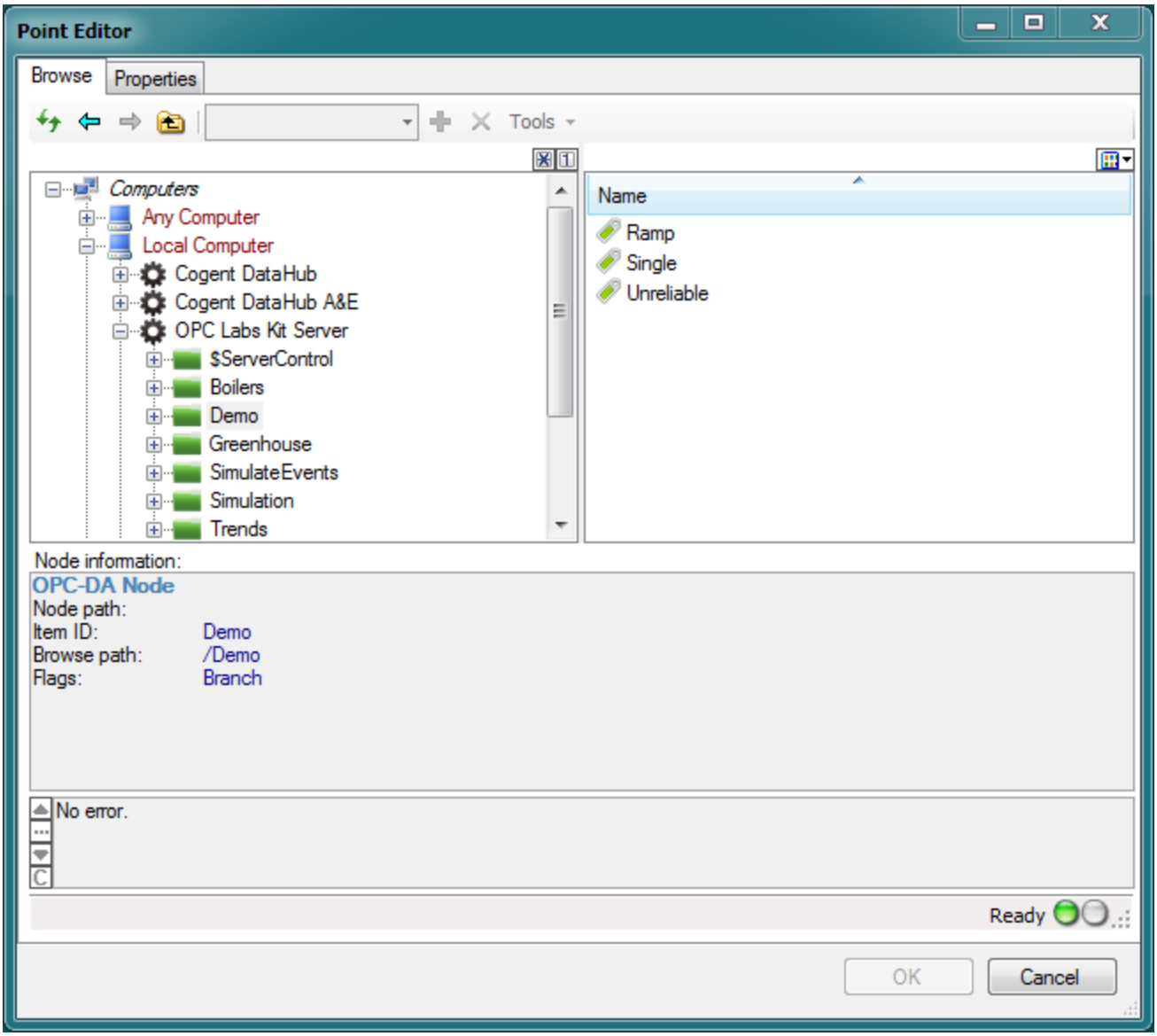
4. Drag a standard TextBox from the **Toolbox** to the form.
5. Right-click on the text box control on the form, and choose "Bind to Point" command.

   Alternatively, you can select the text box control first, and then choose the "Bind to Point" command in the Properties window; for this to work, however, the commands must be displayed in the **Properties** window.



If you do not see these commands, display them by right-clicking in the **Properties** window, and checking "Commands" in the menu.

6. In the "**Point Editor**" dialog, select Computers -> Local Computer -> OPC Labs Kit Server -> Demo -> Ramp, and press OK.



7. Build and run your application. You will see live OPC data changing in the text box. It is also possible to verify the effects of live binding without building the application: On the **bindingExtender** component placed on the form, select the "Design Online" command. This will cause Visual Studio to perform the live binding immediately. Use the "Design Offline" command to revert back to normal mode.

3.2.1.2 Making a first OPC Classic application using traditional coding

If you are interested in OPC Data Client.NET, your task will most certainly involve reading data from an OPC server. Here are a few steps that illustrate how to achieve that, using Microsoft Visual Studio, and a simple Windows Forms application in C#. The steps are quite similar if you are using other tools or a different programming language.

In This Topic

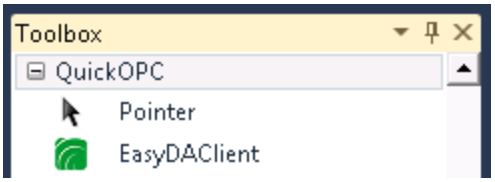
A Form to Read Value from OPC Data Access Server

We will create a form that will read a float value from the OPC server when loaded, and immediately display the formatted value in the text box on the form.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes .NET development.

2. Start Microsoft Visual Studio, and create new Visual C# project, selecting "Windows Forms Application" template. **Make sure that in the upper part of the "New Project" dialog, the target framework is set to ".NET Framework 4.7" or later.**

3. Instantiate the EasyDAClient component in the form: Drag the EasyDAClient component from the "**QuickOPC**" tab of the **Toolbox** to the form's design surface.



   An icon labeled "easyDAClient1" will appear below the form.

   If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

   Note: This is just shortcut way in Windows Forms to declaring and instantiating the component. In other environments, you can instantiate the component using the 'new' (C#) or 'New' (VB.NET) keyword, or any other way provided by the language or tool you are using.

4. Add a textbox to the form: Drag the TextBox component from the Toolbox to the form's design surface. The text box should appear on the form, and in the **Properties** window, you will see that it has been given a "textBox1" name.

5. Add handler for the Load event of the form: Select the form by clicking on its design surface. Then, in the **Properties** window, choose the "Events" view (orange lightning icon), and double-click the line with "Load" event (it is under **Behavior** group). A "Form1_Load" text in bold font will appear next to the event name, and a new event handler will be created for the Load event. The text editor will open for the code, and the caret will be placed into the Form1_Load method.

6. Add following code to the beginning of the **Form1.cs** file:

```
using OpcLabs.EasyOpc.DataAccess;
```

7. Write the event handler implementation. Add the following code to the body of Form1_Load method:

```
textBox1.Text = easyDAClient1.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Single").ToString();
```

8. Build the project.

9. Run the application and observe the results. The text box on the form will be filled with a float value we have read from the OPC server, using a single code line!

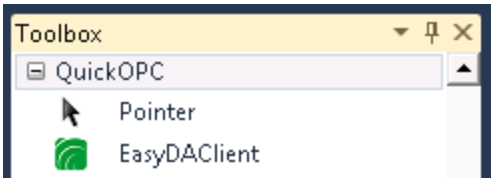
A Form to Read Value from OPC XML-DA Server

We will create a form that will read a float value from the OPC server when loaded, and immediately display the formatted value in the text box on the form.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes .NET development.

2. Start Microsoft Visual Studio, and create new Visual C# project, selecting "Windows Forms Application" template. **Make sure that in the upper part of the "New Project" dialog, the target framework is set to ".NET Framework 4.7" or later.**

3. Instantiate the EasyDAClient component in the form: Drag the EasyDAClient component from the "**QuickOPC**" tab of the **Toolbox** to the form's design surface.



   An icon labeled "easyDAClient1" will appear below the form.

   If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

   Note: This is just shortcut way in Windows Forms to declaring and instantiating the component. In other environments, you can instantiate the component using the 'new' (C#) or 'New' (VB.NET) keyword, or any other way provided by the language or tool you are using.

4. Add a textbox to the form: Drag the TextBox component from the Toolbox to the form's design surface. The text box should appear on the form, and in the **Properties** window, you will see that it has been given a "textBox1" name.

5. Add handler for the Load event of the form: Select the form by clicking on its design surface. Then, in the **Properties** window, choose the "Events" view (orange lightning icon), and double-click the line with "Load" event (it is under **Behavior** group). A "Form1_Load" text in bold font will appear next to the event name, and a new event handler will be created for the Load event. The text editor will open for the code, and the caret will be placed into the Form1_Load method.

6. Add following code to the beginning of the **Form1.cs** file:

```
using OpcLabs.EasyOpc.DataAccess;
```

7. Write the event handler implementation. Add the following code to the body of Form1_Load method:

```
textBox1.Text = easyDAClient1.ReadItemValue(
    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
    "Dynamic/Analog Types/Double").ToString();
```

8. Build the project.

9. Run the application and observe the results. The text box on the form will be filled with a float value we have read from the OPC server, using a single statement!

Notes

If you are targeting an environment different from Windows Forms (such as console application, Web, or WPF), your steps will be similar. Instead of dragging the EasyDAClient component from the Toolbox, simply reference the **OpcLabs.EasyOpcClassic** assembly, and create a new instance of **OpcLabs.EasyOpc.DataAccess.EasyDAClient** object.

3.2.1.3 Where do I go from here?

You now have a basic idea of how to use the OPC Data Client.NET. There are many additional resources that you can use to gain additional knowledge about the product. Here are some of them:

- Play with the Demo applications (accessible from the Start menu or the Launcher application) and explore the various functionalities that the product provides.
- Read the User's Guide.
- Study the Reference documentation.
- Review the additional examples and tools included with the product.

3.2.2 Getting Started with OPC UA under .NET

OPC Data Client-UA allows you to develop using various development models. In this Getting Started procedure, we will present two of them:

- **Making a first OPC Data Client-UA application using Live Binding in Windows Forms (without any coding) (Section 3.2.2.1)**,
- **Making a first OPC UA application using Live Binding in WPF (Section 3.2.2.2)**, and
- **Making a first OPC Data Client-UA application using traditional coding (Section 3.2.2.3)** (imperative programming).

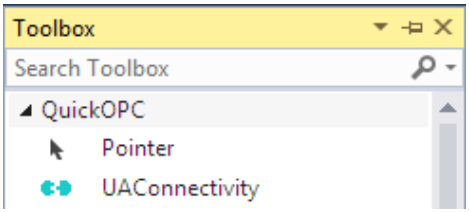
Other development models available are *live mapping*, and *reactive programming (Rx)*.

3.2.2.1 Making a first OPC UA application using Live Binding in Windows Forms

In this Getting Started procedure, we will create a Windows Forms application in C# or Visual Basic that subscribes to OPC Data

Access (OPC-DA) item and continuously displays it changes. The live binding model allows achieving this functionality by simply configuring the OPC Data Client components, without any manual coding.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes .NET development.
2. Start Microsoft Visual Studio, and create new Visual C# or Visual Basic project, selecting "Windows Forms Application" template. **Make sure that in the upper part of the "New Project" dialog, the target framework is set to ".NET Framework 4.7" or later.**
3. Drag the UAConnectivity component from the "**QuickOPC**" tab of the **Toolbox** to the form's design surface.

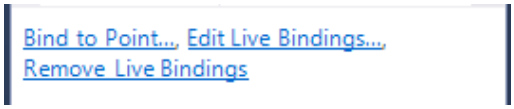


   Notice that three icons will appear below the form: One labeled "uaConnectivity1", the second labeled "pointBinder1", and the third labeled "bindingExtender1".

   If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

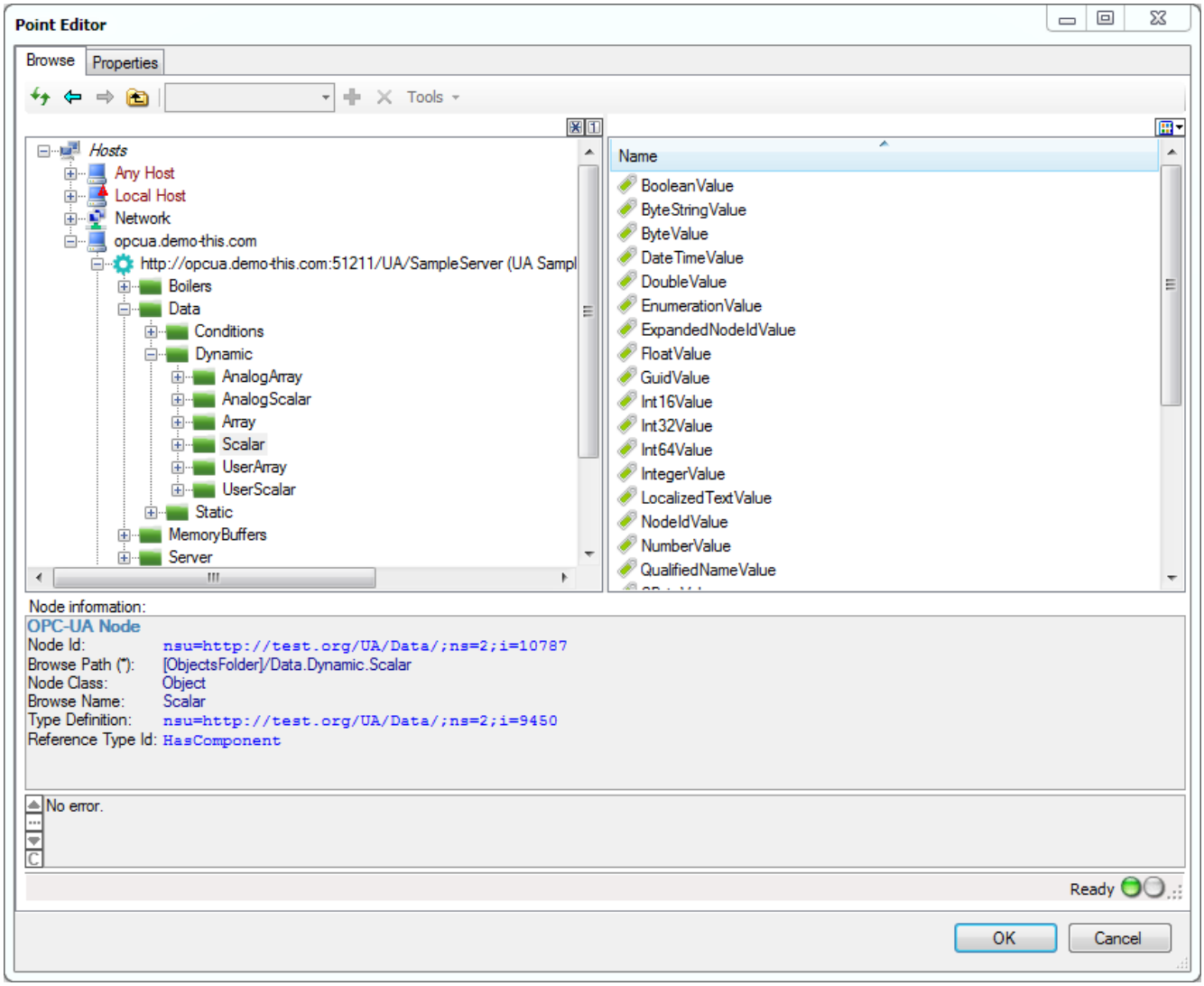
4. Drag a standard TextBox from the **Toolbox** to the form.
5. Right-click on the text box control on the form, and choose "Bind to Point" command.

   Alternatively, you can select the text box control first, and then choose the "Bind to Point" command in the Properties window; for this to work, however, the commands must be displayed in the **Properties** window.



   If you do not see these commands, display them by right-clicking in the **Properties** window, and checking "Commands" in the menu.

6. In the "Point Editor" dialog, select Hosts -> opcua.demo-this.com -> http:// opcua.demo-this.com:51211/UA/Sample Server (UA Sample Server) -> Data -> Dynamic -> Scalar -> Int32Value, and press OK.



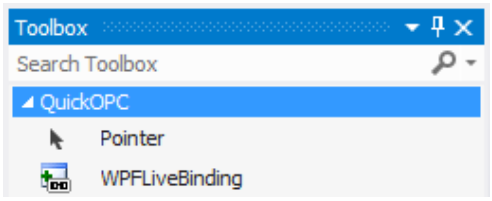
7. Build and run your application. You will see live OPC-UA data changing in the text box. It is also possible to verify the effects of live binding without building the application: On the **bindingExtender** component placed on the form, select the "Design Online" command. This will cause Visual Studio to perform the live binding immediately. Use the "Design Offline" command to revert back to normal mode.

3.2.2.2 Making a first OPC UA application using Live Binding in WPF

In this Getting Started procedure, we will create a WPF application in C# or Visual Basic that subscribes to OPC Data Access (OPC-DA) item and continuously displays it changes. The live binding model allows achieving this functionality by simply configuring the OPC Data Client components, without any manual coding.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes .NET development.
2. Start Microsoft Visual Studio, and create new Visual C# or Visual Basic project, selecting "WPF Application" template. **Make sure that in the upper part of the "New Project" dialog, the target framework is set to ".NET Framework 4.7" or later.**

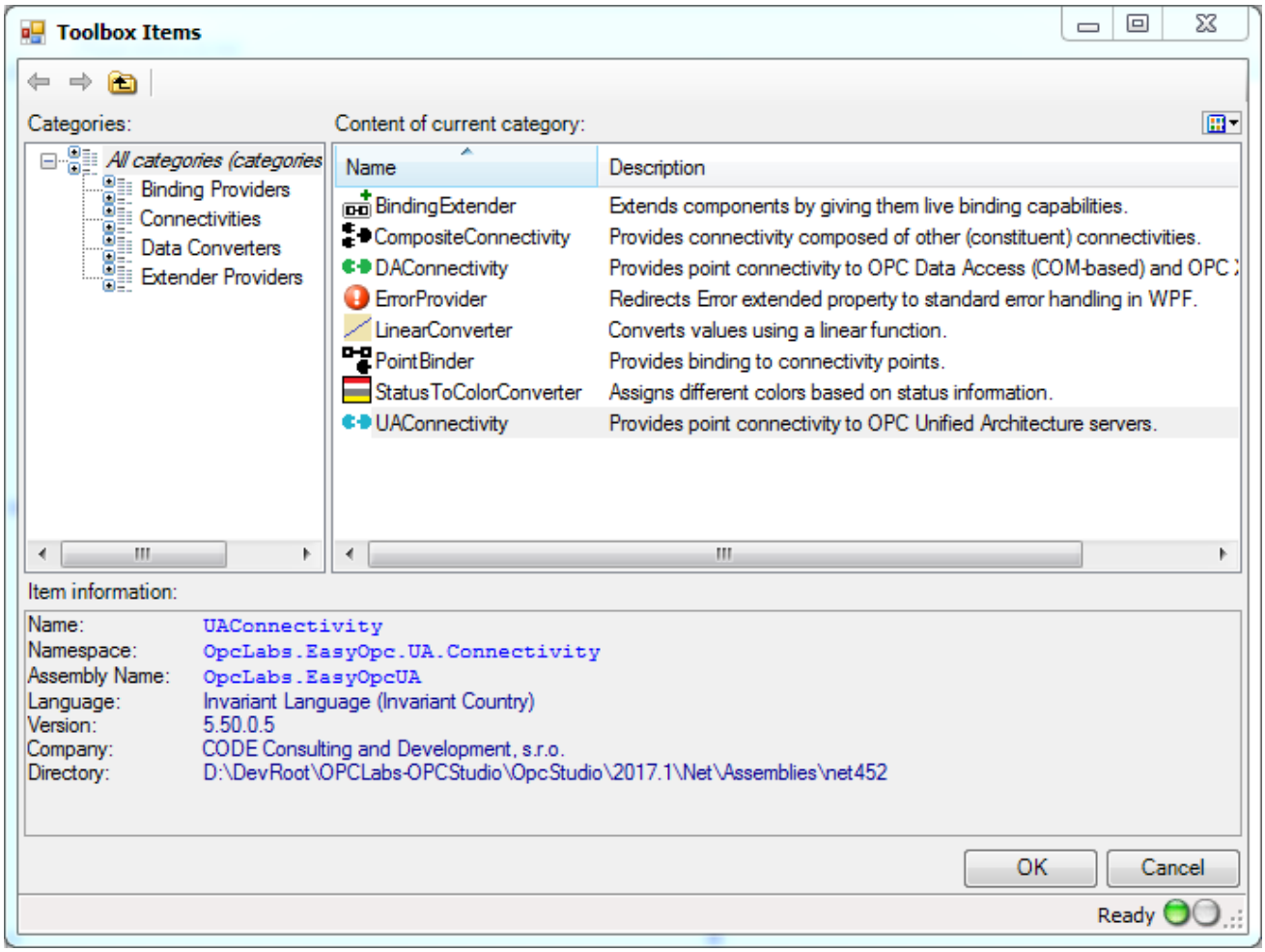
3. Drag the WPFLiveBinding component from the "**QuickOPC**" tab of the **Toolbox** to the window's design surface.



If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

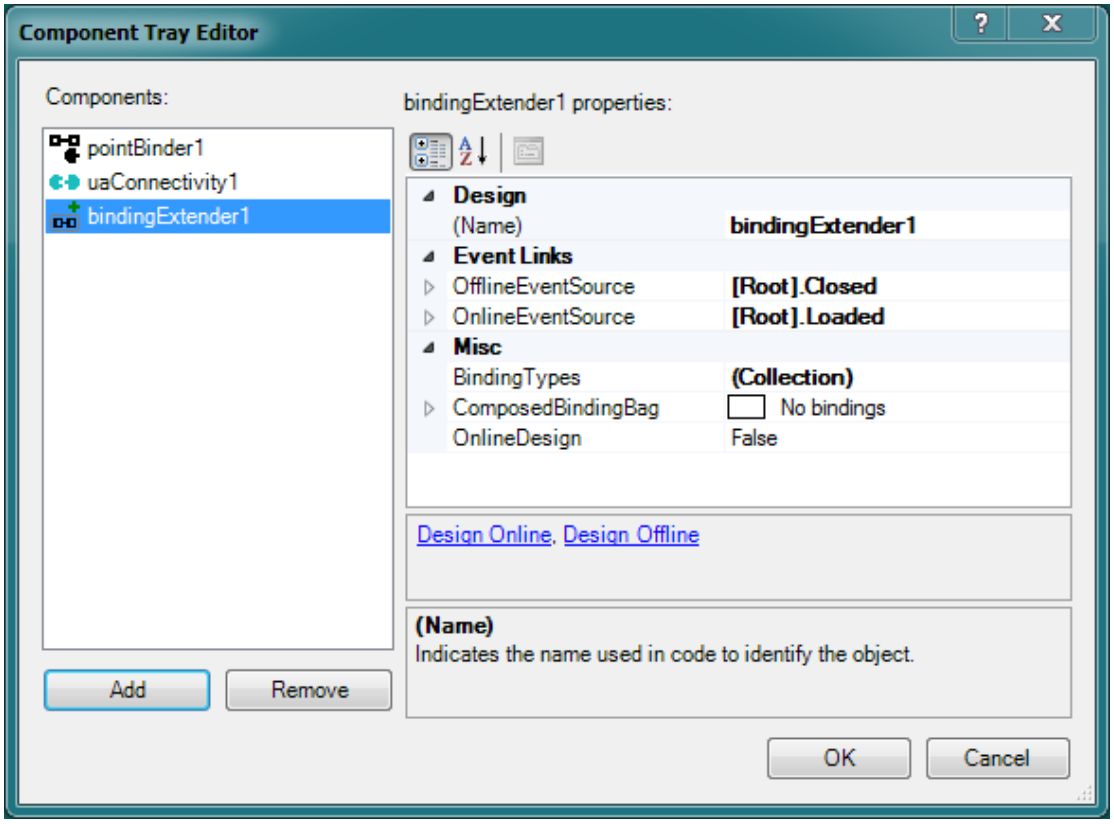
A **Component Tray Editor** dialog with an empty list of components will appear.

4. In the **Component Tray Editor** dialog, press the Add button. A **Toolbox Items** dialog will appear:



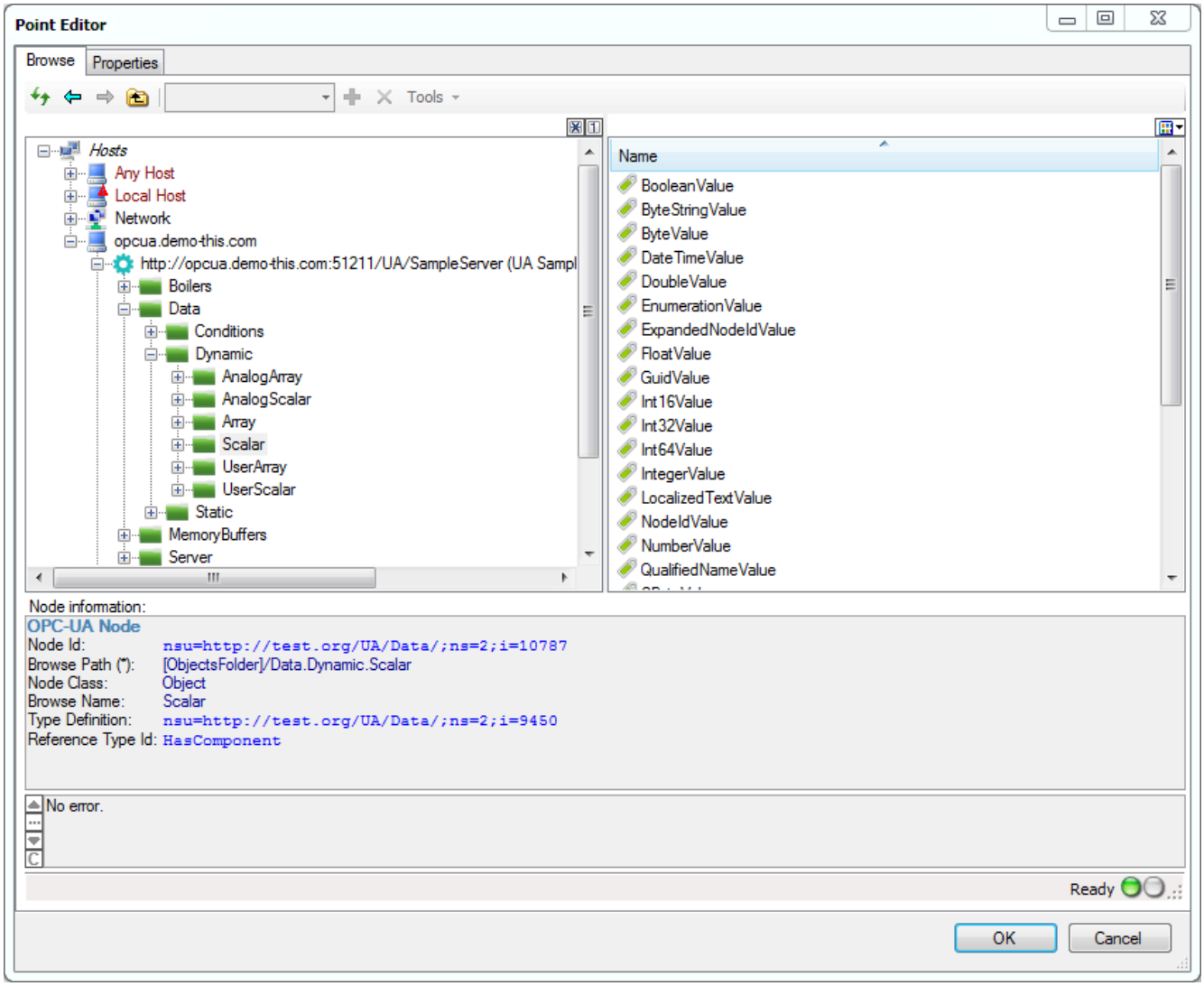
5. Select the UAConnectivity component from the **Content of current category list** of the **Toolbox Items** dialog, and press OK.

Notice that three entries will appear in the **Components** list in the **Component Tray Editor**: One labeled "uaConnectivity1", the second labeled "pointBinder1", and the third labeled "bindingExtender1":



6. In the **Component Tray Editor** dialog, press the OK button.
7. Drag a standard TextBox from the Visual Studio **Toolbox** to the designed window.
8. Right-click on the text box control on the designed window, and choose "Bind to Point" command.

   Alternatively, you can select the text box control first, and then expand the Live Binding category in the **Properties** windows, and next to the Live.Extensions property, choose the "Bind to Point" command there.
9. In the "Point Editor" dialog, select Hosts -> opcua.demo-this.com -> http:// opcua.demo-this.com:51211/UA/Sample Server (UA Sample Server) -> Data -> Dynamic -> Scalar -> Int32Value, and press OK.



10. Build and run your application. You will see live OPC-UA data changing in the text box. It is also possible to verify the effects of live binding without building the application: Right-click anywhere in the designed window area, and select the Online Design Live Binding command (this is an on-off command, with "on" state indicated by a check mark next to the menu item). Alternatively, expand the Live Binding category in the **Properties** window, and next to the Live.Extensions property, check the **Online Design Live Binding** box. This will cause Visual Studio to perform the live binding immediately. Use the same command to revert back to normal mode.

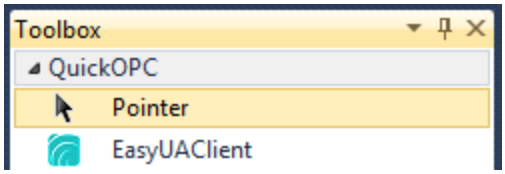
3.2.2.3 Making a first OPC UA application using traditional coding

> If you are targeting the .NET Standard development platform and not the .NET Framework, see **Getting Started under .NET Standard (Section 3.3)** instead.

If you are interested in OPC Data Client-UA, your task will most certainly involve reading data from an OPC server. Here are a few steps that illustrate how to achieve that, using Microsoft Visual Studio, and a simple Windows Forms application in C#. The steps are quite similar if you are using other tools or a different programming language.

We will create a form that will read a float value from the OPC server when loaded, and immediately display the formatted value in the text box on the form.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes .NET development.
2. Start Microsoft Visual Studio, and create new Visual C# project, selecting "Windows Forms Application" template. **Make sure that in the upper part of the "New Project" dialog, the target framework is set to ".NET Framework 4.7" or later.**
3. Instantiate the EasyUAClient component in the form: Drag the EasyUAClient component from the "**QuickOPC**" tab of the **Toolbox** to the form's design surface.



   An icon labeled "easyUAClient1" will appear below the form.

   If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

   Note: This is just shortcut way in Windows Forms to declaring and instantiating the component. In other environments, you can instantiate the component using the 'new' (C#) or 'New' (VB.NET) keyword, or any other way provided by the language or tool you are using.

4. Add a textbox to the form: Drag the TextBox component from the **Toolbox** to the form's design surface. The text box should appear on the form, and in the **Properties** window, you will see that it has been given a "textBox1" name.
5. Add handler for the Load event of the form: Select the form by clicking on its design surface (make sure that you do not have any other control selected). Then, in the **Properties** window, choose the "Events" view (orange lightning icon), and double-click the line with "Load" event (it is under Behavior group). A "Form1_Load" text in bold font will appear next to the event name, and a new event handler will be created for the Load event. The text editor will open for the code, and the caret will be placed into the Form1_Load method.
6. Add following code to the beginning of the **Form1.cs** file:

```
using OpcLabs.EasyOpc.UA;
```

7. Write the event handler implementation. Add the following code to the body of Form1_Load method:

```
textBox1.Text =
easyUAClient1.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853").ToString();
```

8. Build the project.
9. Run the application and observe the results. The text box on the form will be filled with a float value we have read from the OPC server, using a single code line!

If you are targeting an environment different from Windows Forms (such as console application, Web, or WPF), your steps will be similar. In place of Step 4, simply reference the OpcLabs.EasyOpcUA assembly, and create a new instance of **OpcLabs.EasyOpc.UA.EasyUAClient** object.

3.2.2.4 Where do I go from here?

You now have a basic idea of how to use the OPC Data Client-UA. There are many additional resources that you can use to gain additional knowledge about the product. Here are some of them:

- Play with the Demo application (accessible from the Start menu or the Launcher application) and explore the various functionalities that the product provides.
- Read the User's Guide.
- Study the Reference documentation.
- Review the additional examples and tools included with the product.

3.3 Getting Started under .NET Standard

This section of the documentation consists of following parts:

- **Getting Started under .NET Standard using .NET Core CLI Tools (Section 3.3.1)**,
- **Getting Started under .NET Standard using IDE (Section 3.3.2)**, and
- **Getting Started under .NET Standard using Visual Studio Code (Section 3.3.3)**.

Depending on your .NET runtime and development tool, please refer to the corresponding part.

3.3.1 Getting Started under .NET Standard using .NET Core CLI Tools

In this Getting Started, we will create a console application running in .NET Core that will read a value from an OPC server, and display the value on the console.

In This Topic

Prerequisites

- Supported operating system: **Linux** or **Windows**.
- **.NET Core SDK 2.1.103** (or possibly higher version): Linux instructions, Windows instructions.
- A text editor or code editor of your choice.
- Internet connection with access to www.nuget.org (for downloading NuGet packages) and www.opclabs.com (for connection to the public OPC UA server).

If you want to verify the version of .NET Core on your computer, use the ***dotnet --info*** command in the command prompt, and look for the **Version** field under **"Microsoft .NET Core Shared Framework Host"**, or (in newer versions) under **".NET Core runtimes installed:"** in the generated output. Do not use ***dotnet --version***, because this command only returns the version of the .NET Core command-line tools.

Console Application to Read Value from OPC Unified Architecture

Server

1. Open a command prompt.
2. Create a folder named **Hello** and navigate to the folder.
3. Type the following: ***dotnet new console***. This command creates a project file (**Hello.csproj**) and a main program file (**Program.cs**).
4. Type the following: ***dotnet add package OpcLabs.QuickOpc***. This command adds OPC Data Client package reference to the project file.

   > The above command actually adds a reference to the package corresponding always to the very latest OPC Data Client version. If you want to be sure that you referencing a package for OPC Data Client 2020.2, use ***dotnet add package OpcLabs.QuickOpc --version 5.58*** command instead.

5. Using a text editor, add following code to the beginning of the **Program.cs** file:

```
using OpcLabs.EasyOpc.UA;
```

6. In **Program.cs**, replace the body of the Main method by following code:

```
var client = new EasyUAClient();
object value = client.ReadValue(
    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
    "nsu=http://test.org/UA/Data/;i=10853");
Console.WriteLine(value);
```

7. Save the changes to **Program.cs**.
8. In the command prompt, type ***dotnet run***. This will build and launch the program. The value will be read from the OPC server and displayed on the console.

Console Application to Read Value from OPC XML-DA Server

1. Open a command prompt.
2. Create a folder named **HelloXml** and navigate to the folder.
3. Type the following: ***dotnet new console***. This command creates a project file (**HelloXml.csproj**) and a main program file (**Program.cs**).
4. Type the following: ***dotnet add package OpcLabs.QuickOpc***. This command adds OPC Data Client package reference to the project file.

   > The above command actually adds a reference to the package corresponding always to the very latest OPC Data Client version. If you want to be sure that you referencing a package for OPC Data Client 2020.2, use ***dotnet add package OpcLabs.QuickOpc --version 5.58*** command instead.

5. Using a text editor, add following code to the beginning of the **Program.cs** file:

```
using OpcLabs.EasyOpc.DataAccess;
```

6. In **Program.cs**, replace the body of the Main method by following code:

```
var client = new EasyDAClient();
object value = client.ReadItemValue(
    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
    "Dynamic/Analog Types/Double");
Console.WriteLine(value);
```

7. Save the changes to **Program.cs**.

8. In the command prompt, type ***dotnet run***. This will build and launch the program. The value will be read from the OPC server and displayed on the console.

3.3.2 Getting Started under .NET Standard using IDE

In this Getting Started, we will create a console application running in .NET Core that will read a value from an OPC server, and display the value on the console.

In This Topic

Prerequisites

- Supported operating system: **Linux** or **Windows**.
- **.NET Core SDK 2.1.103** (or possibly higher version): Linux instructions, Windows instructions.
- **Visual Studio**, or **JetBrains Rider**.
- Internet connection with access to www.nuget.org (for downloading NuGet packages) and www.opclabs.com (for connection to the public OPC UA server).

If you want to verify the version of .NET Core on your computer, use the ***dotnet --info*** command in the command prompt, and look for the **Version** field under **"Microsoft .NET Core Shared Framework Host"**, or (in newer versions) under **".NET Core runtimes installed:"** in the generated output. Do not use ***dotnet --version***, because this command only returns the version of the .NET Core command-line tools.

Console Application to Read Value from OPC Unified Architecture Server

1. Start the IDE (Integrated Development Environment) of your choice.

2. In this step, we will create a new .NET Core console application project.

   If you are using Visual Studio, select File -> New -> Project from the menu. In the **New Project** dialog, navigate to Installed -> Visual C# -> .NET Core, and select Console App (.NET Core). Press the OK button.

   If you are using JetBrains Rider, select **New Solution** on the **Welcome to JetBrains Rider** screen. In the **New Solution** dialog, select .NET Core -> Console Application. Verify the **Language** or change it to C#. Press the Create button.

3. In this step, we will add a reference to the **OpcLabs.QuickOpc** NuGet package.

   If you are using Visual Studio, switch to the **Solution Explorer** window, right-click on the project node

(ConsoleApp*n*), and select Manage NuGet Packages... command. In the **NuGet: *project*** window, switch to the **Browse** tab, and type ***OpcLabs.QuickOpc*** into the search box. Select the OpcLabs.QuickOpc package in the package list in the left pane of the window. In the right pane of the window, verify or change the package version next to the **Version** label. The version should be "Latest stable 5.58.*build*" (where *build* is a build number). Press the Install button.

If you are using JetBrains Rider, switch to the **NuGet** window; if it is not visible, use View -> Tool Windows -> NuGet (***Alt+7***) command from the menu. In the **NuGet** window, switch to the **Packages** tab, and type ***OpcLabs.QuickOpc*** into the search box. Select the OpcLabs.QuickOpc package under **Available Packages** list in the left pane of the window. In the right pane of the window, verify or change the package version next to the **Version** label. The version should be "5.58.*build*" (where *build* is a build number). Press the green + button next to the name of your project. Press the Yes button when asked to confirm the installation of the package.

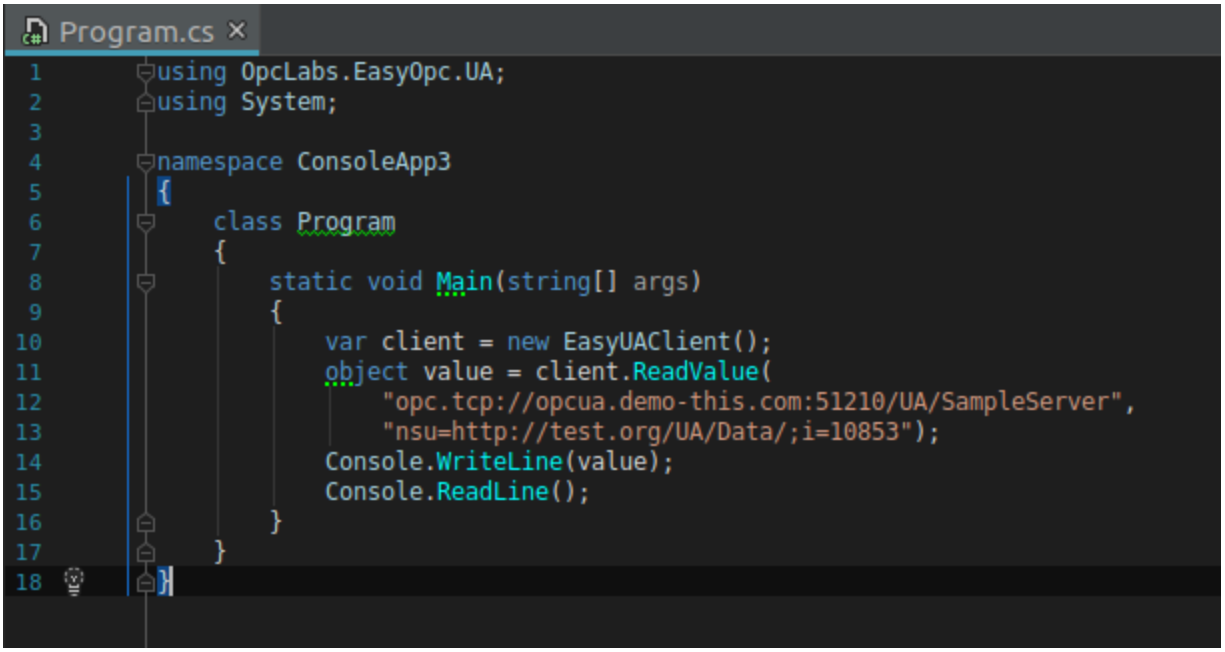
4. Open the **Program.cs** file, and add following code to the beginning of the file:

```
using OpcLabs.EasyOpc.UA;
```

5. In **Program.cs**, replace the body of the Main method by following code:

```
var client = new EasyUAClient();
object value = client.ReadValue(
    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
    "nsu=http://test.org/UA/Data/;i=10853");
Console.WriteLine(value);
Console.ReadLine();
```

The program may now look like this in the IDE:



6. If you are using Visual Studio, select Debug -> Start Debugging (***F5***) from the menu, or press the corresponding button on the toolbar.

If you are using JetBrains Rider, select Run -> Debug 'Default' (***F5***), or press the corresponding button on the toolbar. When the **Edit configuration** dialog appears, press the Debug button.

This will build and launch the program. The value will be read from the OPC server and displayed on the console.

Press ***Enter*** to exit the program.

Console Application to Read Value from OPC XML-DA Server

1. Start the IDE (Integrated Development Environment) of your choice.

2. In this step, we will create a new .NET Core console application project.

   If you are using Visual Studio, select File -> New -> Project from the menu. In the **New Project** dialog, navigate to Installed -> Visual C# -> .NET Core, and select Console App (.NET Core). Press the OK button.

   If you are using JetBrains Rider, select **New Solution** on the **Welcome to JetBrains Rider** screen. In the **New Solution** dialog, select .NET Core -> Console Application. Verify the **Language** or change it to C#. Press the Create button.

3. In this step, we will add a reference to the **OpcLabs.QuickOpc** NuGet package.

   If you are using Visual Studio, switch to the **Solution Explorer** window, right-click on the project node (ConsoleApp*n*), and select Manage NuGet Packages... command. In the **NuGet: *project*** window, switch to the **Browse** tab, and type ***OpcLabs.QuickOpc*** into the search box. Select the OpcLabs.QuickOpc package in the package list in the left pane of the window. In the right pane of the window, verify or change the package version next to the **Version** label. The version should be "Latest stable 5.58.*build*" (where *build* is a build number). Press the Install button.

   If you are using JetBrains Rider, switch to the **NuGet** window; if it is not visible, use View -> Tool Windows -> NuGet (***Alt+7***) command from the menu. In the **NuGet** window, switch to the **Packages** tab, and type ***OpcLabs.QuickOpc*** into the search box. Select the OpcLabs.QuickOpc package under **Available Packages** list in the left pane of the window. In the right pane of the window, verify or change the package version next to the **Version** label. The version should be "5.58.*build*" (where *build* is a build number). Press the green + button next to the name of your project. Press the Yes button when asked to confirm the installation of the package.

4. Open the **Program.cs** file, and add following code to the beginning of the file:

   ```
   using OpcLabs.EasyOpc.DataAccess;
   ```

5. In **Program.cs**, replace the body of the Main method by following code:

   ```
   var client = new EasyDAClient();
   object value = client.ReadItemValue(
       "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
       "Dynamic/Analog Types/Double");
   Console.WriteLine(value);
   ```

6. If you are using Visual Studio, select Debug -> Start Debugging (***F5***) from the menu, or press the corresponding button on the toolbar.

   If you are using JetBrains Rider, select Run -> Debug 'Default' (***F5***), or press the corresponding button on the toolbar. When the **Edit configuration** dialog appears, press the Debug button.

   This will build and launch the program. The value will be read from the OPC server and displayed on the console. Press ***Enter*** to exit the program.

3.3.3 Getting Started under .NET Standard using Visual Studio Code

⚠ This article is for development with Visual Studio Code, a multi-platform source code editor developed by Microsoft. For development with "full" Visual Studio (Windows-only) IDE, see **Getting Started under .NET Standard using IDE (Section 3.3.2)**.

In This Topic

Prerequisites

- Supported operating system: **Linux** or **Windows**.
- **.NET Core SDK 2.1.103** (or possibly higher version): Linux instructions, Windows instructions.
- **Visual Studio Code** (https://code.visualstudio.com/).
- **C# extension for Visual Studio Code**.
- Internet connection, including access to www.opclabs.com (for connection to the public OPC UA server)..

If you want to verify the version of .NET Core on your computer, use the ***dotnet --info*** command in the command prompt, and look for the **Version** field under **".NET Core Shared Framework Host"**, or (in newer versions) under **".NET Core runtimes installed:"** in the generated output. Do not use ***dotnet --version***, because this command only returns the version of the .NET Core command-line tools.

Console Application to Read Value from OPC Unified Architecture Server

1. Open a command prompt.
2. Create a folder named **Hello** and navigate to the folder.
3. Type the following: ***dotnet new console***. This command creates a project file (**Hello.csproj**) and a main program file (**Program.cs**).
4. Type the following: ***dotnet add package OpcLabs.QuickOpc***. This command adds OPC Data Client package reference to the project file.

   The above command actually adds a reference to the package corresponding always to the very latest OPC Data Client version. If you want to be sure that you referencing a package for OPC Data Client 2020.2, use ***dotnet add package OpcLabs.QuickOpc --version 5.58*** command instead.

5. Type the following: ***code .*** (note the dot '.' as a parameter to the command.) Visual Studio Code starts and its window opens.
6. At this point, you might be asked **"Required assets to build and debug are missing from 'vscode'. Add them?"**. If so, press the Yes button in the popup window.
7. A notification will appear **"There are unresolved dependencies from 'Hello.csproj'. Please execute the restore**

command to continue.". Press the Restore button in the popup window.

8. In the **Explorer** window, click on the Program.cs file. The file opens in the code editor.
9. Add following code to the beginning of the **Program.cs** file:

```
using OpcLabs.EasyOpc.UA;
```

10. In **Program.cs**, replace the body of the Main method by following code:

```
var client = new EasyUAClient();
object value = client.ReadValue(
    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
    "nsu=http://test.org/UA/Data/;i=10853");
Console.WriteLine(value);
```

11. Select Debug -> Start Debugging (***F5***) from the menu, or press the corresponding button on the toolbar.

    This will build and launch the program. The value will be read from the OPC server and displayed in the **Debug Console** window.

Console Application to Read Value from OPC XML-DA Server

1. Open a command prompt.
2. Create a folder named **HelloXml** and navigate to the folder.
3. Type the following: ***dotnet new console***. This command creates a project file (**HelloXml.csproj**) and a main program file (**Program.cs**).
4. Type the following: ***dotnet add package OpcLabs.QuickOpc***. This command adds OPC Data Client package reference to the project file.

   > The above command actually adds a reference to the package corresponding always to the very latest OPC Data Client version. If you want to be sure that you referencing a package for OPC Data Client 2020.2, use ***dotnet add package OpcLabs.QuickOpc --version 5.58*** command instead.

5. Type the following: ***code .*** (note the dot '.' as a parameter to the command.) Visual Studio Code starts and its window opens.
6. At this point, you might be asked **"Required assets to build and debug are missing from 'vscode'. Add them?"**. If so, press the Yes button in the popup window.
7. A notification will appear **"There are unresolved dependencies from 'Hello.csproj'. Please execute the restore command to continue."**. Press the Restore button in the popup window.
8. In the **Explorer** window, click on the Program.cs file. The file opens in the code editor.
9. Add following code to the beginning of the **Program.cs** file:

```
using OpcLabs.EasyOpc.DataAccess;
```

10. In **Program.cs**, replace the body of the Main method by following code:

```
var client = new EasyDAClient();
object value = client.ReadItemValue(
    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
    "Dynamic/Analog Types/Double");
Console.WriteLine(value);
```

11. Select Debug -> Start Debugging (***F5***) from the menu, or press the corresponding button on the toolbar.

    This will build and launch the program. The value will be read from the OPC server and displayed in the **Debug Console** window.

3.4 Getting Started with OPC UA PubSub under .NET

In this Getting Started, we will use Visual Studio (on Windows) create a console application running that will subscribe to datasets produced by an OPC UA demo publisher, and display the received data on the console. You can target .NET Framework or .NET Core.

Note: The same code will work under Linux as well, but the steps to create the program are different - yo uwill need to use .NET Core CLI Tools, or other IDE such as JetBrains Rider.

In This Topic

1. Start Visual Studio.

2. In this step, we will create a new .NET console application project.

   Select File -> New -> Project from the menu. The dialog(s) that follow differ in different versions of Visual Studio, so we wil just describe the intent here: Select Console App (.NET Core) or Console App (.NET Framework), in C#. If given a choice of framework, make sure you select one of the supported **.NET Runtimes (Section 2.4.1)**. Finalize the dialog, creating the project.

   If you were not given a choice of framework, right-click on the project node in the **Solution Explorer** window, select Properties, and check the **Target Framework** setting.

3. In this step, we will add a reference to the **OpcLabs.QuickOpc** NuGet package.

   Switch to the **Solution Explorer** window, right-click on the project node (ConsoleApp*n*), and select Manage NuGet Packages... command. In the **NuGet: *project*** window, switch to the **Browse** tab, and type ***OpcLabs.QuickOpc*** into the search box. Select the OpcLabs.QuickOpc package in the package list in the left pane of the window. In the right pane of the window, verify or change the package version next to the **Version** label. The version should be "Latest stable 5.58.*build*" (where *build* is a build number). Press the Install button.

   > If you are developing for .NET Framework (not .NET Core) and have installed OPC Data Client using its full **Setup** program, you can instead use assembly references to the OPC Data Client assemblies installed locally. To do so, switch to the **Solution Explorer** window, right-click on the project node (ConsoleApp*n*), and select Add Reference... command. In the **Reference Manager** dialog, select Assemblies -> Extensions, check the boxes next to "OPC Labs Base Library Core" and "OPC Labs EasyOPC-UA Library", and press OK.

4. Open the **Program.cs** file, and add following code to the beginning of the file:

   ```
   using OpcLabs.EasyOpc.UA.PubSub;
   ```

5. In **Program.cs**, replace the body of the Main method by following code (use Copy&Paste from here):

   ```
   var subscriber = new EasyUASubscriber();
   subscriber.SubscribeDataSet("opc.udp://239.0.0.1", (_, eventArgs) =>
   {
   ```

```
        if (eventArgs.Succeeded)
        {
            if (!(eventArgs.DataSetData is null))
            {
                Console.WriteLine();
                Console.WriteLine($"Dataset data: {eventArgs.DataSetData}");
                foreach (var pair in eventArgs.DataSetData.FieldDataDictionary)
                    Console.WriteLine(pair);
            }
        }
        else
        {
            Console.WriteLine();
            Console.WriteLine($"*** Failure: {eventArgs.ErrorMessageBrief}");
        }
    });
    Console.ReadLine();
    subscriber.UnsubscribeAllDataSets();
```

6. Visit Tool Downloads page and download and unpack **OPC UA Demo Publisher** (the easiest to work with is Self-extracting EXE, but other format will do as well). In accordance with instructions on the UADemoPublisher Basics page, run the OPC UA Demo Publisher without further parameters.

   > If you have installed OPC Data Client using its full **Setup** program, you can bypass the tool download and installation described above, and simply run Tools -> OPC UA Demo Publisher from the **Launcher** application, or Windows **Start** menu.

   The OPC UA Demo Publisher will start broadcasting messages that our application will receive. You will see a progress indication (message counts) in its console window. You can run the OPC UA Demo Publisher either on the same computer where you are developing the application, or on a different computer on the same local network (IP subnet).

7. Select Debug -> Start Debugging (***F5***) from the menu, or press the corresponding button on the toolbar.

   This will build and launch the program. The datasets from the publisher will be received by your subscriber application, and displayed on the console. Press ***Enter*** to exit the program.

Troubleshooting

See OPC UA PubSub Common Traps And Pitfalls if things do not work as expected - and specifically, if your subscriber application is not receiving anything and is giving you timeout errors.

Probably the most common issue is the necessity to properly specify a (non-default) network interface. For the **OPC UA Demo Publisher**, this is done using the **--ConnectionNetworkInterface** (**-cni**) option. For a suggestion about how this is done in your application, see e.g. the code comments in **Examples - OPC UA PubSub - Callback using a lambda (Section 13.2.11.1)**.

3.5 Getting Started under COM

This section of the documentation consists of following parts:

- **Getting Started with OPC Classic under COM (Section 3.5.2)**, and
- **Getting Started with OPC UA under COM (Section 3.5.2)**.

Depending on whether you develop OPC "Classic" or OPC Unified Architecture applications under the COM platform, please refer to the corresponding part.

3.5.1 Getting Started with OPC Classic under COM

Start here: **Making a first COM application in VB6 (Section 3.5.1.1)**.

3.5.1.1 Making a first COM application in VB6

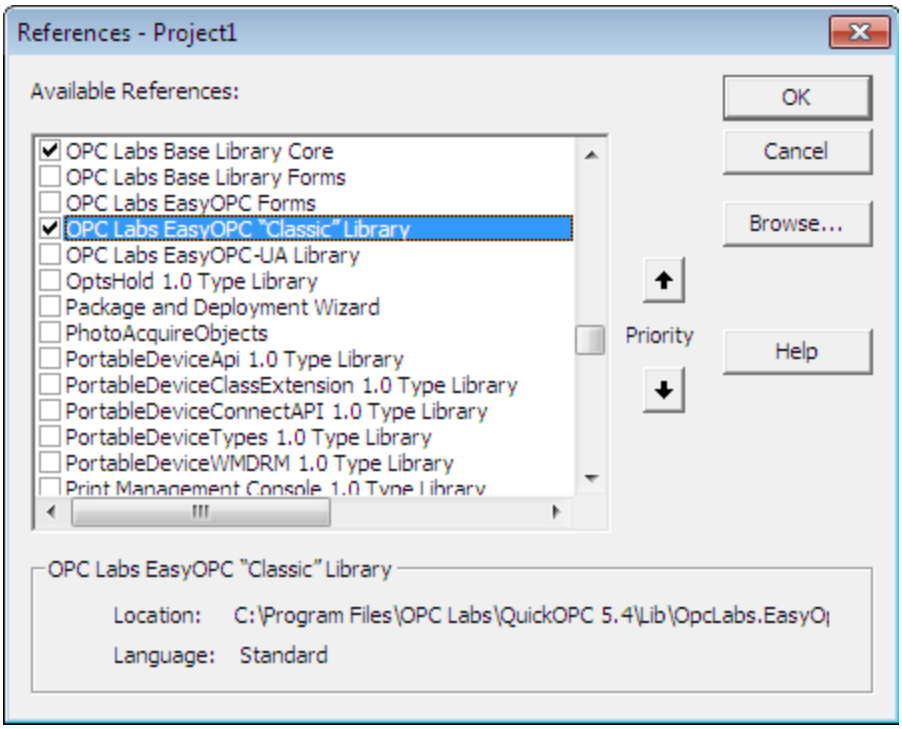
If you are interested in OPC Data Client-COM, your task will most certainly involve reading data from an OPC server. Here are a few steps that illustrate how to achieve that, using Visual Basic 6.0. The steps are similar if you are using other tools or a different programming language.

In This Topic

A Form to Read Value from OPC Data Access Server

We will create a form that will read a float value from the OPC server when loaded, and immediately display the formatted value in the text box on the form.

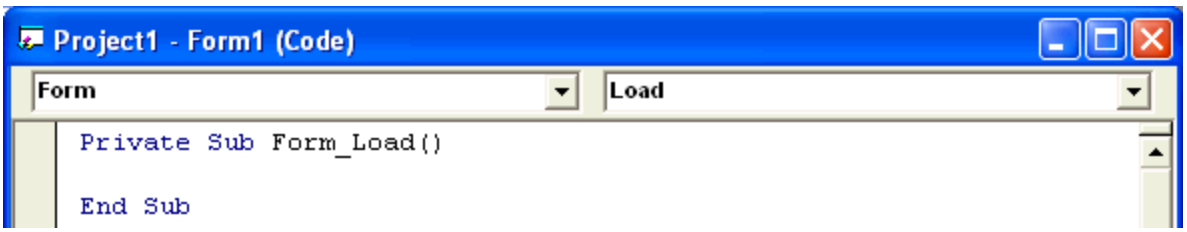
1. Install OPC Data Client. Make sure that you have selected an installation choice that includes COM development.
2. Start Visual Basic 6.0, and create new project, selecting "Standard EXE" project type.
3. Reference the EasyOPC-DA component: Select Project -> References from the menu, and look for "OPC Labs EasyOPC "Classic" Type Library" in the list of available references. Check the box next to it, and press OK.



Note: If some of the OPC Data Client the type libraries are not shown in the References list, use the Browse... button to locate and select them manually. The type libraries are installed under the **SDK\lib** subdirectory under the OPC Data Client installation folder. Sometimes, simply restarting the Visual Basic 6.0 helps.

4. Place a Text Box onto the form's surface. Keep the name Text1.

5. Add handler for the Load event of the form: Switch to Code View, and select the Form object and then its Load member. Visual Basic will add the handler's skeleton as follows:



6. Write the event handler implementation. Add the following code to the body of Form_Load method:

```
' Create EasyOPC-DA component
Dim EasyDAClient As New EasyDAClient

' Read item value and display it
Me.Text1 = EasyDAClient.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Single")
```

7. Build and run the application by selecting Run -> Start from the menu, and observe the results. The text box on the form will be filled with a float value we have read from the OPC server.

Notes

If you are using a tool that does not have a referencing mechanism similar to Visual basic, you may need to create the EasyOPC-DA component directly. To do so, use the tool's mechanism to create instance of a COM object, passing it a ProgID string "OpcLabs.EasyOpc.DataAccess.EasyDAClient".

3.5.1.2 Where do I go from here?

You now have a basic idea of how to use the OPC Data Client-COM. There are many additional resources that you can use to gain additional knowledge about the product. Here are some of them:

- Play with the Demo application (accessible from the Start menu or the Launcher application) and try obtaining data from the included Simulation OPC server, but also from other OPC servers on your computer or your network.
- Read the User's Guide.
- Study the Reference documentation.
- Review the additional examples and tools included with the product.

3.5.2 Getting Started with OPC UA under COM

Start here: **Making a first UA application in VB6 (Section 3.5.2.1)**.

3.5.2.1 Making a first UA application in VB6

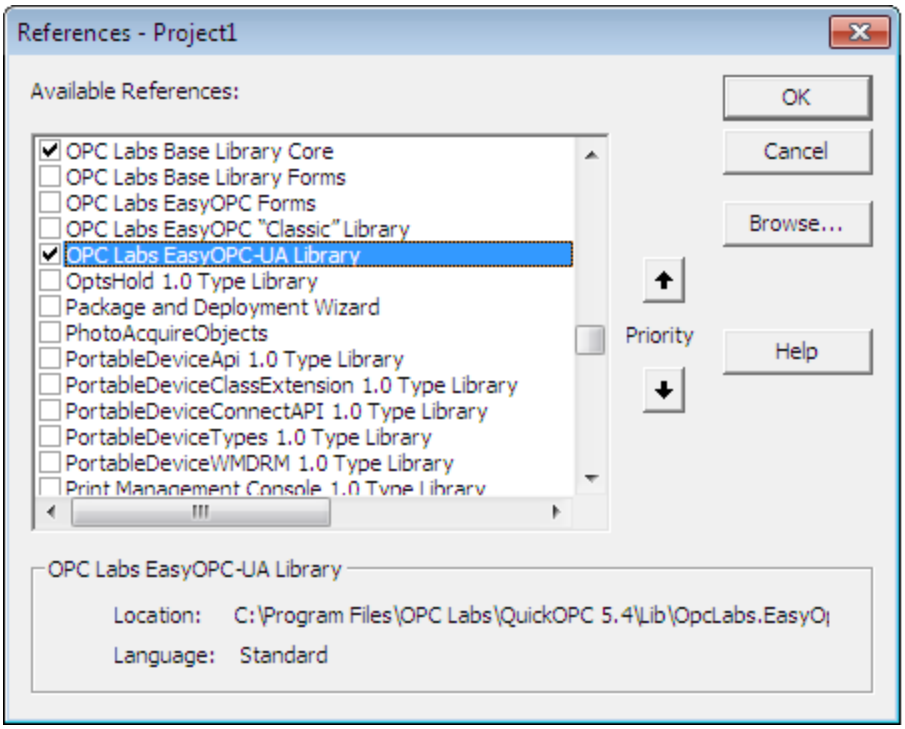
If you are interested in OPC Data Client-UA, your task will most certainly involve reading data from an OPC UA server. Here are a few steps that illustrate how to achieve that, using Visual Basic 6.0. The steps are similar if you are using other tools or a different programming language.

In This Topic

A Form to Read Value from OPC Unified Architecture Server

We will create a form that will read a float value from the OPC UA server when loaded, and immediately display the formatted value in the text box on the form.

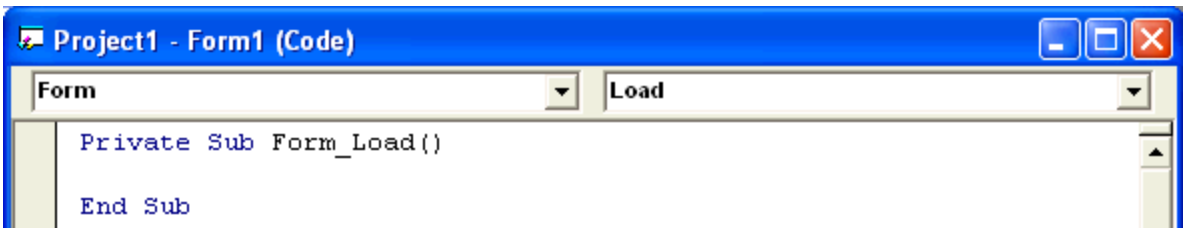
1. Install OPC Data Client. Make sure that you have selected an installation choice that includes COM development.
2. Start Visual Basic 6.0, and create new project, selecting "Standard EXE" project type.
3. Reference the EasyOPC component: Select Project -> References from the menu, and look for "OPC Labs EasyOPC-UA Type Library" in the list of available references. Check the box next to it, and press OK.



Note: If some of the OPC Data Client the type libraries are not shown in the References list, use the Browse... button to locate and select them manually. The type libraries are installed under the **SDK\lib** subdirectory under the OPC Data Client installation folder. Sometimes, simply restarting the Visual Basic 6.0 helps.

4. Place a Text Box onto the form's surface. Keep the name Text1.

5. Add handler for the Load event of the form: Switch to Code View, and select the Form object and then its Load member. Visual Basic will add the handler's skeleton as follows:



6. Write the event handler implementation. Add the following code to the body of Form_Load method:

```
' Create EasyOPC-UA component
Dim Client As New EasyUAClient

' Read node value and display it
Me.Text1 = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
```

7. Build and run the application by selecting Run -> Start from the menu, and observe the results. The text box on the form will be filled with a float value we have read from the OPC UA server.

Notes

If you are using a tool that does not have a referencing mechanism similar to Visual basic, you may need to create the EasyOPC-UA component directly. To do so, use the tool's mechanism to create instance of a COM object, passing it a ProgID string "OpcLabs.EasyOpc.UA.EasyUAClient".

3.5.2.2 Where do I go from here?

You now have a basic idea of how to use the OPC Data Client-COM. There are many additional resources that you can use to gain additional knowledge about the product. Here are some of them:

- Play with the Demo application (accessible from the Start menu) and explore the various functionalities that the product provides.
- Read the User's Guide.
- Study the Reference documentation.
- Review the additional examples and tools included with the product.

4 Fundamentals

This chapter describes the fundamental concepts used within OPC Data Client component. Please read it through, as the knowledge of Fundamentals is assumed in later parts of this document.

4.1 Product Parts

4.1.1 Assemblies

At the core of OPC Data Client.NET, there are .NET assemblies that contain reusable library code. You reference these assemblies from the code of your application, and by instantiating objects from those assemblies and calling methods on them, you gain the OPC functionality.

For easy recognition among other assemblies when used in a larger context, all our assemblies start with "OpcLabs" prefix.

Assembly List

Following assemblies form the main part of OPC Data Client.NET:

Assembly Name or File	Title	Description	Strong-named	Present locally - for .NET Framework	Present in NuGet package - .NET Framework	Present in NuGet package - .NET Standard
App_Web_OpcLabs.EasyOpcClassicRaw.amd64	EasyOPC "Classic" Raw Library	Used internally. Low-level OPC "Classic" code, a mixed mode assembly, for 64-bit processes (x64 architecture).	yes	✓	✓	×
App_Web_OpcLabs.EasyOpcClassicRaw.x86	EasyOPC "Classic" Raw Library	Used internally. Low-level OPC "Classic" code, a mixed	yes	✓	✓	×

		mode assembly, for 32-bit processes (x86 architecture).				
OpcLabs.BaseLib	OPC Labs Base Library Core	Supporting code.	yes	✓	✓	✓
OpcLabs.BaseLibForms	OPC Labs Base Library Forms	Contains fundamental and common classes for Windows Forms.	yes	✓	✓	×
OpcLabs.BaseLibPresentation	OPC Labs Base Library Presentation	Contains fundamental and common classes for Windows Presentation Foundation (WPF).	yes	×	✓	×
OpcLabs.EasyOpcClassic	EasyOPC "Classic" Library	Contains classes that facilitate easy work with various OPC specifications, such as OPC Data Access, OPC XML-DA, and OPC Alarms and Events.	yes	✓	✓	✓
OpcLabs.EasyOpcForms	EasyOPC Forms	Contains classes that facilitate easy work with OPC (both "Classic" and Unified Architecture) from Windows Forms applications.	yes	✓	✓	×

OpcLabs.EasyOpcUA	EasyOPC-UA Library	Contains classes that facilitate easy work with OPC Unified Architecture.	yes	✓	✓	✓

The name of the physical file that contains the assembly is always same as the name of the assembly, with a file extension ".dll".

When targeting .NET Framework, the OPC Unified Architecture components are based on version 1.03 of the .NET Stack from OPC Foundation. When targeting .NET Standard, the OPC Unified Architecture components are based on version 1.04 of the .NET Stack from OPC Foundation.

For the curious, OPC Data Client has been developed in Microsoft Visual Studio 2019. The layers that directly use COM (such as the OpcLabs.EasyOpcClassic assembly) are written in managed C++ (C++/CLI). More precisely, they are mixed mode assemblies, where the bulk of the code is in MSIL instructions, with a few exceptions where necessary. All other parts are written in pure C#.

Additional packages

OPC Data Client also includes packages (sets of assemblies) that are only used in specific circumstances, as documented. They are available as NuGet packages (from www.nuget.org , for use in .NET Standard projects), or as ZIP packages (links to corresponding ZIP file downloads are installed with the product and reachable through Redistributables item in the Start menu or the Launcher program).

The additional packages include e.g.:

- MQTT communication packages
- OPC UA PubSub JSON mapping package
- OpcLabs.Pcap communication package

The additional packages may have their own licensing requirements, different from the core OPC Data Client.

Assemblies for .NET Framework

The assembly files are installed into a subdirectory called **Assemblies\net47** under the installation directory of the product.

OPC Data Client components were consciously written to target Microsoft .NET Framework 4.7, i.e. they do not depend on features available only in the later version of the framework. As such, you can use the components in applications targeting version 4.7, 4.7.1, 4.7.2 or 4.8 of the Microsoft .NET Framework.

There is also a **Design** subfolder under the **Assemblies** folder, which contains additional OPC Data Client assemblies that are used during design time only. These assemblies are never referenced or used in run-time by your projects.

For development purposes, the assemblies under the **Design** subfolder are installed into the GAC (Global Assembly Cache) by the setup program. Other (runtime) assemblies are not installed into the GAC by default (the installation program can do it if you select the custom installation, it will then offer you this option near the end of the installation). You do not have to install assemblies into the GAC on the runtime computers, though. We recommend against installing the runtime assemblies into the GAC, unless you have a compelling reason for it.

The somewhat strange naming of certain assemblies, with names starting with "**App_Web_**" prefix, is needed for proper mixed-mode assembly loading under ASP.NET. Do not be confused by the presence of the word "Web" in the name – the

same assemblies are needed outside of Web applications as well.

The .NET assemblies are also necessary when you are using COM components for OPC. They are implemented in .NET, and exposed to the COM worlds using a COM interop.

When specifically selected during an installation (with the "Embedded assemblies" install component, or as part of the Full install), additional assemblies (that are normally embedded within OPC Data Client assemblies and thus invisible to the developer) are placed alongside the usual OPC Data Client assemblies. These assemblies are needed in certain rare situations, e.g. when registering type libraries for the dependent assemblies for COM interop, or under certain hosting environments or in some development tools.

Assemblies for .NET Framework are also used when you target the Excel as a development platform, with **Excel Option (Section 12.1)**.

Assemblies for .NET Standard

In .NET Standard, the OPC Data Client are not installed using a **Setup Program (Section 3.1.1)**. Instead, they are provided in form of **NuGet Packages (Section 3.1.2)**, available from www.nuget.org. The assemblies do not have a fixed or "well-known" installation location on your computer, and are installed separately into each project that uses them.

In addition, the OPC Data Client NuGet packages have dependencies on other NuGet packages, and so on. The precise structure and content of these dependencies depends on the targeted runtime. Consequently, by installing the OPC Data Client NuGet package(s) into your project, it ends up referencing more (possibly many) assemblies than explicitly listed here.

Strong-naming

As a general rule, OPC Data Client assemblies are strong-named.

An exception to this rule is made when we need to reference other assemblies that are not strong-named. In such case, we move such parts into a separate OPC Data Client assembly that is not strong-named. Usage of these parts is always optional and depends on the functionality you need. An example of this approach is the OpcLabs.Pcap assembly, which references the SharpPcap and PacketDotNet assemblies. You only need it when your application uses OPC UA PubSub Ethernet mapping, or when it works with Wireshark capture files.

The OPC Data Client assemblies that are strong-signed cannot directly reference the assemblies that are not strong-signed. For this reason, the assemblies that are strong-signed load the assemblies that are not strong-signed dynamically. Consequently, different (and platform-dependent) rules apply to where such assemblies are being searched for.

Also, assemblies that are not strong-signed cannot be put to GAC (Global Assembly Cache).

4.1.1.1 XML Comments

Together with the .DLL files of the assemblies, there are also .XML files that contain XML comments for them. The texts contained in these files are used by various tools to provide features such as IntelliSense and Object Browser information in Visual Studio.

4.1.1.2 ReSharper Annotations

API members are annotated using JetBrains ReSharper (http://www.jetbrains.com/resharper/) attributes, such as the most common **[CanBeNull]** and **[NotNull]** attributes. ReSharper users benefit from the annotations, as possible improper

usages are recognized and highlighted by ReSharper's code inspection feature.

4.1.2 COM Components

At the core of OPC Data Client-COM there are COM components that contain reusable library code. You reference these components from the code of your application, and by instantiating objects from those components and calling methods on them, you gain the OPC functionality.

In This Topic

COM components for OPC "Classic" and OPC UA are implemented in .NET, and exposed to the COM worlds using a COM interop. As such, they are contained in the .NET assemblies described under a corresponding chapter earlier, and installed in a subdirectory called **Assemblies\net47** under the installation directory of the product.

The assemblies become usable from a COM world by registering them using the Regasm.exe (Assembly Registration Tool), which is a part of .NET Framework. The OPC Data Client installation program registers the assemblies for COM interop automatically.

The Assembly Registration tool reads the metadata within an assembly and adds the necessary entries to the registry, which allows COM clients to create .NET Framework classes transparently. Once a class is registered, any COM client can use it as though the class were a COM class. The class is registered only once, when the assembly is installed. Instances of classes within the assembly cannot be created from COM until they are actually registered.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.1.2.1 Type Libraries (COM)

In Microsoft COM, the components are described by their corresponding Type Libraries. Type libraries are binary files (.tlb, .dll or .exe files) that include information about types and objects exposed by an ActiveX (COM) application. A type library can contain any of the following:

In This Topic

- Information about data types, such as aliases, enumerations, structures, or unions.
- Descriptions of one or more objects, such as a module, interface, IDispatch interface (dispinterface), or component object class (coclass). Each of these descriptions is commonly referred to as a typeinfo.
- References to type descriptions from other type libraries.

By including the type library with OPC Data Client, the information about the objects in the library is made available to the users of the applications and programming tools.

All type libraries (those that that are not embedded within other specific .dll or .exe files) in are located under the **SDK**\lib subdirectory in the product installation folder. Type libraries for 64-bit development are located under **SDK\lib\x64**. Note that the interface exposed by OPC Data Client (including the size of integer types) is the same, regardless whether it is for

32-bit or 64-bit development; there are just slight "formal" differences caused by the way the type libraries for 32-bit vs. 64-bit development are generated. Most development can simply use the 32-bit type libraries under the **SDK\lib** subdirectory.

Note that it is not strictly necessary to use type libraries for COM development. With late binding, it is possible to make use of the COM objects of OPC Data Client "as they are". Type libraries are used mainly with early binding.

Whether or not to use the type libraries depends in part on the tool or language you are using. Some allow only late binding (e.g. VBScript), some may allow only early binding, and yet some allow both (e.g. VB6). When your tool allows both options, usually the early binding is better (for performance and type checking reasons), but there are also situations where late binding is still appropriate or necessary. Some of these situations have to do with particular "quirks" present in the tools.

For example, in order to assign atomic values (such as numbers or strings) to a State property of OPC Data Client objects, you will need to use late binding in VB6. This is because the property as declared as VARIANT, but VB6 treats it as Object, and does not allow the non-object values be stored into it with early binding.

As described earlier, COM components of OPC Data Client are actually created from .NET components, using COM interop tools from .NET Framework. For this reason, there is one COM type library for each assembly that contains the OPC Data Client components.

This means that following type libraries are available to you:

Type Library File	Title	Description
OpcLabs.BaseLib.tlb	OPC Labs Base Library Core	Supporting code
OpcLabs.BaseLibForms.tlb	OPC Labs Base Library Forms	Contains fundamental and common classes for Windows Forms.
OpcLabs.EasyOpcClassic.tlb	OPC Labs EasyOPC "Classic" Library	Contains classes that facilitate easy work with OPC Classic.
OpcLabs.EasyOpcForms.tlb	OPC Labs EasyOPC Forms	Contains classes that provide OPC-related (both "Classic and Unified Architecture) user interface.
OpcLabs.EasyOpcUA.tlb	EasyOPC-UA Library	Contains classes that facilitate easy work with OPC Unified Architecture.

This documentation is for OPC Data Client version 2020.2. This is known as the *external version* of the product. The *internal version* number is 5.58, and that is also the version of the type libraries.

Each of these libraries is marked with an internal version number of the product, e.g. 5.58. Many COM tools (including e.g. the Visual Basic's References dialog, or OleView), however, display the version number of a type library in a hexadecimal notation, but without giving any indication of it. This may lead to confusion, because e.g. for OPC Data Client internal version 5.58, the version gets displayed as 5.3a (a hexadecimal representation of the internal version). Be careful to interpret the version numbers correctly.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.1.3 Demo Applications

OPC Data Client ships with several applications that demonstrate its usage and capabilities. Depending on the development development platform you target, there are:

- **.NET Framework Demo Applications (Section 4.1.3.1)**
- **COM Demo Applications (Section 4.1.3.2)**
- **Excel Demo Applications (Section 12.1.4)**

There are currently no demo applications for .NET Standard. Their purpose is served by the **Examples (Section 4.1.4)**.

4.1.3.1 .NET Framework Demo Applications

The demo applications for .NET Framework are categorized by the OPC technology they use:

(Section 4.1.3.1.1)

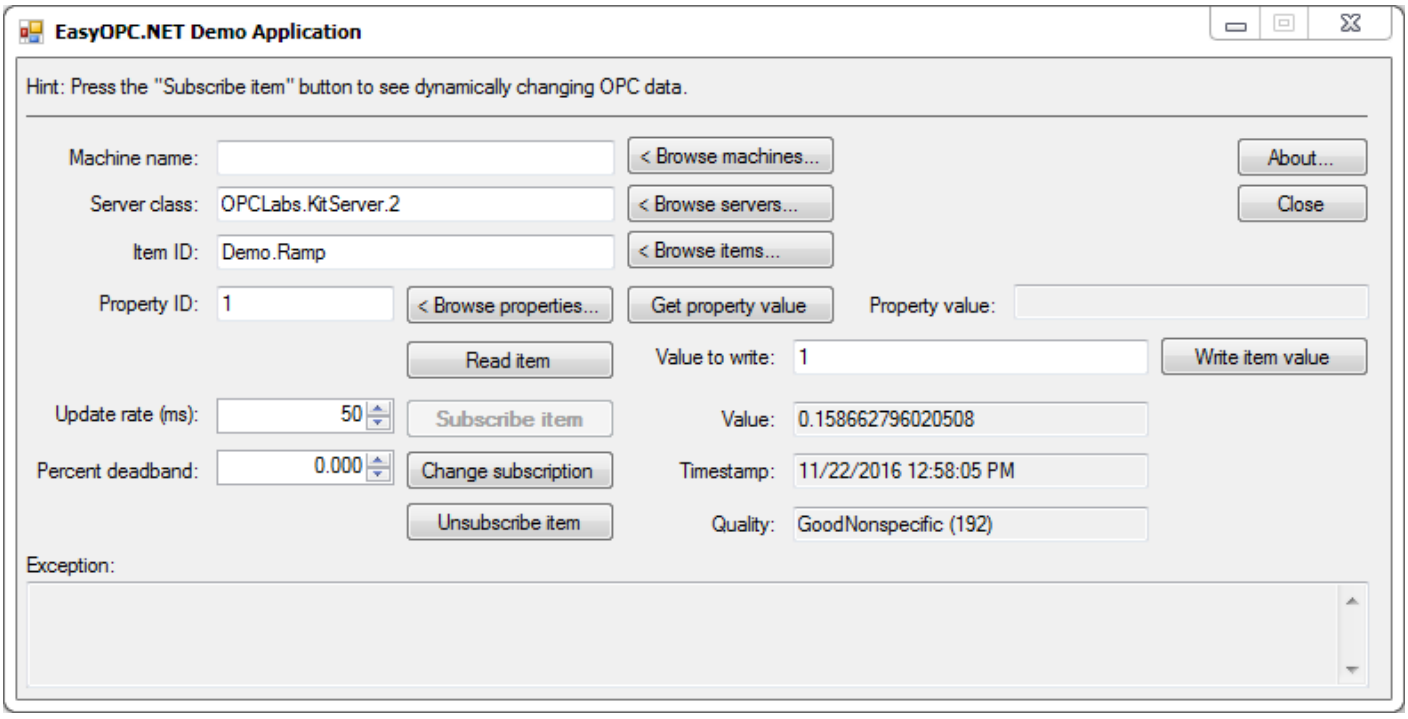
- **.NET Framework Demo Applications for OPC Classic (Section 4.1.3.1.1)**
- **.NET Framework Demo Applications for OPC UA (Section 4.1.3.1.2)**

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

4.1.3.1.1 .NET Framework Demo Applications for OPC Classic

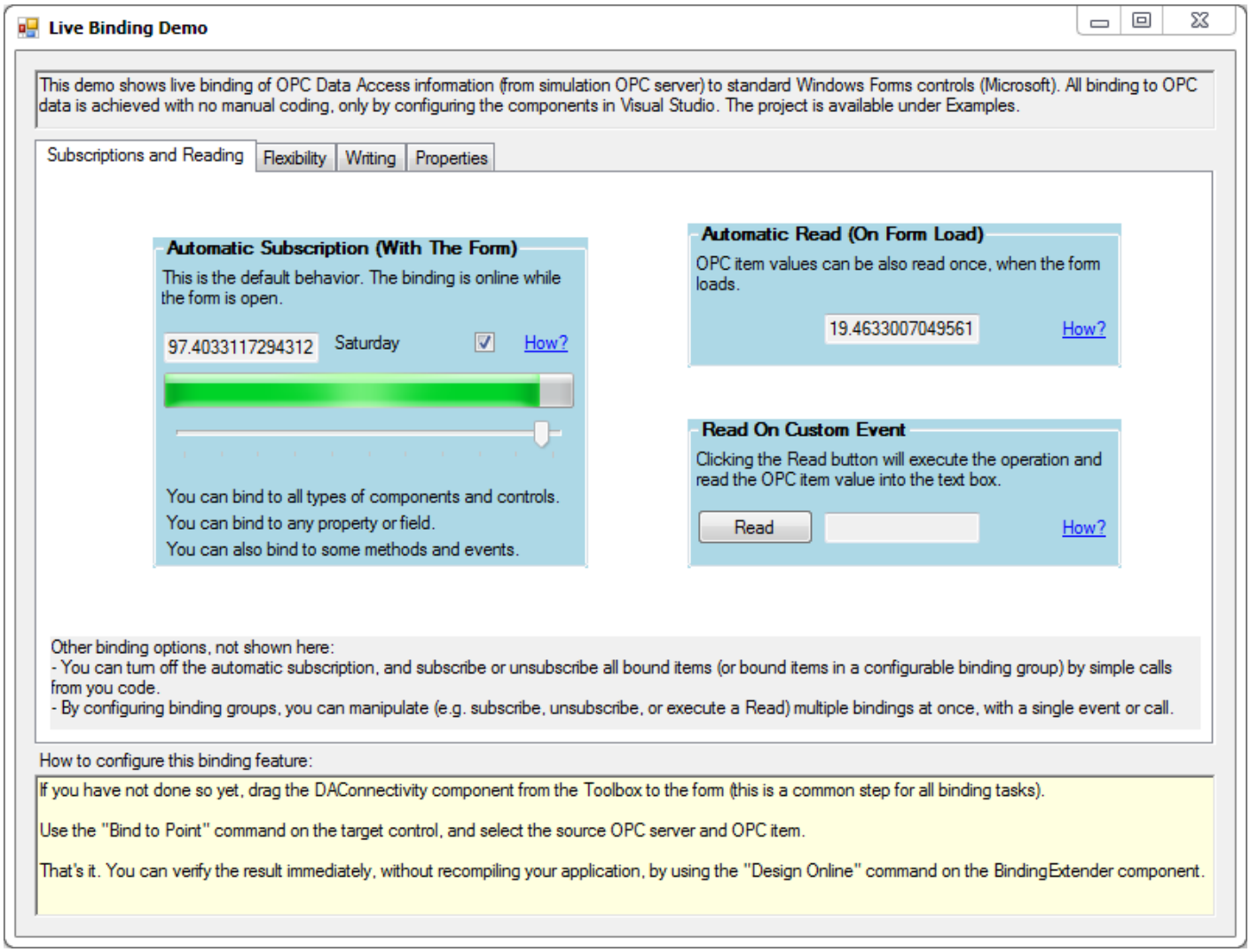
The OPC Data Client.NET demo applications are available from the Start menu or the Launcher application.

The basic demo is written using imperative programming model (explained further down in this document).



Note: There is also a similar application with an extra support for OPC XML-DA.

The Live Binding Demo shows live binding of OPC Data Access information (from simulation OPC server) to standard Windows Forms controls (Microsoft). All binding to OPC data is achieved with no manual coding, only by configuring the components in Visual Studio. This demo also serves as a tutorial application for the live binding development model: When you click on any of the “How?” links, the bottom pane displays instructions explaining how to configure that particular feature.



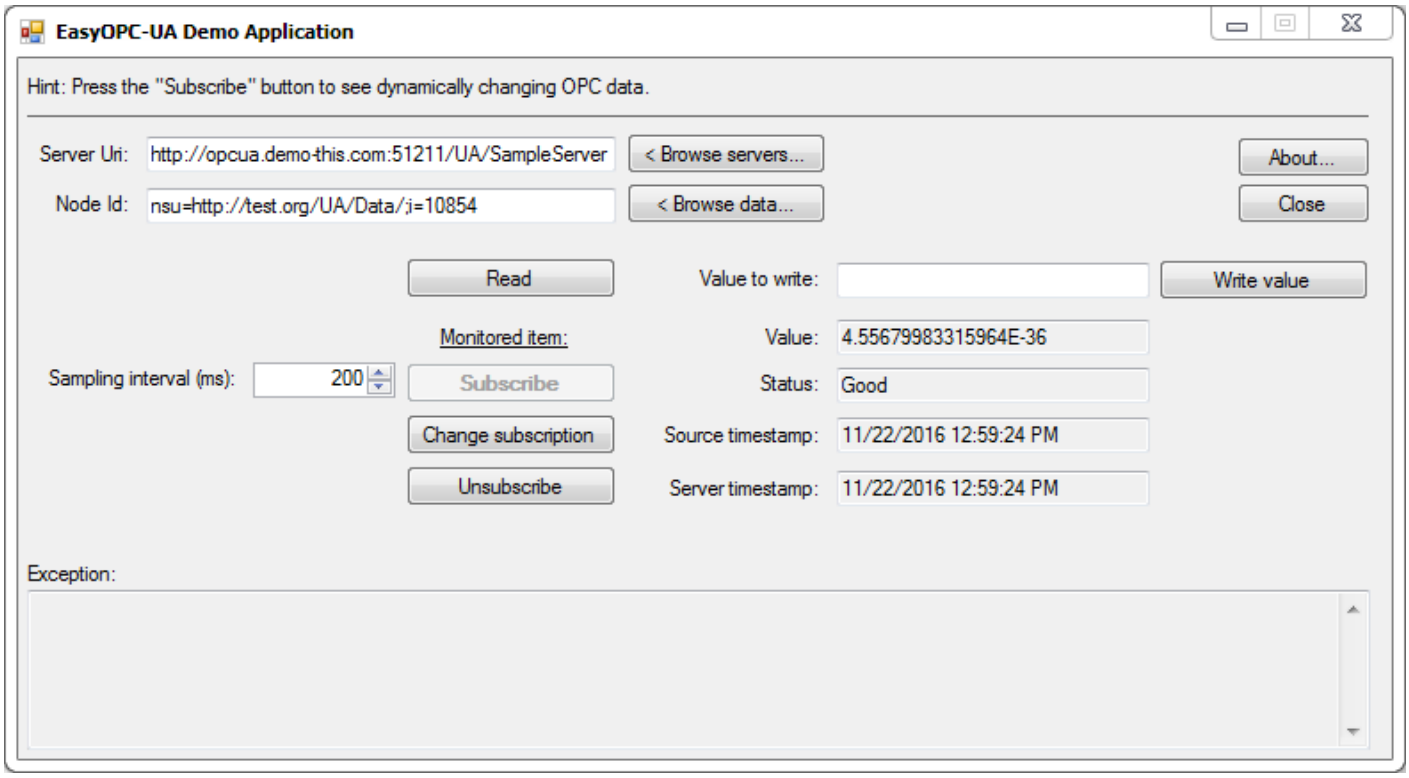
There are also several “integration demos”, showing how OPC Data Client features can be used in combination with other products that are not part of OPC Data Client. Following integration demos are currently available:

- Demo with Industrial Gadgets .NET
- Demo with Instrumentation Controls
- Demo with Symbol Factory .NET

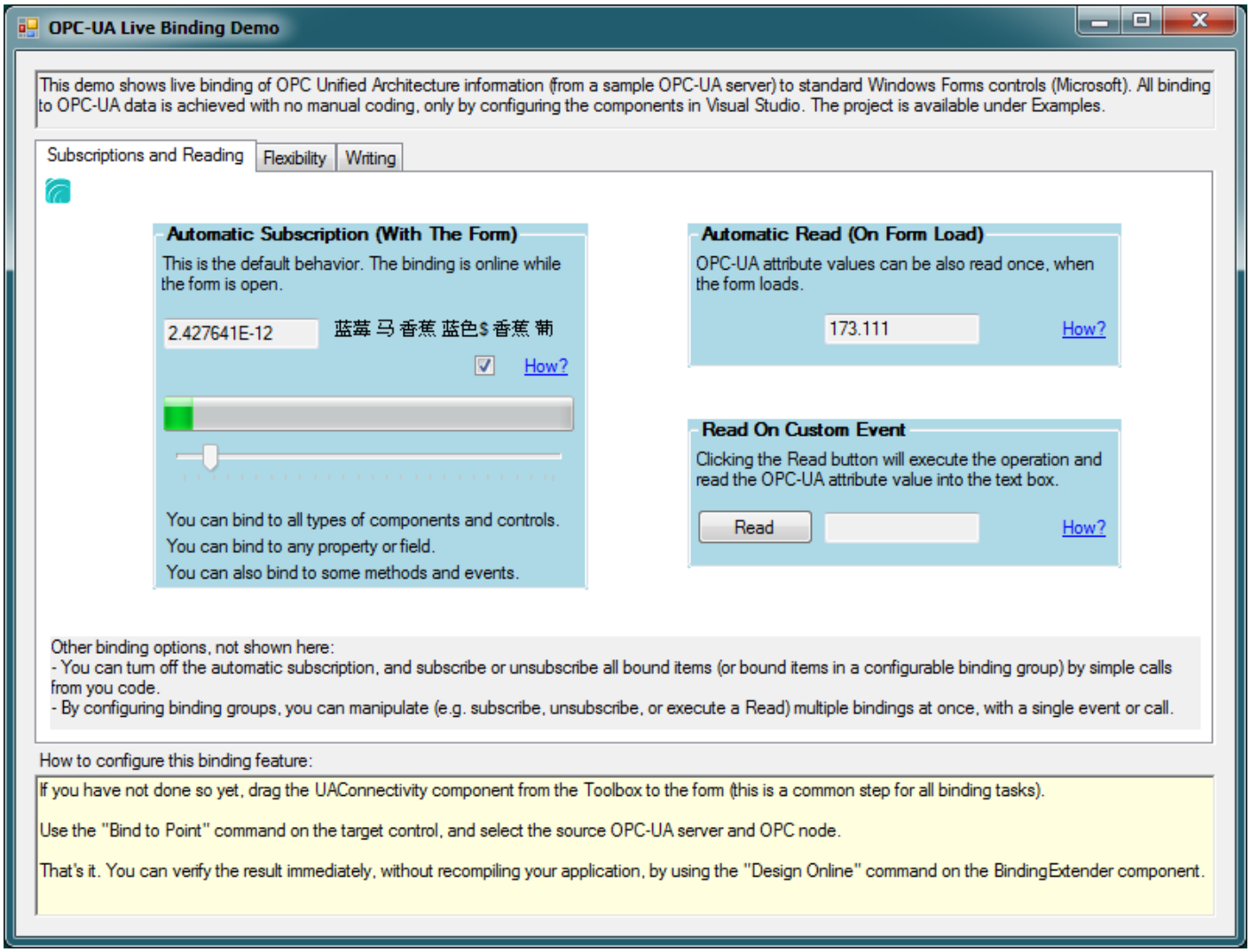
There is also a similar application that demonstrates the Live Binding capabilities with OPC Unified Architecture.

4.1.3.1.2 .NET Framework Demo Applications for OPC UA

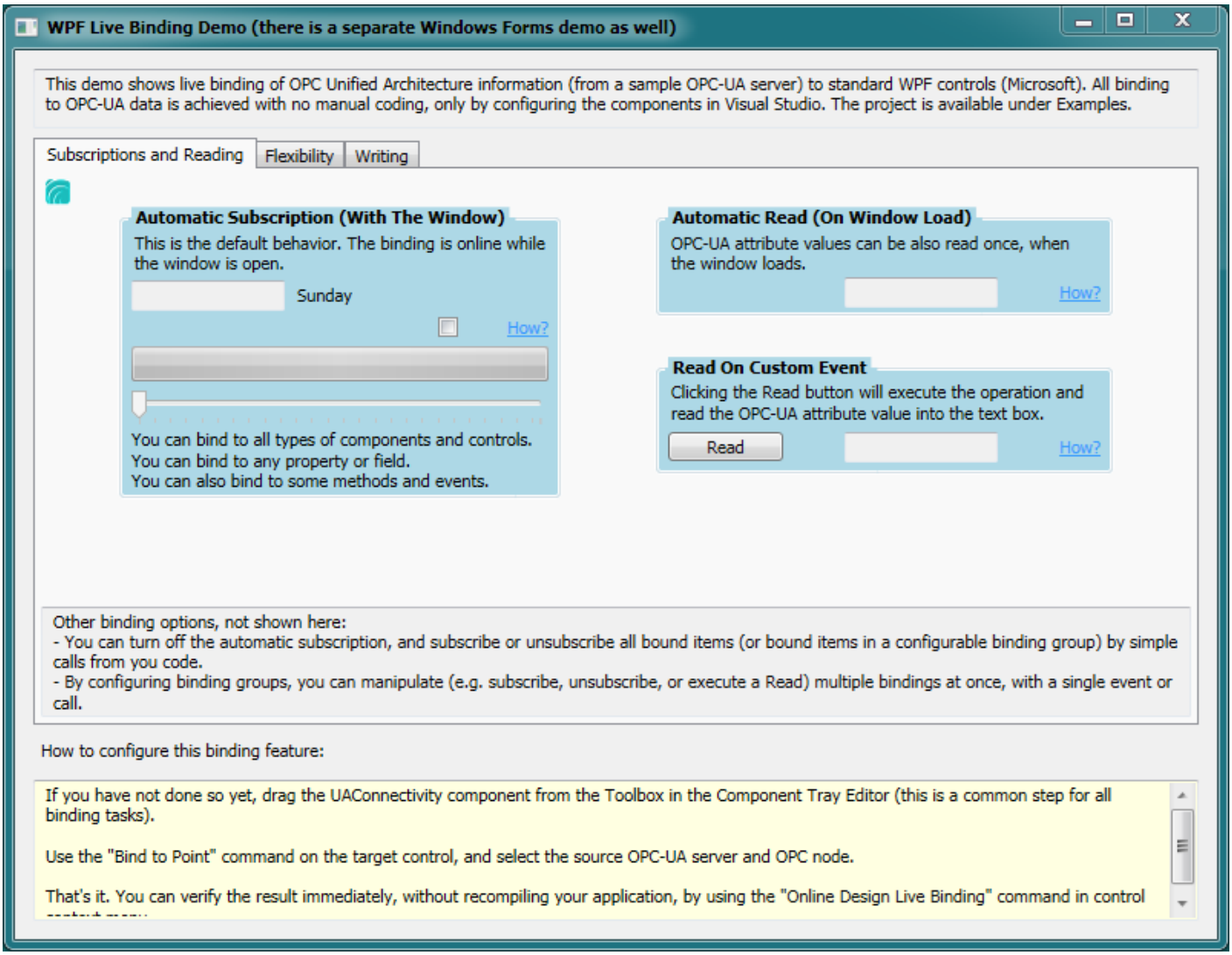
OPC Data Client-UA installs with a demo application that allows exploring various functions of the product. The demo application is available from the Start menu or the Launcher application.



The Live Binding Demo shows live binding of OPC Unified Architecture information (from simulation OPC server) to standard Windows Forms controls (Microsoft). All binding to OPC data is achieved with no manual coding, only by configuring the components in Visual Studio. This demo also serves as a tutorial application for the live binding development model: When you click on any of the “How?” links, the bottom pane displays instructions explaining how to configure that particular feature.

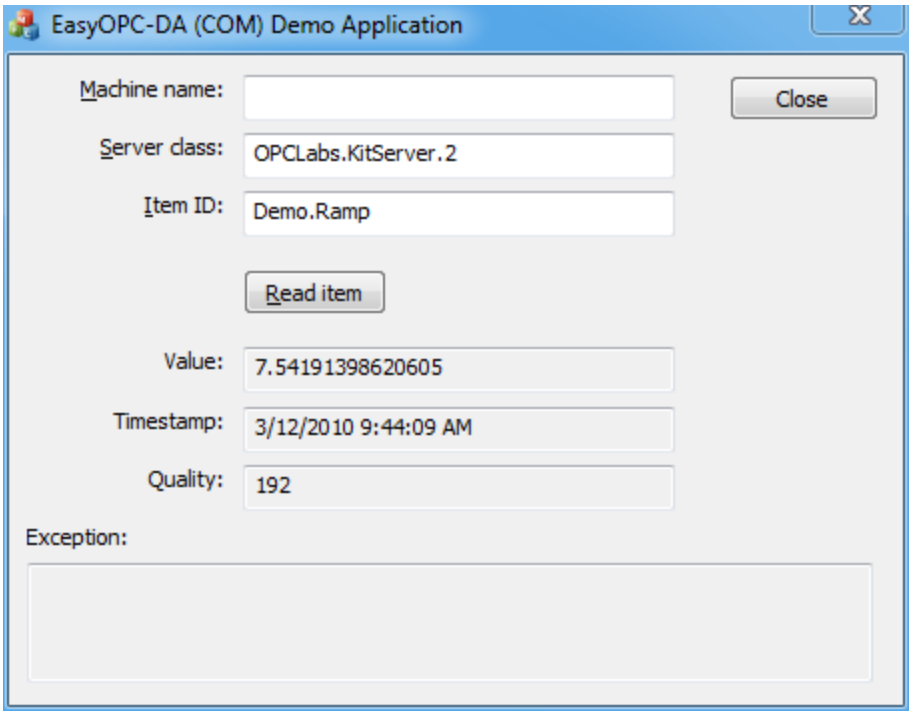


There is also a Live Binding demo application written for Windows Presentation Foundation (WPF), with functionality closely matching that of Windows Forms application:



4.1.3.2 COM Demo Applications

OPC Data Client-COM installs with a demo application that allows exploring basic functions of the product. The demo application is available from the Start menu or the Launcher application.



4.1.4 Examples

Example code, projects and solutions are available from the Start menu or the Launcher application (if you have used the **Setup Program (Section 3.1.1)** on Windows to install OPC Data Client). You can also access them from the file system:

- **COM Examples (Section 13.1.1)** (OPC Data Client-COM and OPC Data Client-UA for COM) are in the **Examples-COM** subfolder of the installation folder.
- **.NET Framework Examples (Section 13.1.2)** (OPC Data Client.NET and OPC Data Client-UA for .NET) are in the **Examples-NET** subfolder of the installation folder.
- **.NET Standard Examples (Section 13.1.3)** for .NET Core runtime are in the **Examples-NETCore** subfolder of the installation folder.

If you are developing for .NET Standard and have not used the **Setup Program (Section 3.1.1)**, you can download the **.NET Standard Examples (Section 13.1.3)** from the Web, in form of archives.

There is a separate **Examples (Section 13)** section in this document that contains list of all examples and their purpose, with chapters for **.NET Framework Examples (Section 13.1.2)**, **COM Examples (Section 13.1.1)** and **.NET Standard Examples (Section 13.1.3)**. You can access these documentation chapters from the Start menu as well.

4.1.4.1 LINQPad support

LINQPad (http://www.linqpad.net/) is a software utility for .NET that allows you to quickly and interactively test out C#, VB.NET and F# code without the need for an IDE. We highly recommend that you use this tool (free edition exists) for experiments and learning.

OPC Data Client works well with LINQPad. We have made some special tweaks for it as well, e.g. to ensure a proper expandable display of all types in LINQPad.

OPC Data Client standalone installer ships with examples for LINQPad; see the Examples section in this document for details. The NuGet packages include the LINQPad examples as well.

4.1.5 Documentation and Help

The documentation consists of following parts:

- **User's Guide and Reference (online)**. The reference is formatted according to Visual Studio style and standards, and integrates with Visual Studio help system. This document include, among other content:
  - Getting Started. Short step-by-step instructions to create your first project.
  - Examples in .NET Framework, .NET Standard and COM.
  - Documentation for StreamInsight Option and Excel Option.
  - Reference.
- **User's Guide (PDF, online)**. The content is same as User's Guide and Reference above, but the Reference part is not included.
- **What's New** (online). Contains information about changes made in this and earlier versions.

Because the COM components are implemented by exposing the .NET objects of OPC Data Client to COM, you may need to refer to OPC Data Client reference documentation in order to find information about COM objects and interfaces of OPC Data Client as well.

You can access all the above mentioned documentation from the **Start** menu or the **Launcher** application.

In addition, there is IntelliSense and Object Browser information available from the Visual Studio environment.

The OPC Data Client help contents integrate Microsoft Visual Studio 2017/2019 Help (Microsoft Help Viewer 2 format). This also means that the reference documentation works as a contextual help; e.g. pressing F1 when the cursor is on some of the OPC Data Client types or members invokes directly the help page for that type or member.

> ⚠ See TechNet article Visual Studio 2017: Getting Started with Help Viewer for information about how to enable offline documentation in Visual Studio.
>
> In addition, for the contextual help to work, you need to have the help preference set properly. In Visual Studio, do this:
>
> Help -> Set Help Preference -> Launch in Help Viewer
>
> (i.e. **not** Launch in Browser).

4.1.6 Bonus Material

Bonus Material

From OPC Labs Knowledge Base
Jump to navigation Jump to search

Introduction

The QuickOPC installation also contains bonus tools and materials related to the product. The bonus material may be updated and enhanced more frequently than the base product, but it is not an official part of the product, and support for it is not guaranteed.

Note that some tools are only available with or related to QuickOPC-COM, QuickOPC.NET or QuickOPC-UA. In such case, they are labeled accordingly, or marked with corresponding COM or .NET icon further down in the text.

OPC-DA Quality Decoder

Enter the (rather cryptic) OPC-DA quality value that some tools display, press “Decode”, and this application will decode the value and show you the various bit fields contained in the OPC-DA quality.

This is a WPF (Windows Presentation Foundation) application. Its source code is included in the examples solution.

UA Configuration Tool

This is an OPC Foundation tool, providing a centralized place to configure the applications based on the .NET stack, manage application certificates, manage certificate repositories, and COM interoperability. For documentation, see

- Launching the UA Configuration Tool, and
- UA Configuration Tool Overview.

The UA Configuration Tool is under OPC Foundation MIT License 1.00, whose text is also included (in **license.txt** file) in the installatrion folder of the tool.

Navigation menu

Personal tools

- Log in

Namespaces

- Page
- Discussion

☐

Variants

Views

- Read
- View source
- View history

☐

4.1.7 Tools and Online Services

Besides offering components to integrate within your project, OPC Data Client also offers several tools and online service that can assist you in development. They include:

- **License Manager (Section 11.1)** - GUI, and **LMConsole Utility (License Manager Console) (Section 11.1.1)**
- **Connectivity Explorer (Section 11.2)**
- **The Launcher application (Section 11.3)**
- **OpcCmd Utility (Section 11.4)**
- **Demo Servers and Publishers (Section 11.5)** (some installable locally, some are online)

4.2 Typical Usage

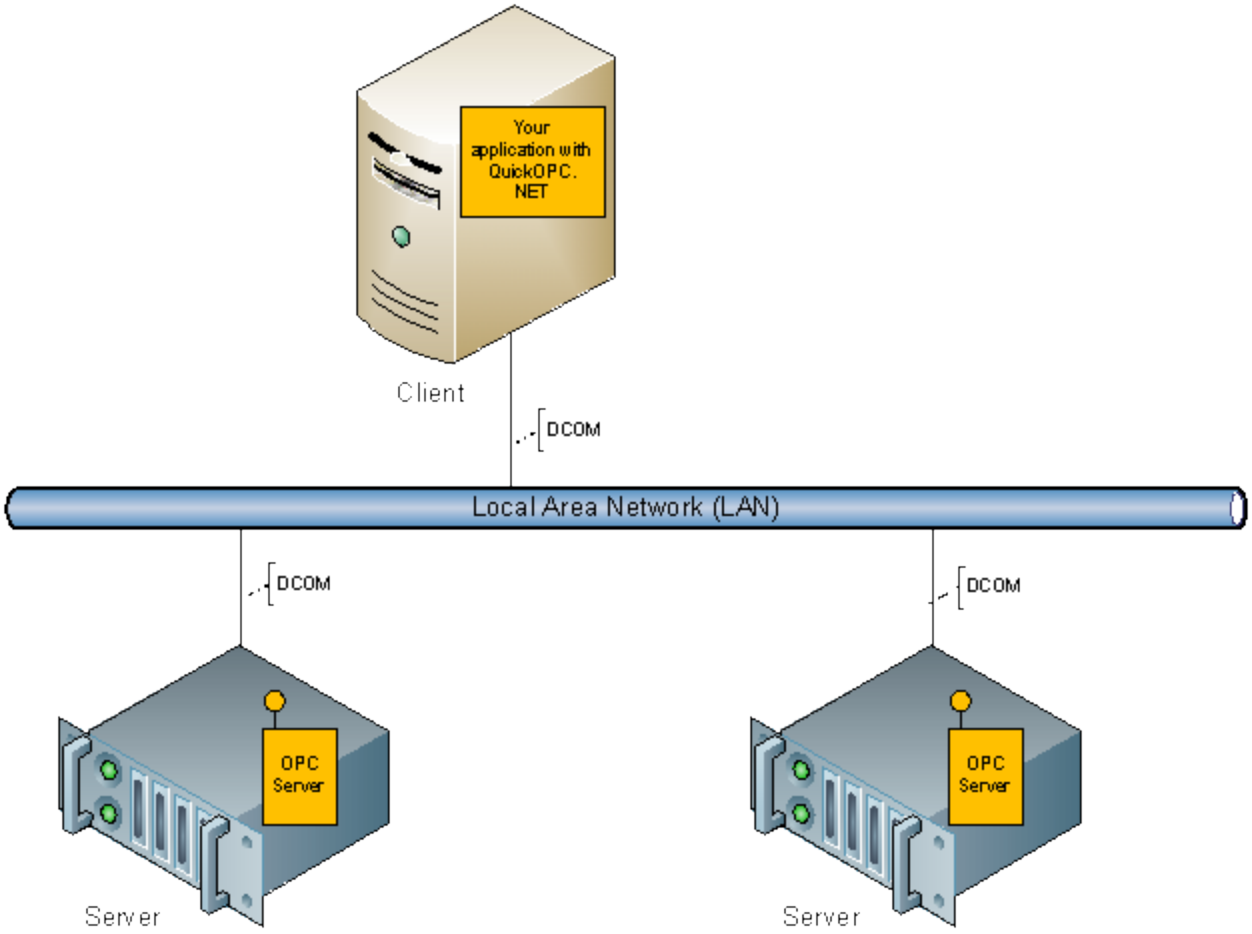
OPC Data Client.NET and OPC Data Client-UA for .NET are suitable for use from within any tool or language based on Microsoft .NET. There are many different scenarios for it, but some are more common.

OPC Data Client-COM and OPC Data Client-UA for COM are suitable for use from within any tool or language based on Microsoft (COM) Automation. There are many different scenarios for it, but some are more common.

4.2.1 OPC Classic thick-client .NET applications on LAN

The most typical use of OPC Data Client.NET involves a thick-client user application written in one of the Microsoft .NET languages. This application uses the types from OPC Data Client.NET object model, and accesses data within OPC servers that are located on remote computers on the same LAN (Local Area Network). The communication with the target OPC server is performed by Microsoft COM/DCOM technology.

The following picture shows how the individual pieces work together:

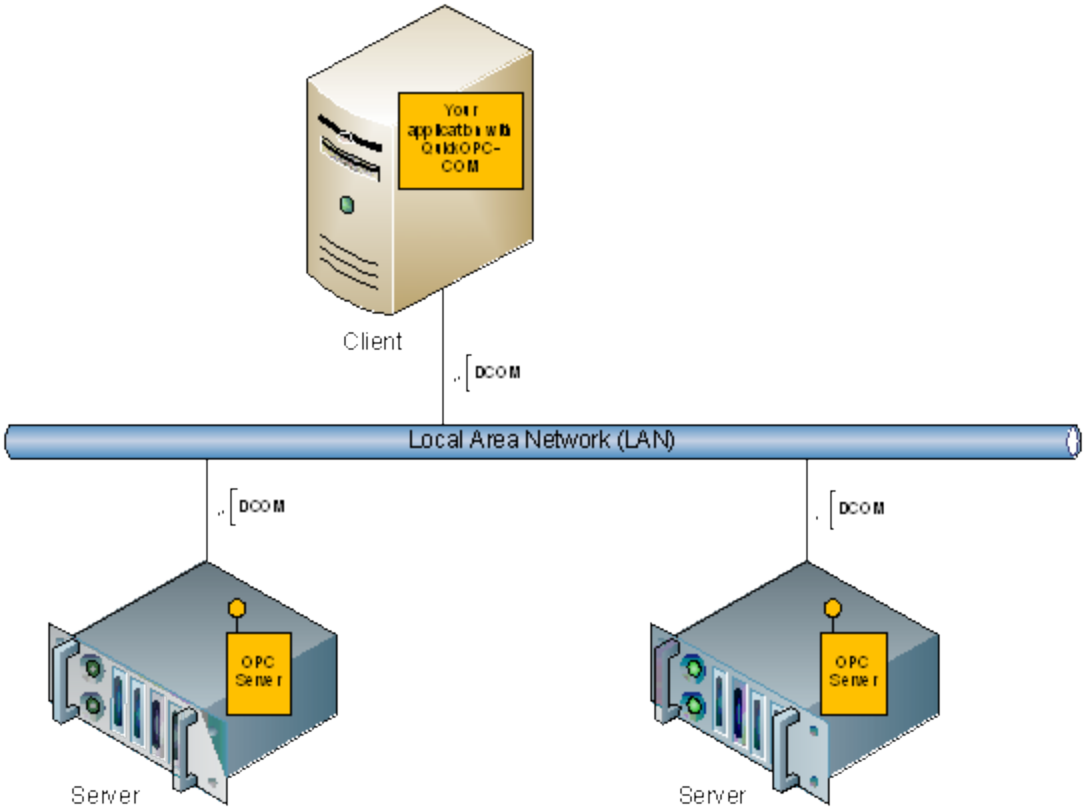


4.2.2 OPC Classic or OPC UA thick-client COM applications on LAN

The most typical use of OPC Data Client-COM or OPC Data Client-UA for COM involves a thick-client user application written in a tool or language that supports COM automation. This application uses the types from OPC Data Client-COM (or OPC Data Client-UA for COM) object model, and accesses data within OPC servers that are located on remote computers on the same LAN (Local Area Network). The communication with the target

OPC server is performed by Microsoft COM/DCOM technology.

The following picture shows how the individual pieces work together:



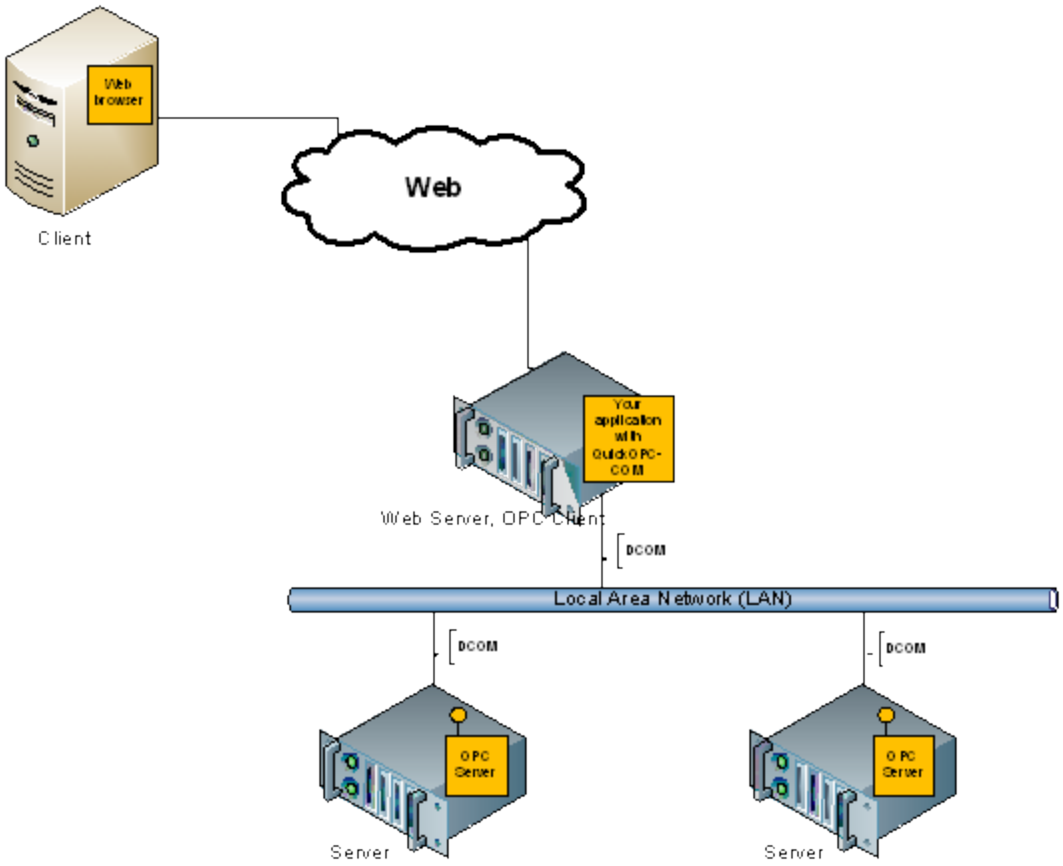
4.2.3 OPC Classic Web applications (server side)

The other typical use of OPC Data Client is to place it on the Web server inside a Web application. The Web application provides HTML pages to the client's browser, runs in a Web server, such as Microsoft IIS (Internet Information Server), and is written using tools and languages such as

- ASP/VBScript, PHP, Python and others – any language that supports COM automation, or
- ASP.NET, C#, Visual Basic.NET, or any other .NET language.

The Web application uses the types from OPC Data Client object model, and accesses data within OPC servers that are located on remote computers on the same LAN (Local Area Network). The communication with the target OPC server is performed by Microsoft COM/DCOM technology. No OPC-related (or indeed, COM or Microsoft-related) software needs be installed on the client machine; a plain Web browser such as Internet Explorer (IE) or FireFox is sufficient.

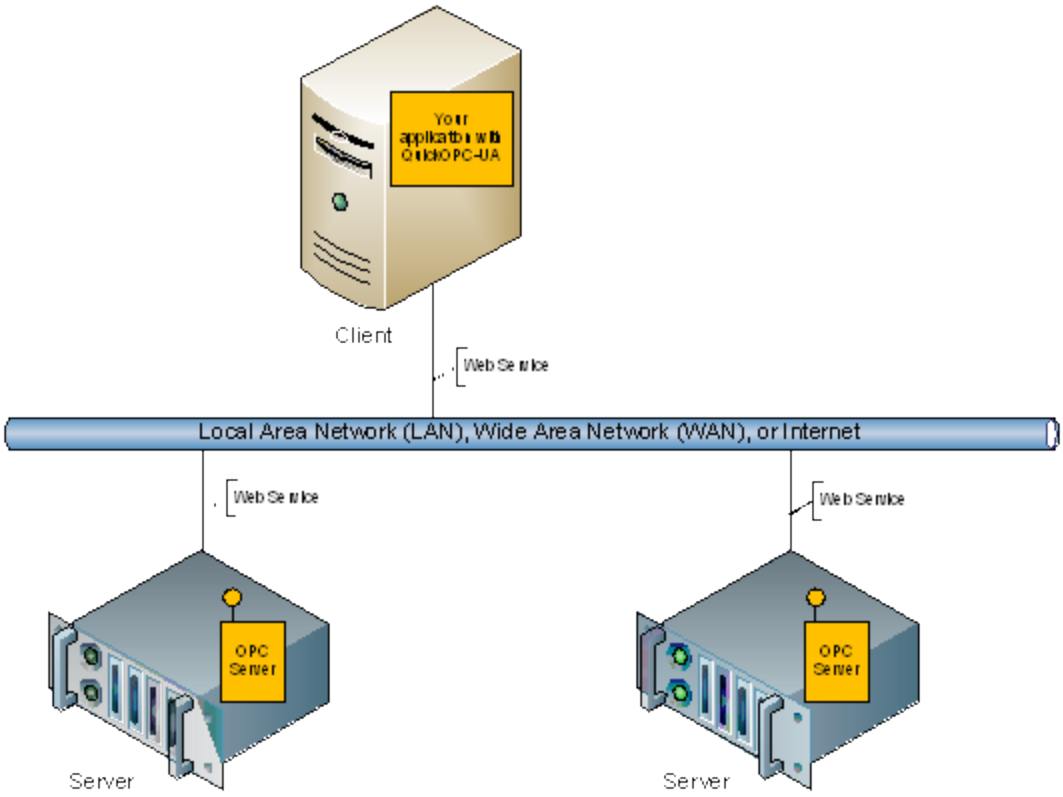
The following picture shows how the individual pieces work together:



4.2.4 OPC UA Thick-client applications on LAN, WAN or Internet

The most typical use of OPC Data Client-UA involves a thick-client user application written in one of the Microsoft .NET languages. This application uses the types from OPC Data Client-UA object model, and accesses data within OPC servers that are located on remote computers on the same LAN (Local Area Network) or on WAN (Wide Area Network) or on the Internet. The communication with the target OPC server is performed by means of Web Services.

The following picture shows how the individual pieces work together:

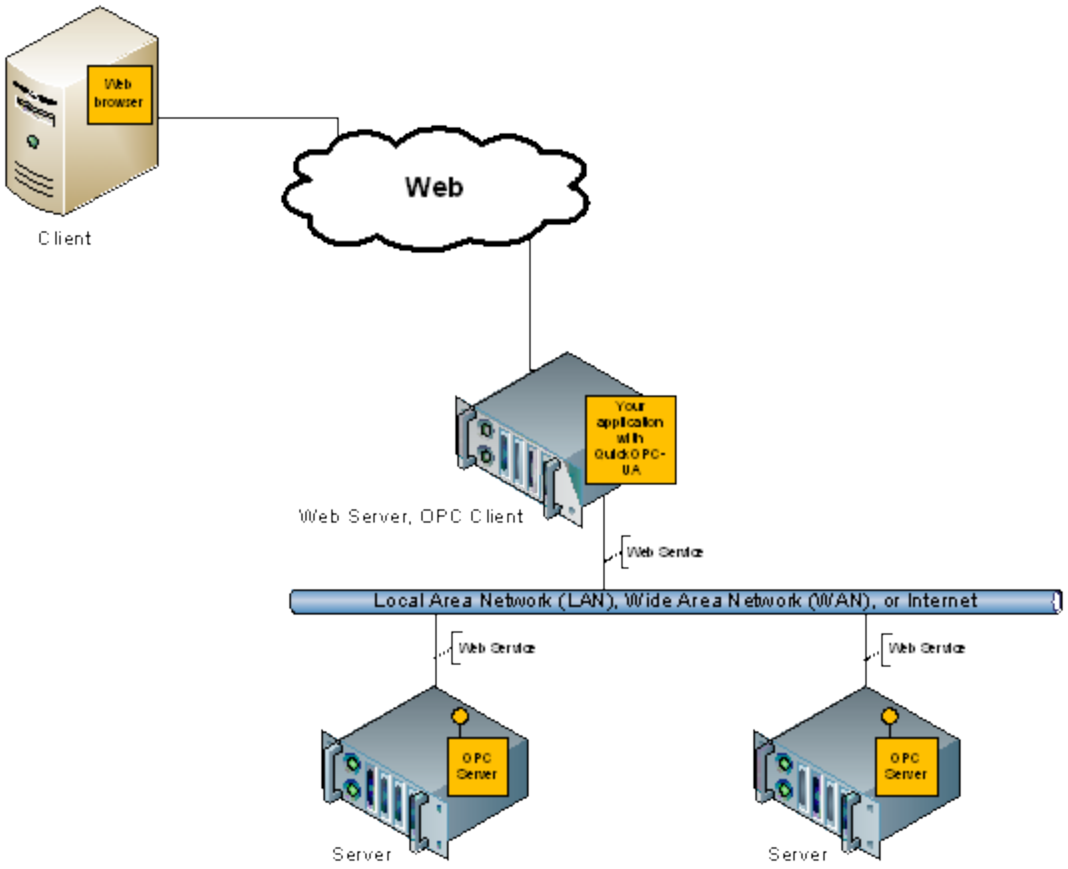


4.2.5 OPC UA Web applications (server side)

The other typical use of OPC Data Client-UA is to place it on the Web server inside a Web application. The Web application provides HTML pages to the client's browser, runs in a Web server, such as Microsoft IIS (Internet Information Server), and is written using tools and languages such as ASP.NET (using C#, Visual Basic.NET, or other .NET language).

The Web application uses the types from OPC Data Client-UA object model, and accesses data within OPC servers that are located on remote computers on the same LAN (Local Area Network). The communication with the target OPC server is performed by means of Web Services. No OPC-related (or indeed, Microsoft-related) software needs be installed on the client machine; a plain Web browser such as Internet Explorer (IE) or Firefox is sufficient.

The following picture shows how the individual pieces work together:



4.3 Referencing the Assemblies (.NET)

In .NET, in order to take advantage of the functionality provided by OPC Data Client, you need to reference its assemblies in your own project.

The following chapters describe how this is done. They provide **Overview of the Assemblies Available (Section 4.3.1)**, and discuss the various ways to reference assemblies:

- **Manual Assembly Referencing (Section 4.3.2)**
- **Assembly Referencing with Visual Studio Toolbox (Section 4.3.3)**
- **Assembly Referencing with NuGet (Section 4.3.4)**

In addition, **Licensing in Visual Studio (Section 4.3.5)** and the **.NET Namespaces (Section 4.3.6)** provided by OPC Data Client are described.

4.3.1 Overview of the Assemblies Available

Most projects will need:

- "**OPC Labs Base Library Core**" (OpcLabs.BaseLib.dll)
- for OPC Classic: "**OPC Labs EasyOPC "Classic" Library**" (OpcLabs.EasyOpcClassic.dll)
- for OPC UA: "**EasyOPC-UA Library**" (OpcLabs.EasyOpcUA.dll)

Some project will also need other assemblies, such as

- "**OPC Labs Base Library Forms**" (OpcLabs.BaseLibForms.dll) **(not in .NET Standard)**
- "**OPC Labs Base Library Presentation**" (OpcLabs.BaseLibPresentation.dll) **(not in .NET Standard)**
- "**OPC Labs EasyOPC Forms**" (OpcLabs.EasyOpcForms.dll) **(not in .NET Standard)**

You do not need to (and should not) ever reference the following assemblies:

- **EasyOPC "Classic" Raw Library (x64)** (App_Web_OpcLabs.EasyOpcClassicRaw.amd64.dll)
- **EasyOPC "Classic" Raw Library (x86)** (App_Web_OpcLabs.EasyOpcClassicRaw.x86.dll)

This does not mean that these assemblies are useless or are not used. They are used indirectly, by the relevant assemblies that you do reference, and they will copied over to the target automatically by the build tools.

In OPC UA PubSub, if you want to work with raw Ethernet packet mappings, or Wireshark capture files, the OpcLabc.Pcap assembly must be made available to your program - usually you achieve it by referencing this assembly, although it is not directly used by your code.

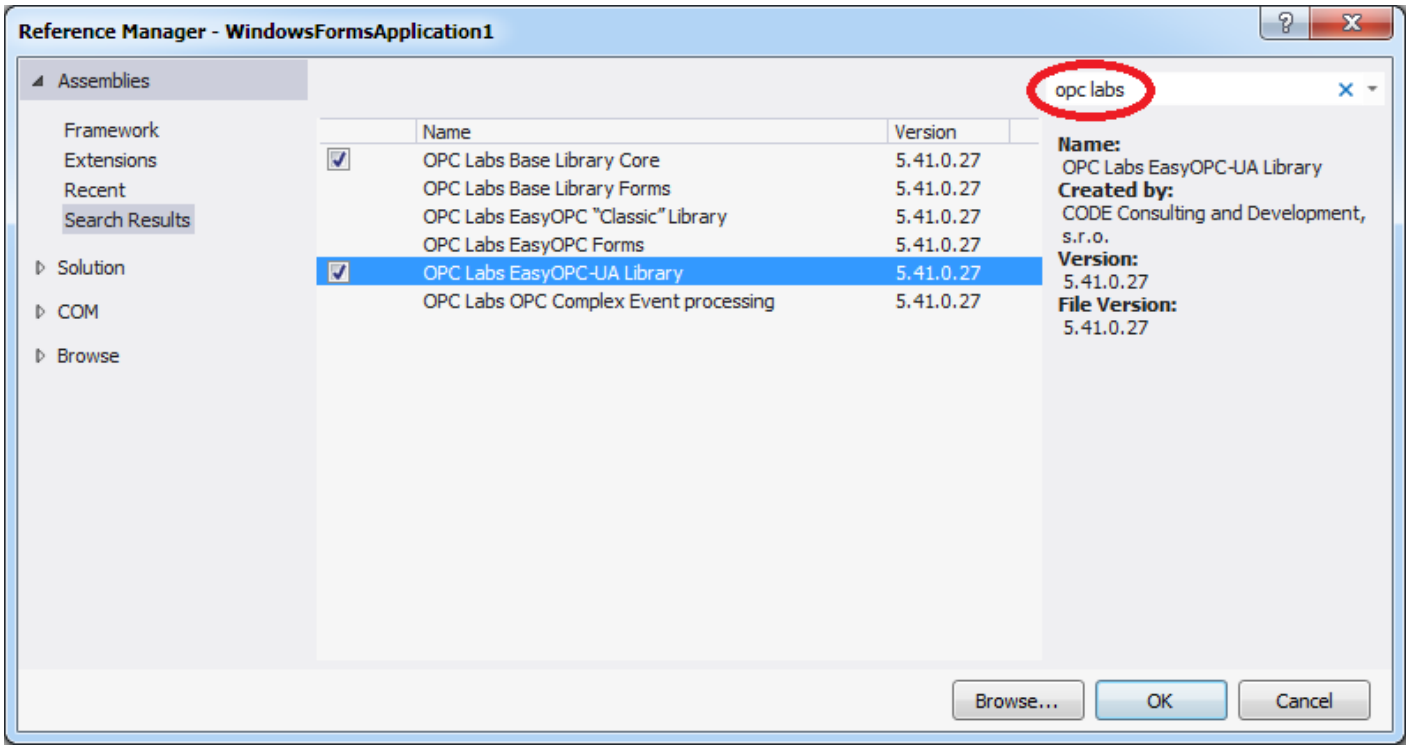
4.3.2 Manual Assembly Referencing

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

Your application first needs to reference the OPC Data Client.NET or OPC Data Client-UA assemblies in order to use the functionality contained in them. How this is done depends on the language and tool you are using:

- For Visual Basic in Visual Studio, select the project in Solution Explorer, and choose Project -> Add Reference command.
- For Visual C# in Visual Studio, select the project in Solution Explorer, and choose Project -> Add Reference command.
- For Visual C++ in Visual Studio, select the project in Solution Explorer, and choose Project -> References command.

You are then typically presented with an "Add Reference" dialog. The OPC Data Client "Classic" and OPC Data Client-UA assemblies are listed under its Assemblies (or .NET) tab. Select those that you need (see their descriptions in "Product Parts" chapter), and press OK.



You can easily see all OPC Data Client assemblies grouped together (as in the picture above), if you type "opc labs" into the search box. You can also sort the list alphabetically by Component Name, by clicking on the corresponding column header; then, scroll to the proper place in the list.

Notes:

- The list content is pulled in background, so you may need to wait a little until all components are listed and you can properly sort them or search for them.
- This documentation is for **OPC Data Client version 2020.2**. This is known as the *external version* of the product. The *internal version* number is **5.58**, and that is how the assemblies are listed in the Reference Manager.
- OPC Data Client.NET assemblies need the .NET Framework 4.7 or later. The dialog filters the components according to the target platform setting of you project. You will not see OPC Data Client.NET assemblies listed them listed if your project's target framework is not set to .NET Framework 4.7 or later.

4.3.3 Assembly Referencing with Visual Studio Toolbox

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

If you are using the Visual Studio Toolbox (described further below) to add instances of components to your project, the assembly references are created for you by Visual Studio automatically, when you drag the component onto the designer's surface.

This is the approach used in the "Getting Started" articles:

- **Making a first OPC Classic application using Live Binding in Windows Forms (Section 3.2.1.1)**
- **Making a first OPC Classic application using traditional coding (Section 3.2.1.2)**
- **Making a first OPC UA application using Live Binding in Windows Forms (Section 3.2.2.1)**
- **Making a first OPC UA application using Live Binding in WPF (Section 3.2.2.2)**
- **Making a first OPC UA application using traditional coding (Section 3.2.2.3)**

This technique only works for certain kinds of projects (such as Windows Forms or WPF) and certain kinds of project items in them.

4.3.4 Assembly Referencing with NuGet

If you have used NuGet to install a OPC Data Client package into your project, all necessary assembly references are done for you automatically. See **NuGet Packages (Section 3.1.2)** in the **Installation (Section 3.1)** chapter for details on how to install the NuGet packages into your project.

This is the main technique used for referencing assemblies in projects targeting .NET Standard development platform.

You may, however, sometimes want to *remove* certain assembly references later, if you do not need them. For example, the **OpcLabs.QuickOpc** package will automatically reference the assemblies needed for all supported OPC technologies, but if you do not need e.g. OPC "Classic" or OPC Unified Architecture at all, you may remove the corresponding assembly references from your project.

4.3.5 Licensing in Visual Studio

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

OPC Data Client uses, in part, .NET licensing model for component and controls. The internal working of it are largely invisible to you. The visible manifestation of the licensing model in Visual Studio is that the designer creates and maintains a file named **licenses.licx**, which contains a list of the licensed components, together with their assembly and version information. The file is also automatically updates when you upgrade to a newer version of build of OPC Data Client. You can safely rely on Visual Studio maintaining this file.

As opposed to other components and controls that use this licensing model in full, OPC Data Client license keys do not actually get embedded into your compiled project. The OPC Data Client license should always be installed onto the machine separately, using e.h. the License Manager utility of OPC Data Client.

4.3.6 .NET Namespaces

The OPC Data Client "Classic" and OPC Data Client-UA class libraries are made up of namespaces. Each namespace contains types that you can use in your program: classes, structures, enumerations, delegates, and interfaces.

All our namespaces begin with OpcLabs name. OPC Data Client "Classic" and OPC Data Client-UA define types mainly in namespaces described in tables under the links below.

- In OpcLabs.BaseLib Assembly
- In OpcLabs.BaseLibForms Assembly
- In OpcLabs.BaseLibPresentation Assembly
- In OpcLabs.EasyOpcClassic Assembly
- In OpcLabs.EasyOpcUA Assembly

You can use symbols contained in the namespaces by using their fully qualified name, such as OpcLabs.EasyOpc.DataAccess.EasyDAClient. In order to save typing and achieve more readable code, you will typically instruct your compiler to make the namespaces you use often available without explicit reference. To do so:

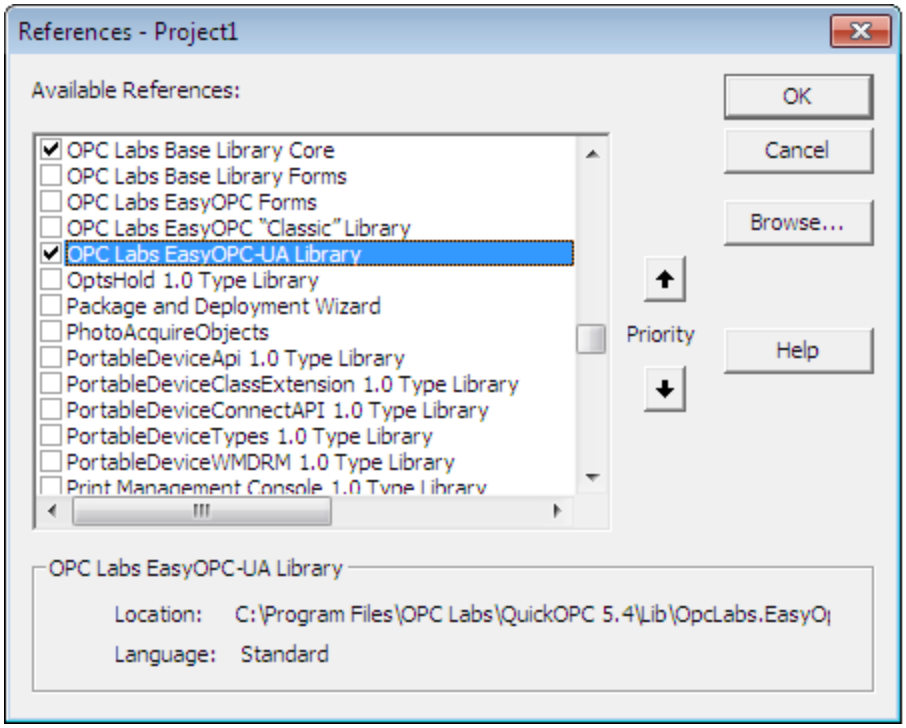
- In Visual Basic (VB.NET), place the corresponding Imports statements at the beginning of your code files.
- In Visual C#, place the corresponding using directives at the beginning of your code files.
- In Visual C++, place the corresponding using namespace directives at the beginning of your code files.
- In Visual F#, place the corresponding open directives at the beginning of your code files.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.4 Referencing the Components (COM)

Your application first needs to reference the OPC Data Client-COM or OPC Data Client-UA for COM components in order to use the functionality contained in them. How this is done depends on the language and tool you are using.

In Visual Basic 6.0 and Visual Basic for Applications (e.g. Microsoft Excel, Word, Access, PowerPoint, and many non-Microsoft tools), select Project -> References (or Tools -> References) from the menu. You are then presented with a **References** dialog. The OPC Data Client-COM type libraries should be listed in alphabetical order, prefixed with **"OPC Labs"** in their name so that you can easily find them grouped together:



Check the boxes next to the libraries you are referencing, and press OK.

Some tools provide a different user interface for referencing the components, while yet others do not have any user interface at all, and you need to write a source code statement that references the type library directly.

If the library you are looking for is not listed, it may be necessary to specify its location explicitly. Sometimes, simply restarting the Visual Basic 6.0 helps. Otherwise, press the Browse... button (or a corresponding UI element in other tools), and navigate to the **SDK\lib** subdirectory of the OPC Data Client product installation folder on your disk. Then, select the

type library file you are looking for (e.g. **OpcLabs.BaseLib.tlb** for "OPC Labs Base Library Core").

Note that referencing the type library in WSH (Windows Script Host) or ASP is only needed if you want to use certain features, such as the named constants included in the library. It is not necessary to reference the type library for simply instantiating the components and making methods calls, as for method calls, VBScript or JScript code in WSH or ASP can interrogate the created objects and use late binding to perform the calls.

It is also possible to do away with referencing the component in Visual Basic 6.0 and Visual Basic for Applications, and proceed simply to instantiating the object(s) as described further below, in a way similar to VBScript, but you would lose several features such as the early-binding, IntelliSense, and ability to use symbolic objects names and enumeration constants from the type library.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.4.1 COM Components for OPC Classic

In order to gain access to all COM objects of OPC Data Client "Classic", you need types described in the following type libraries:

Type Library Name	LIBID	Note
OPC Labs Base Library Core	ecf2e77d-3a90-4fb8-b0e2-f529f0cae9c9	
OPC Labs Base Library Forms	A0D7CA1E-7D8C-4D31-8ECB-84929E77E331	only for User Interface objects
OPC Labs EasyOPC "Classic" Library	1F165598-2F77-41C8-A9F9-EAF00C943F9F	
OPC Labs EasyOPC Forms	2C654FA0-6CD6-496D-A64E-CE2D2925F388	only for User Interface objects

For illustration and for your convenience, we have listed below examples of source code reference statements for the "OPC Labs EasyOPC "Classic" Library" in various languages and tools.

Language/tool	Statement
C++	#import"libid:1F165598-2F77-41C8-A9F9-EAF00C943F9F" // OpcLabs.EasyOpcClassic
WSH	<referenceguid="{1F165598-2F77-41C8-A9F9-EAF00C943F9F}" /> <!--OpcLabs.EasyOpcClassic-->
ASP	<!—METADATA TYPE="TypeLib" NAME="OPC Labs Easy "Classic" Library" UUID="{1F165598-2F77-41C8-A9F9-EAF00C943F9F}" -->

4.4.2 COM Components for OPC UA

In order to gain access to all COM objects of OPC Data Client-UA, you need types described in the following type libraries:

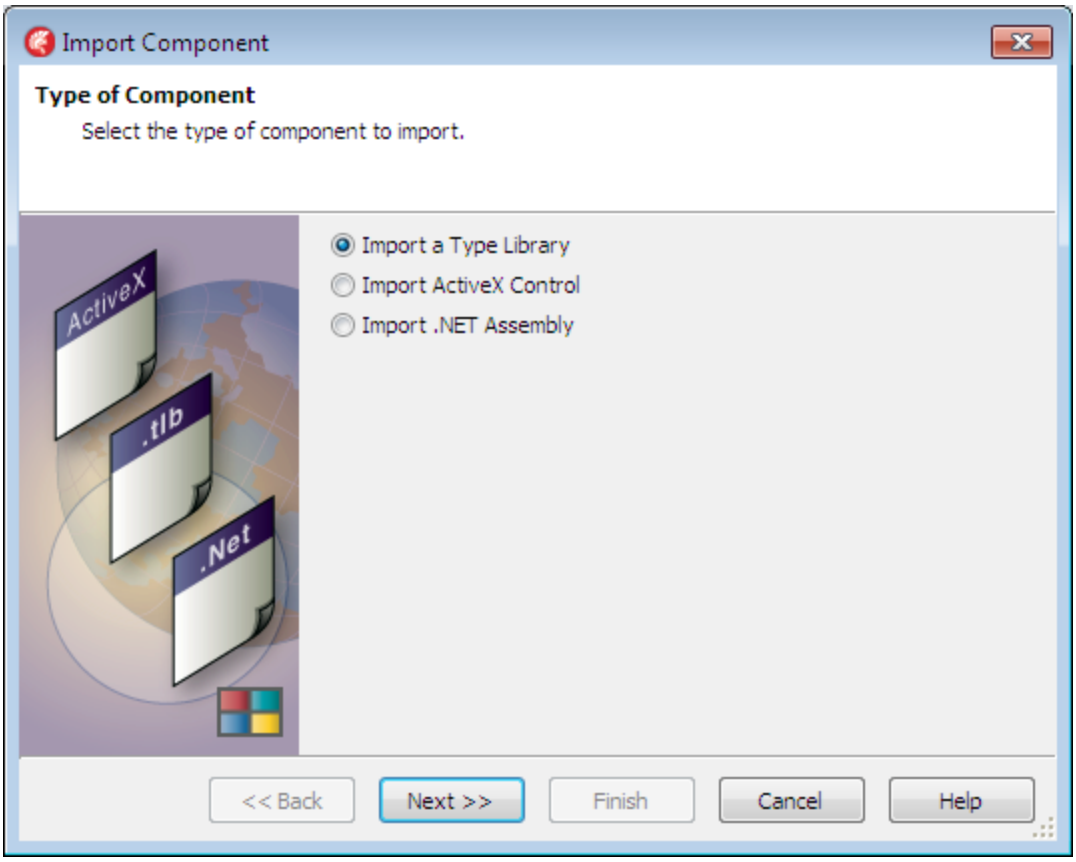
Type Library Name	LIBID	Note
OPC Labs Base Library Core	ecf2e77d-3a90-4fb8-b0e2-f529f0cae9c9	
OPC Labs Base Library Forms	A0D7CA1E-7D8C-4D31-8ECB-84929E77E331	only for User Interface objects
OPC Labs EasyOPC-UA Library	E15CAAE0-617E-49C6-BB42-B521F9DF3983	
OPC Labs EasyOPC Forms	2C654FA0-6CD6-496D-A64E-CE2D2925F388	only for User Interface objects

4.4.3 Importing Type Libraries to Delphi

Use the steps below to import OPC Data Client-UA for COM type libraries to Delphi XE7, Delphi XE8 or Delphi 10 Seattle.

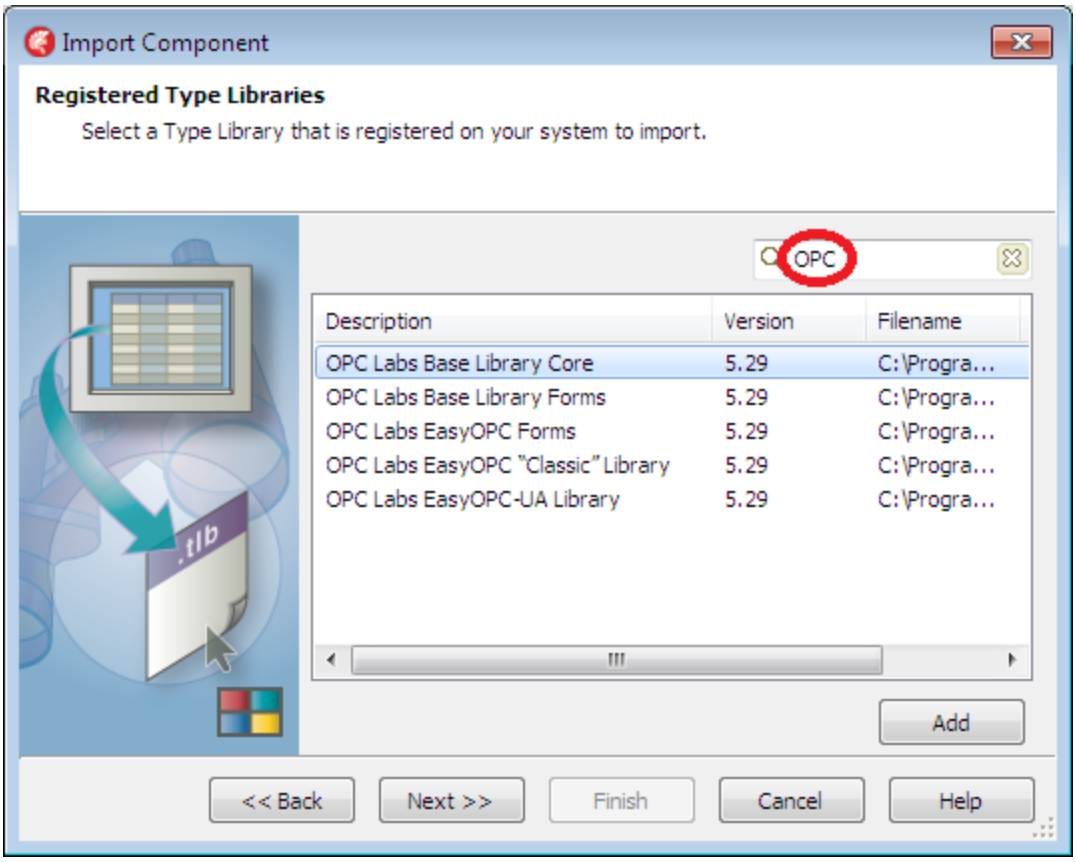
Note, however, that for the purpose of our examples, we already ship Delphi (XE7) components created from importing OPC Data Client type libraries together with the examples. It may be faster and safer to simply reuse them. They are located in the “Imports” subdirectory under the Delphi examples directory (**Examples-COM/OP/Imports** relatively from the OPC Data Client installation folder).

1. In Delphi, select command Component -> Import Component. The first page of the “Import Component” wizardappears, titled “Type of Component”:



Select “Import a Type Library” and press the Next button.

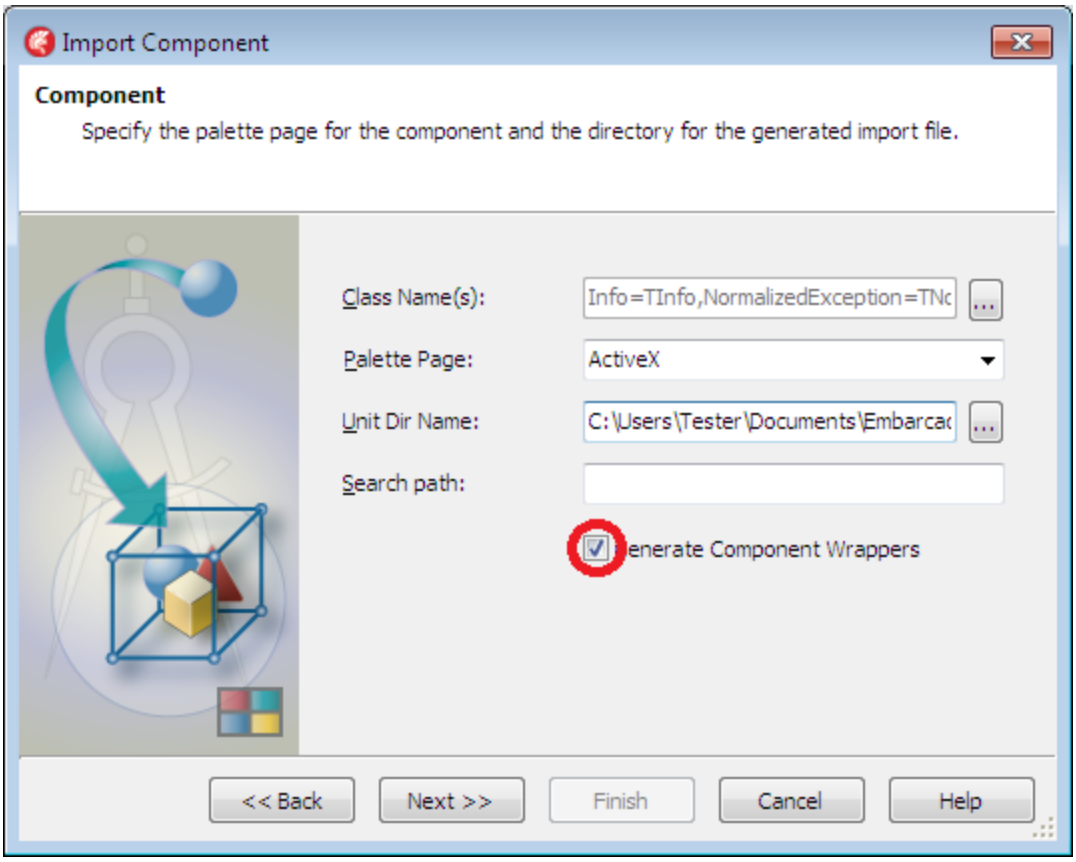
2. The page “Registered Type Libraries” appears:



You may need to resize the form to make it larger, and/or drag the column header dividers to make the columns wider.

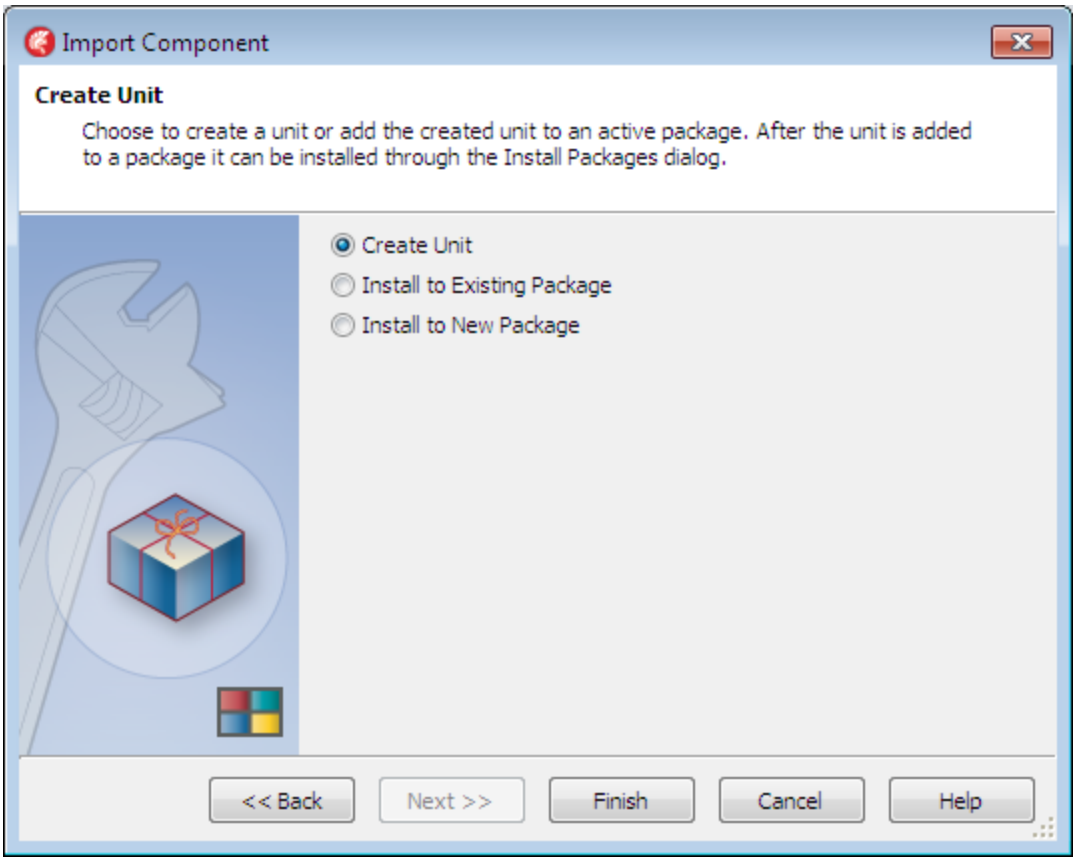
Type “OPC” into the search box, in order to narrow down the list of available type libraries. Select the type library you want to import, and press the Next button.

3. The page “Component” appears:



Check the “Generate Component Wrappers” box, and optionally modify the other settings as needed. Then, press the “Next button”.

4. The page “Create Unit” appears:



Select the option you need (typically, you can leave the "Create Unit" choice selected), and press the Finish button.

5. If you need to import another type library, repeat the process from Step 1.

> ⚠ Note that importing a type library may also silently regenerate (and overwrite) an existing imported component that the type library depends on, but then without the "Generate Component Wrappers" option – potentially messing things up. To avoid this, you can either
>
> a. make backup copies of existing components before importing any new type library, and restore the overwritten components afterwards, if needed; or
> b. import the type libraries in the reverse order of layering, starting with those that are at the top of the hierarchy and proceeding downwards: 1) OPC Labs EasyOPC Forms, 2) and 3) remaining OPC Labs libraries, 4) OPC Labs Base Library Forms and 5) OPC Labs Base Library Core; or
> c. if possible, simply use the pre-imported components that come with our examples, as described above.

4.4.4 Importing Type Libraries in Visual C++

There are several ways to consume the OPC Data Client component in Microsoft Visual C++. One of the easiest is to use the Compiler COM Support. This is done by placing #import directives into your code; the directives reference the OPC Data Client type libraries. The compiler converts the type libraries into C++ header files that describe the COM interfaces.

You can place the #import directives in your code one by one, but you can also simply #include a **QuickOpc.h** file provided with the product in the **SDK\include** subdirectory. This file imports all OPC Data Client type libraries and

declares wrappers for OPC Data Client interfaces and classes. Its content is as follows:

C++

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
// This files imports all QuickOPC libraries (and system libraries they require),
creating Compiler COM Support wrappers.
// It also contains additional definitions for creating event sinks.
#pragma once



// #import system libraries

// mscorlib
#pragma warning(push)
#pragma warning(disable:4278)    // 'ReportEvent': identifier in type library
'BED7F4EA-1A96-11D2-8F08-00A0C9A6186D' is already a macro; use the 'rename' qualifier
#if _MSC_VER >= 1300
#define IMPORT_MSCORLIB "libid:BED7F4EA-1A96-11D2-8F08-00A0C9A6186D"
#else
#define IMPORT_MSCORLIB "mscorlib.tlb"
#endif
#import IMPORT_MSCORLIB
#pragma warning(pop)
using namespace mscorlib;

// System.Drawing
#if _MSC_VER >= 1300
#define IMPORT_SYSTEM_DRAWING "libid:D37E2A3E-8545-3A39-9F4F-31827C9124AB"
#else
#define IMPORT_SYSTEM_DRAWING "System.Drawing.tlb"
#endif
#import IMPORT_SYSTEM_DRAWING
using namespace System_Drawing;

// System.Windows.Forms
#pragma warning(push)
#pragma warning(disable:4192)    // automatically excluding 'IDataObject' while
importing type library 'System.Windows.Forms.tlb'
#if _MSC_VER >= 1300
#define IMPORT_SYSTEM_WINDOWS_FORMS "libid:215D64D2-031C-33C7-96E3-61794CD1EE61"
#else
#define IMPORT_SYSTEM_WINDOWS_FORMS "System.Windows.Forms.tlb"
#endif
#import IMPORT_SYSTEM_WINDOWS_FORMS
#pragma warning(pop)
using namespace System_Windows_Forms;



// #import QuickOPC libraries

// OpcLabs.BaseLib
#if _MSC_VER >= 1300
```

```
#define IMPORT_OPCLABS_BASELIB    "libid:ecf2e77d-3a90-4fb8-b0e2-f529f0cae9c9"
#else
#define IMPORT_OPCLABS_BASELIB    "OpcLabs.BaseLib.tlb"
#endif
#import IMPORT_OPCLABS_BASELIB \
    rename("password", "Password") \
    rename("value", "Value")
using namespace OpcLabs_BaseLib;

// OpcLabs.BaseLibForms
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_BASELIBFORMS    "libid:A0D7CA1E-7D8C-4D31-8ECB-84929E77E331"
#else
#define IMPORT_OPCLABS_BASELIBFORMS    "OpcLabs.BaseLibForms.tlb"
#endif
#import IMPORT_OPCLABS_BASELIBFORMS
using namespace OpcLabs_BaseLibForms;

// OpcLabs.EasyOpcClassic
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_EASYOPCCLASSIC    "libid:1F165598-2F77-41C8-A9F9-EAF00C943F9F"
#else
#define IMPORT_OPCLABS_EASYOPCCLASSIC    "OpcLabs.EasyOpcClassic.tlb"
#endif
#import IMPORT_OPCLABS_EASYOPCCLASSIC \
    rename("itemId", "ItemId") \
    rename("machineName", "MachineName") \
    rename("requestedUpdateRate", "RequestedUpdateRate") \
    rename("serverClass", "ServerClass")
using namespace OpcLabs_EasyOpcClassic;

// OpcLabs.EasyOpcUA
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_EASYOPCUA    "libid:E15CAAE0-617E-49C6-BB42-B521F9DF3983"
#else
#define IMPORT_OPCLABS_EASYOPCUA    "OpcLabs.EasyOpcUA.tlb"
#endif
#import IMPORT_OPCLABS_EASYOPCUA \
    rename("applicationUriString", "ApplicationUriString") \
    rename("attributeData", "AttributeData") \
    rename("browsePath", "BrowsePath") \
    rename("endpointDescriptor", "EndpointDescriptor") \
    rename("expandedText", "ExpandedText") \
    rename("inputArguments", "InputArguments") \
    rename("inputTypeCodes", "InputTypeCodes") \
    rename("nodeDescriptor", "NodeDescriptor") \
    rename("nodeId", "NodeId") \
    rename("password", "Password") \
    rename("serverCapabilities", "ServerCapabilities") \
    rename("value", "Value")
using namespace OpcLabs_EasyOpcUA;

// OpcLabs.EasyOpcForms
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_EASYOPCFORMS    "libid:2C654FA0-6CD6-496D-A64E-CE2D2925F388"
#else
#define IMPORT_OPCLABS_EASYOPCFORMS    "OpcLabs.EasyOpcForms.tlb"
```

```
#endif
#import IMPORT_OPCLABS_EASYOPCFORMS
using namespace OpcLabs_EasyOpcForms;



// DISPID-s

// EasyAEClientConfigurationEvents
#define DISPID_EASYAECLIENTCONFIGURATIONEVENTS_LOGENTRY    1001

// EasyAEClientEvents
#define DISPID_EASYAECLIENTEVENTS_EVENTINGFAILURE 1001
#define DISPID_EASYAECLIENTEVENTS_NOTIFICATION 1002

// EasyDAClientConfigurationEvents
#define DISPID_EASYDACLIENTCONFIGURATIONEVENTS_LOGENTRY    1001

// EasyDAClientEvents
#define DISPID_EASYDACLIENTEVENTS_EVENTINGFAILURE 1001
#define DISPID_EASYDACLIENTEVENTS_ITEMCHANGED 1002

// EasyUAClientConfigurationEvents
#define DISPID_EASYUACLIENTCONFIGURATIONEVENTS_LOGENTRY    1001

// EasyUAClientEvents
#define DISPID_EASYUACLIENTEVENTS_DATACHANGENOTIFICATION  1001
#define DISPID_EASYUACLIENTEVENTS_EVENTINGFAILURE  1002
#define DISPID_EASYUACLIENTEVENTS_EVENTNOTIFICATION  1003
#define DISPID_EASYUACLIENTEVENTS_SERVERCONDITIONCHANGED  1004

// EasyUASubscriberEvents
#define DISPID_EASYUASUBSCRIBEREVENTS_DATASETMESSAGE  1001
#define DISPID_EASYUASUBSCRIBEREVENTS_EVENTINGFAILURE  1002
#define DISPID_EASYUASUBSCRIBEREVENTS_RESOLVERACCESS  1003
#define DISPID_EASYUASUBSCRIBEREVENTS_SUBSCRIPTIONRESOLVED  1004
```

The file also makes sure (using the **rename** attributes) that the symbols are properly cased, and adds useful #define-s for defining event sinks.

4.5 Conventions

Conventions

From OPC Labs Knowledge Base
Jump to navigation Jump to search

With the help of various conventions, we try to make it easier to find and discover the information needed, and understand the components' internal consistency.

Contents

Date and Time Conventions

- All OPC-related properties that are of DateTime type (and are consistently in UTC) have corresponding properties with a "Local" postfix, which contain the same value but expressed in local time. For example, DAVtq.Timestamp contains a timestamp in UTC, whereas its DAVtq.TimestampLocal contains the same in local time.

Naming Conventions

Common Naming Conventions

- In OPC-UA, the name part "MonitoredItem" is used wherever the type or member applies to both Data Access and Alarms and Conditions, whereas the names that use the term "DataChange" apply to Data Access only.

Type Names

In addition to being compliant with common Microsoft recommendations for names, and in the .NET part with Microsoft .NET Framework guidelines for names, we follow certain additional naming conventions:

- Types which are specific to very simplified ("easy") model for working with OPC start with the prefix Easy.
- Types which are specific to OPC Data Access start with DA (or EasyDA) prefix, types which are specific to OPC Alarms and Events start with AE (or EasyAE) prefix. Types that are shared among multiple OPC specifications do not have these prefixes.
- Types which are specific to OPC Unified Architecture start with UA (or EasyUA) prefix. Types that are shared among multiple OPC specifications do not have these prefixes.

Note that the second and third conventions work in addition and together with the fact that there are separate DataAccess, AlarmsAndEvents, and UA namespaces for this purpose too.

The above described conventions also give you a bit of hint where to look for a specific type, if you know its name. For example, if the name starts with DA or EasyDA, the type is most likely in the OpcLabs.EasyOpc.DataAccess namespace. Without any special prefix, the type is likely to be in the OpcLabs.EasyOpc or OpcLabs.BaseLib namespace.

Interface Names in COM

As opposed to .NET, every object access in COM is solely made over its interfaces. In COM, after the object is instantiated, you never "see" the object as such – only the interfaces it has. Therefore, at least one additional interface (besides the COM-defined IUnknown and IDispatch) is needed on each .NET exposed to COM.

We consistently use following conventions in COM:

- Incoming interfaces have a form of *_XXXX*, where *XXXX* is the distinguishing part, e.g. _EasyUAClient, for the EasyUAClient object.
- Outgoing interface (for events) have a form of D*XXXX*Events, where *XXXX* is the distinguishing part, e.g.

DEasyUAClientEvents, for the EasyUAClient object.

The incoming interface are dual interfaces. The outgoing interfaces are dispatch-only interfaces, hence the use of 'D' as a prefix.

ProgIDs (COM)

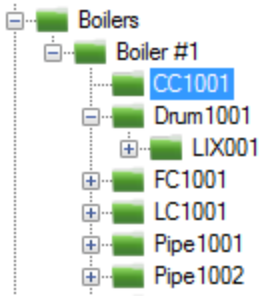
In COM, all our objects have ProgIDs that are identical to their qualified type names in .NET. For example, the main EasyUAClient component resides in the OpcLabs.EasyOpc.UA namespace, and its fully qualified name and therefore also its ProgID is "OpcLabs.EasyOpc.UA.EasyOpcUA". The main EasyDAClient component resides in the OpcLabs.EasyOpc.DataAccess namespace, and its fully qualified name and therefore also its ProgID is "OpcLabs.EasyOpc.DataAccess.EasyOpcDA". This convention makes it easy to instantiate COM objects, based on the namespace-qualified names of their .NET counterparts.

Coloring Conventions

We maintain visual consistency in the product by using specific colors for certain functionality areas. This is typically seen in various icons, images and bitmaps.

Green color is used for OPC Data Access.

For example:

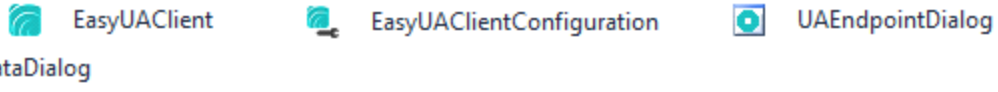


Orange color is used for OPC Alarms&Events.

For example:

Turquoise color is used for OPC Unified Architecture. Because OPC-UA supports multiple functionalities, the turquoise color is sometimes used in combination with other colors, such as green for Data, or orange for Events.

For example:



Retrieved from "https://kb.opclabs.com/index.php?title=Conventions&oldid=1651"
Category:

- QuickOPC

Navigation menu

4.6 Components and Objects

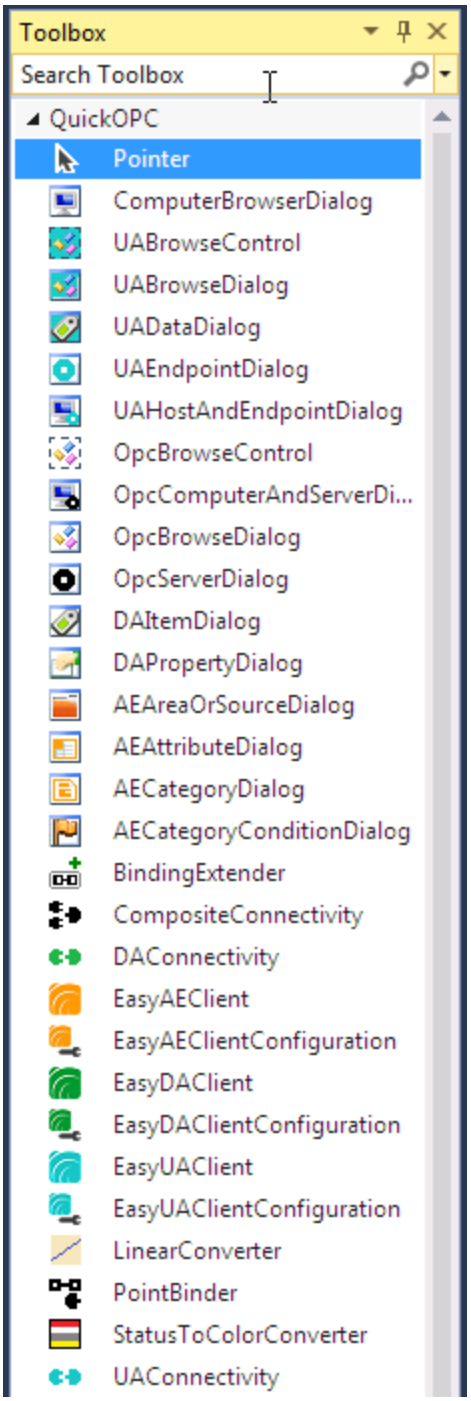
OPC Data Client is a library of many objects. They belong into two basic categories: *Computational objects* provide "plumbing" between OPC servers and your application. They are invisible to the end user. *User interface objects* provide OPC-related interaction between the user and your application.

In This Topic

Components and Objects in .NET Framework

If you are working with an IDE (Integrated Development Environment) such as Visual Studio, the main OPC Data Client components appear in the Visual Studio Toolbox (see example picture below). Note that, however, most OPC Data Client objects can be used directly from the code, and that the Toolbox only offers the components for certain environments, such as Windows Forms; in other environments, you can still use OPC Data Client, but you will instantiate the components directly. Also note that there are many more objects to OPC Data Client than just those classified as designer components and shown on the Toolbox.

The OPC Data Client components in the Toolbox look similar to this:



If, for some reason, some or all of the OPC Data Client components do not appear in the toolbox, see **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

Components and Objects in .NET Standard

In .NET Standard, you instantiate the OPC Data Client objects directly, using the constructs available in the language you are using - for a example, the new keyword in C#, the New keyword in VB.NET, etc.

Components and Objects in COM

In OPC Data Client-COM, the precise way to instantiate the object depends on the programming language you are using. For example:

- In Visual Basic 6.0 and Visual Basic for Applications, if you have referenced the component(s), use the New keyword with the appropriate class name.
- In VBScript (or in VB if you have not referenced the component), use the CreateObject function with the ProgID of the class.
- In JScript, use the 'new ActiveXObject(...)' construct, with the ProgID of the class.
- In C++, create the smart interface pointer passing '__uuidof(...)' to the constructor, using the CLSID of the class.
- In Delphi (Object Pascal), call .Create on the class type create by importing the type library.
- In PHP, use the 'new COM(...)' construct, with the ProgID of the class.
- In Python, use 'win32com.client.Dispatch(...)' construct, with the ProgID of the class.
- In Visual FoxPro, use the CREATEOBJECT function with the ProgID of the class.

4.6.1 Computational Objects

For easy comprehension, there is just one computational object that you need to start with, for each OPC specification:

- For OPC Data Access, it is the EasyDAClient object.
- For OPC Alarms and Events, it is the EasyAEClient object.
- For OPC Unified Architecture, it is the EasyUAClient object.
- For OPC UA PubSub, it is the EasyUASubscriber object.

All other computational objects are helper objects (see further in this chapter). You achieve all OPC computational tasks by calling methods on one of these objects. The remainder of this paragraph describes the use of EasyDAClient object; the steps for EasyAEClient or EasyUAClient object are similar.

In order to be able to use one of these objects, you need to instantiate it first.

In OPC Data Client "Classic" and OPC Data Client-UA, there are two methods that you can use:

- Write the code that creates a new instance of the object, using an operator such as New (Visual Basic), new (Visual C#) or gcnew (Visual C++).
- Drag the component from the Toolbox to the designer surface. This works because they are derived from System.ComponentModel.Component (but is not limited to be used as a "full" component only). You can use this approach in Windows Forms applications, in Console applications and some other types of applications if you first add a designable component to your application. The designer creates code that instantiates the component, assigns a reference to it to a field in the parent component, and sets its properties as necessary. You can then use designer's features such as the Properties grid to manipulate the component.

Note: In addition, there is a static Create() method on each EasyXXClient or EasyXXSubscriber class. This methods works the same as parameterless constructor. For example (in C#), EasyUAClient.Create() is the same as new EasyUAClient().

For OPC Data Client-COM and OPC Data Client-UA for COM, the following table contains information needed to instantiate the objects.

Class Name	CLSID	ProgID

EasyAEClient	3643545B-221F-4960-BF47-8A4DDEC81A67	OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient
EasyDAClient	6B0B5307-BCB6-4953-A832-BFCF952F7561	OpcLabs.EasyOpc.DataAccess.EasyDAClient
EasyUAClient	54AFB0EA-9809-4D1D-AFBE-0EC164C59A45	OpcLabs.EasyOpc.UA.EasyUAClient
EasyUASubscriber	EDC1F10E-3FC6-4604-9BC6-4FFF579D271A	OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber

As described in the "Naming Conventions", in OPC Data Client-COM, all objects have ProgIDs that are identical to their qualified type names in .NET. For example, the ProgID of the main EasyUAClient is therefore "OpcLabs.EasyOpc.UA.EasyOpcUA".

For COM development, CLSIDs of all objects and IIDs of all interfaces can also be found in the Reference documentation (look for the GuidAttribute on the classes and interfaces).

4.6.1.1 Interfaces and Extension Methods

In OPC Data Client.NET, the main methods constituting the functionality of EasyDAClient, EasyAEClient, and EasyUAClient components are actually implementations of IEasyDAClient, IEasyAEClient, and IEasyUAClient interfaces, respectively. The remaining methods and method overloads, which simply build upon the core interface methods, are implemented as extension methods on the interface. Also, at many places, arguments and properties that accept one of the IEasyXXClient interfaces, instead of a concrete component.

In most languages (certainly in C# and Visual Basic), this design leads to the same syntax as if all the methods were implemented directly on the core concrete object. The approach that we hace chosen allows to supply a different implementation of the component's interface where needed, and is currently used for simulation purposes in the browsing controls and dialogs.

4.6.1.2 Isolated Clients and Subscribers

Normally, instances of a client or subscriber object (such as EasyDAClient, EasyAEClient, EasyUAClient) act commonly, sharing the same connections to the target OPC servers (or to the message-oriented middleware, in case of PubSub), and also many common parameters. This way, you can create a large number of these instances and work with them in an easy way, without having to worry about negative effects on the target. If, for example, you create two instances of an EasyDAClient object, and subscribe to some OPC items in each of these instances, only one connection will be created to the OPC server.

This approach works well for most applications, and it allows easy coding in scenarios such as Web development, where each page request may require OPC operations, and the requests may be coming in quick succession and in large numbers, and even processed in parallel.

In some cases, however, you may want to have a dedicated connection to the OPC server, or have more control over the parameters of the connection. In such case, you can set the Isolated property of the EasyXXClient or EasyXXSubscriber object to 'true'. By doing so, operations invoked on this instance of the client object will work with their own connection to the target OPC server, or message-oriented middleware. We say that the client (subscriber) object is *isolated*.

If you create more such isolated client objects, each of them will work with its own connection. Obviously you need to be more careful with this approach, in order to keep the load on the target OPC server in reasonable limits.

4.6.1.3 Shared Instance

Instead of explicitly instantiating the EasyDAClient (or EasyAEClient, EasyUAClient, EasyUASubcriber) objects, you can also use a single, pre-made instance of it, resulting in shorter code. You can access it as a SharedInstance static property on the class, and it contains a default, shared instance of the client object.

Use this property with care, as its usability for larger projects is limited. Its main use is for testing and for non-library application code where just a single instance is sufficient.

If you plan to use events on the EasyXXClient or EasyXXSubscriber component, the shared instance is not suitable for Windows Forms, WPF or similar environments, where a specific SynchronizationContext may be used with each form.

We also do not recommend using the shared instance for library code (if you are developing a Class Library project), due to conflicts that may arise if your library sets some instance parameters which may not be the same as what other libraries or the final application expect.

In OPC Data Client-COM, you can access the shared instance through the EasyXXClientConfiguration (or EasyXXSubscriberConfiguration) object.

4.6.2 User Interface Objects

In OPC Data Client "Classic" and OPC Data Client-UA, you instantiate a user interface object in Windows Forms applications by dragging the appropriate component from the Toolbox to the designer surface. The designer creates code that instantiates the component, assigns a reference to it to a field in the parent component, and sets its properties as necessary. You can then use designer's features such as the Properties grid to manipulate the component.

If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

In OPC Data Client-COM and OPC Data Client-UA for COM, you can use some of the user interface objects (namely, the dialogs – but not the controls) as well. Create the objects as any other OPC Data Client objects on the COM platforms. The convention described earlier (with computational objects) always applies, and therefore the ProgIDs of the user interface objects are the same as the namespace-qualified names of corresponding .NET types.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.6.3 Helper Types

The types described here do not directly perform any OPC operations, but are commonly used throughout OPC Data Client for properties and method arguments.

4.6.3.1 Time Periods

Many method arguments and properties describe time periods, such as update rates, delays, timeouts etc. For

consistency, they are all integers, and they are expressed as number of milliseconds. In OPC UA, an exception to this rule is for the maximum age of value (for reading), which is a floating point number in order to allow a finer precision.

Some method arguments and properties (but only some – see Reference documentation for each method argument or property) allow a special value that represents an "infinite" time period.

In OPC Data Client "Classic" and OPC Data Client-UA, the value for "infinite" time period is equal to Timeout.Infinite (from System.Threading namespace).

Note: Time periods should not be confused with absolute time information, which is usually expressed by means of DateTime structure.

In OPC Data Client-COM, the value for "infinite" time period is equal to -1.

Note: Time periods should not be confused with absolute time information, which is usually expressed by means of Windows DATE data type.

4.6.3.2 Data Objects

The data objects hold the actual data made available by the OPC technology.

4.6.3.2.1 Quality in OPC Classic

OPC Classic represents a quality of a data value by several bit-coded fields. OPC Data Client encapsulates the OPC quality in a DAQuality class. The bit fields in OPC quality are inter-dependent, making it a bit complicated to encode or decode it. The DAQuality type takes care of this complexity. In addition, it offers symbolic constants that represent the individual coded options, and also has additional functionality such as for converting the quality to a string.

The following table attempts to depict the elements of DAQuality and their relations:

<table>
<tr><td colspan="3">DAQuality</td></tr>
<tr><td>property DAQualityChoice QualityChoiceBitField { get; }
property bool IsBad { get; }
property bool IsGood { get; }
property bool IsUncertain { get; }</td><td>SubStatus</td><td rowspan="2">property DALimitChoice LimitBitField { get; set; }</td></tr>
<tr><td colspan="2">property DAStatusChoice StatusBitField { get; set; }
SetQualityAndSubStatus(...)</td></tr>
</table>

You can see that the StatusBitField is actually consisted of QualityChoiceBitField, and a SubStatus. But the semantics of SubStatus is highly dependent on QualityChoiceBitField, and therefore the SubStatus cannot be accessed separately. For the same reason, you cannot directly set the QualityChoiceBitField without providing a value for SubStatus at the same time. Instead, you can call SetQualityAndSubstatus method to modify the two fields at the same time.

Note that OPC Alarms and Events specification "borrows" the quality type from OPC Data Access, and therefore the DAQuality type is used with OPC Alarms and Events as well.

Constants for DAQuality values are contained in the DAQualities enumeration.

4.6.3.2.2 Value, Timestamp and Quality (VTQ) in OPC Classic

The combination of data value, timestamp and quality (abbreviated sometimes as VTQ) is common in OPC. This combination is returned e.g. when an OPC item is read. Note that according to OPC specifications, the actual data value is only valid when the quality is Good or Uncertain (with a certain exception).

OPC Data Client has a DAVtq object for this combination. The object provides access to individual elements of the VTQ combination, and also allows common operations such as comparisons. It can also be easily converted to a string containing textual representation of all its elements.

If you only want a textual representation for a data value from the VTQ, use the DisplayValue method. This is recommended over trying to extract the Value property and converting it to string, as you will automatically receive an empty string if the value is not valid according to OPC rules (e.g. Bad quality), and a null reference case is handled as well.

4.6.3.2.3 Variant Type (VarType) in OPC Classic

In some places in OPC, your code needs to indicate which type of data you expect to receive back, or (in the opposite direction), you receive indication about which type of data certain piece of information is. OPC uses Windows VARTYPE for this (describes a data contained in Windows VARIANT).

OPC Data Client.NET gives you a .NET encapsulation for indicating variant data types, so that you do not have to look up and code in the numeric values of Windows VARTYPE. Instead, wherever you see that a method argument, a property, or other element is of VarType type, you can supply one of the constants defined in the VarTypes enumeration. For example, VarTypes.I2 denotes a 16-bit signed integer, VarTypes.R4 denotes a 32-bit float, and VarTypes.BStr denotes a string. For arrays of values, use constants named VarTypes.ArrayOfXXXX (e.g. VarTypes.ArrayOfI4), or (if the element type is determined in the run time) the VarType.MakeArrayType method.

Note: Microsoft.NET framework contains a similar type, System.Runtime.InteropServices.VarEnum. The types have some similarities, but should not be confused.

OPC Data Client-COM gives you an enumeration for indicating variant data types, so that you do not have to look up and code in the numeric values of Windows VARTYPE. Instead, wherever you see that a method argument, a property, or other element accepts a data type, you can supply one of the constants defined in the VarTypes enumeration. For example, I2 denotes a 16-bit signed integer, R4 denotes a 32-bit float, and BStr denotes a string. For arrays of values, use logical 'or' to combine the element type with the Array constant. Note, however, that the enumeration symbols are only accessible if you reference or import the EasyOPC Type Library, and not all languages and tools are capable of doing it.

4.6.3.2.4 OPC UA Status Code

OPC represents a result of a service or operation in a status code, which consists of several bit-coded fields. OPC Data Client-UA encapsulates the status code in a UAStatusCode class. The bit fields in status code are inter-dependent, making it a bit complicated to encode or decode it. The UAStatusCode type takes care of this complexity. In addition, it offers symbolic constants that represent the individual coded options, and also has additional functionality such as for converting the status code to a string.

The following table attempts to depict the elements of UAStatusCode and their relations, together with related properties and methods:

<table>
<tr><td colspan="6">UAStatusCode
property InternalValue</td></tr>
<tr><td colspan="2">property CodeBits</td><td colspan="4">property FlagBits</td></tr>
<tr><td>property Severity</td><td>property Condition</td><td>property StructureChanged</td><td>property SemanticsChanged</td><td rowspan="2">property InfoType
property HasDataValueInfo</td><td colspan="2">if HasDataValueInfo == true:</td></tr>
<tr><td colspan="4"></td><td>property LimitInfo</td><td>property Overflow</td></tr>
</table>

The CodeBits part represents the numeric value of an error or condition, whereas the FlagBits part provides more information on the meaning of the status code.

OPC Data Client-UA provides a UACodeBits class that contains constants for errors and conditions defined by the OPC-UA specifications. You can compare the resulting CodeBits to the constants if your code needs to look for a specific error or condition.

The semantics of some parts is dependent on InfoType, and therefore the LimitInfo and Overflow properties can only be accessed if the InfoType has a corresponding value, which is indicated by the value of HasDataValueInfo property.

4.6.3.2.5 OPC UA Attribute Data

The combination of data value, source and serve timestamps and status code is common in OPC-UA. This combination is returned e.g. when an attribute of an OPC node is read. Note that according to OPC specification, the actual data value is only valid with certain status codes (typically, when the status is Good).

OPC Data Client-UA has a UAAttributeData object for this combination. The object provides access to individual elements of this combination, and also allows common operations such as comparisons. It can also be easily converted to a string containing textual representation of all its elements.

If you only want a textual representation for a data value from the attribute data, use the DisplayValue method. This is recommended over trying to extract the Value property and converting it to string, as you will automatically receive an empty string if the value is not valid according to OPC rules (e.g. Bad status), and a null reference case is handled as well.

4.6.3.3 Argument Objects

Argument objects are "holders" that contain all arguments necessary to perform certain operation. OPC Data Client has method overrides that accept individual arguments, but also method overrides that accept argument object as an input. The argument object can be reused with other operations without having to specify individual arguments again, and it can also participate in multiple-operation methods where an array of argument objects is accepted on input, and an array of result objects is returned on output.

All argument objects derive from OperationArguments.

4.6.3.4 Result Objects

Result objects are returned by methods that work on multiple elements simultaneously such as EasyDAClient.ReadMultipleItems, or EasyUAClient.ReadMultiple. Such methods return an array of OperationResult objects, or an array of objects derived from OperationResult. This approach is chosen, among other reasons, because the method cannot throw an exception if an operation on a single element fails (as operations on other elements might have succeeded).

Each OperationResult has an Exception property, which indicates the outcome of the operation. If the Exception is a null reference, the operation has completed successfully. There is also a State property, which contains user-defined information that you have passed to the method.

The objects derived from OperationResult have additional properties, which contain the actual results in case the operation was successful. Such objects are e.g. ValueResult (contains a data value), DAVtqResult (contains value, timestamp, and quality combination for OPC "Classic"), or UAAttributeDataResult (contains value, timestamps, and status code combination for OPC-UA).

The OperationResult object (and therefore all result objects derived from it as well) has a Diagnostics property, which contains a collection of warning and informational diagnostics entries gathered, performing the requested operation. You can use this collection to retrieve additional information about the outcome of the operation.

The additional OperationResult.DiagnosticsSummary string contains a textual summary of diagnostics information, one message per line.

Note that diagnostics information in the OperationResult is gathered and available both in case of success, and in case of

a failure.

Currently, the diagnostics information is only available for operations on OPC Unified Architecture (OPC-UA), i.e. not for OPC "Classic".

4.6.3.5 Element Objects

Element objects contain all information gathered about some OPC entity. They are typically returned by browsing methods.

There are following types of element objects for OPC Classic:

- ServerElement object contains information gathered about an OPC Classic (OPC COM or OPC XML) server.
- AEAttributeElement contains information gathered about an OPC Alarms and Events attribute.
- AECategoryElement contains information gathered about an OPC Alarms and Events category.
- AEConditionElement contains information gathered about an OPC Alarms and Events condition.
- AENodeElement object contains information gathered about an OPC node (areas or source in OPC Alarms and Events server's address space).
- AESubconditionElement contains information gathered about an OPC Alarms and Events subcondition.
- DANodeElement object contains information gathered about an OPC node (branch or leaf in OPC Data Access server's address space).
- DAPropertyElement contains information gathered about an OPC Data Access property.

There are following types of element objects for OPC UA:

- UADiscoveryElement object contains information gathered about an OPC-UA application (OPC-UA server).
- UANodeElement object contains information gathered about an OPC node (in OPC server's address space). The node can represent e.g. a branch in the organizing tree of nodes, or a process value, or a property.

Element objects are also returned when you invoke one of the common OPC dialogs for selecting OPC server, OPC-DA item or an OPC-DA property.

4.6.3.6 Descriptor Objects

A *descriptor object* contains information that fully specifies certain OPC entity (but does not contain any "extra" information that is not needed to identify it uniquely). Descriptor objects are used by some method overloads to reduce the number of individual arguments, and to organize them logically.

There are following types of descriptor objects for OPC Classic:

- ServerDescriptor contains information necessary to identify and connect to an OPC Classic server, such as the server's ProgID.
- AENodeDescriptor contains information necessary to identify a node (area or source) in OPC A&E address space, such as Node Id (a string).
- DANodeDescriptor contains information necessary to identify a node (branch or leaf) in OPC DA address space, such as Node Id (a string).
- DAItemDescriptor contains information necessary to identify an OPC item, such as its Item Id, together with additional information such as the data type requested.
- DAPropertyDescriptor contains information necessary to identify an OPC property, such as its Property Id (or a qualified name, in case of OPC XML).

There are following types of descriptor objects for OPC UA:

- UAEndpointDescriptor contains information necessary to identify and connect to an OPC server, such as the server's URL.
- UANodeDescriptor contains information necessary to identify an OPC node in the server, such as the UANodeId that holds the text of its expanded node Id.

Note: For OPC-UA, the meaning of these objects is described further below in chapter "Identifying Information in OPC-UA".

In a ServerDescriptor, the primary means of server identification is the ServerClass property, which contains a string that can hold either ProgID of the server, or its {CLSID}. There are also additional ServerDescriptor properties, all related to the ServerClass property, allowing to individually retrieve or modify parts of the server class designation. The ProgId property contains a ProgID of the server (and is an empty string if the ProgID is not given). The Clsid property contains the CLSID of the server in a form of a Guid (and is Guid.Empty if no CLSID for the server is given). The ClsidString property contains the CLSID string of the server (and is an empty string if no CLSID for the server is given). The ObjectId property contains the object ID, i.e. a ProgID, {CLSID}, or ProgID/{CLSID}.

Each ServerDescriptor is also represented in a string form that uses the URL syntax (e.g. "opcda://127.0.0.1/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397}"). For more information about these "OPC URLs", see http://www.opclabs.com/resources/developer-blog/1086-a-partial-story-of-opc-urls .

If you have received an element object (e.g. from browsing methods or common OPC dialogs), you can convert it to a descriptor object. In OPC Classic, for example, there is a constructor for ServerDescriptor object that accepts ServerElement as an input, and there is a constructor for DAItemDescriptor object that accepts DANodeElement as an input, too. In OPC UA, there is a constructor for UAEndpointDescriptor object that accepts UADiscoveryElement as an input, and there is a constructor for UANodeDescriptor object that accepts UANodeElement as an input, too.

Nodes and items in OPC address space can be specified using their node Id (or item Id), or using so-called browse paths.

The item Id (or node Id) is a string that uniquely addresses an item (or a node) in OPC address space. Its syntax is determined by the OPC server. It usually consists of the names of nodes used to reach that node from the root of the OPC address space, together with some decorating characters and separators. This is not a requirement, however, and the OPC server may even use item Ids that do not resemble the node names at all. Unless you have detailed knowledge of the particular OPC server you are connecting to, you should not attempt to create item Id strings yourself, from various parts – they should always come from the server. Item Ids (node Ids) are specified by the ItemId property of the DANodeDescriptor or DAItemDescriptor object. Strings convert implicitly to DANodeDescriptor or DAItemDescriptor containing the input string as the value of ItemId property, and you can therefore simply use the string item Ids in place of node and item descriptors.

4.6.3.7 Parameter Objects

Parameter objects are just holders for settings that influence certain aspect of how OPC Data Client works (for example, timeouts). There are several types of parameter objects (such as Timeouts, HoldPeriods, and more). For more information on their details, see the "Setting Parameters" and "Advanced Topics" chapters, and also the Reference documentation.

The parameters of all EasyXXClient objects are consistently organized into three groups:

- SharedParameters (static parameters that are always shared among object instances),
- InstanceParameters (parameters that can always be set independently for each object instance), and
- AdaptableParameters (parameters that are normally shared, but can be made specific to an instance if the Isolated property is set to 'true').

An instance of any EasyXXClient object that is not isolated (i.e. has its Isolated property equal to 'false', which is the default) actually uses following settings:

- Shared parameters from the static property EasyXXClient.SharedParameters,
- adaptable parameters from the static property EasyXXClient.AdaptableParameters, and
- instance parameters from its instance property EasyXXClient.InstanceParameters.

An isolated instance (i.e. an instance which has its Isolated property explicitly set to 'true') of any EasyXXClient object actually uses following settings:

- Shared parameters from the static property EasyXXClient.SharedParameters,
- adaptable parameters from its instance property EasyXXClient.IsolatedParameters, and
- instance parameters from its instance property EasyXXClient.InstanceParameters.

In OPC Data Client.NET, there are sometimes useful conversions and implicit conversions that allow you to easily construct parameter objects. For example, the DAGroupParameters object has a FromInt32 static method, and a corresponding implicit conversion operator, that allow it be constructed from an integer that represents the requested update rate (in milliseconds). This means that in C# and many other languages, you can simply use an integer update rate at all places where DAGroupParameters object is expected, if you are fine with specifying just the update rate and keeping the other properties of DAGroupParameters at their defaults.

All parameter and policy objects (and their constituents) consistently have a static Default property. This allows for a cleaner code where an arguments is needed, but you want to supply a default (there is no need to call the default constructor).

All parameter objects consistently derive from a Parameters base class. This class has a StandardName property (a string). The standard name identifies certain well-known combination of parameter values. When all the parameters correspond to a well-known combination, the StandardName property contains a symbolic name of such combination; otherwise, it contains an empty string. You can also set the StandardName property yourself, and the parameter values will change to reflect the name.

Some other objects that have similar nature also derive from the Parameters base class.

4.6.3.8 Utility Classes

The VarTypeUtilities static class provides a FromType method that allows you to determine the COM-based type VarType (used in OPC operations) that corresponds to a given .NET Type.

The DAUtilities static class provides methods that allow you to determine whether a given data type (VARIANT), percent deadband, update rate or value age are valid values in OPC Data Access.

The AEUtilities static class provides methods that allow you to determine whether given data type (VarType), event severity, event type filter, or notification rate are valid values in OPC Alarms&Events.

4.7 Connection-less Approach

OPC (Client-Server model) is inherently stateful. For starters, connections to OPC servers are long-living entities with rich internal state, and other objects in OPC model such as OPC groups have internal state too. OPC Data Client hides most of the OPC's stateful nature by providing a connection-less (stateless) interface for OPC tasks.

This transformation from a stateful to a stateless model is actually one of the biggest advantages you gain by incorporating OPC Data Client. There are several advantages to a stateless model from the perspective of your application. Here are the most important of them:

- The code you have to write is shorter. You do not have to make multiple method calls to get to the desired state first. Most common tasks can be achieved simply by instantiating an object (needed just once), and making a single method call.
- You do not have to worry about reconstructing all the state after some failure. OPC Data Client reconstructs the OPC state silently in background when needed. This again brings tremendous savings in coding.

The internal state of OPC Data Client components (including e.g. the connections to OPC servers) outlives the lifetime of the individual instances of the main EasyDAClient, EasyAEClient or EasyUAClient object. You can therefore create an instance of this object as many times as you wish, without incurring performance penalty to OPC communications. This characteristic comes extremely handy in programming server-side Web applications and similar scenarios: You can implement your code on page level, and make OPC requests from the page code itself. The OPC connections will outlive the page round-trips (if this was not the case, the OPC server would become bogged down quickly).

4.8 Simultaneous Operations

OPC (Cleint-Server model) works with potentially large quantities of relatively small data items that may change rapidly in time. In order to handle this kind of load effectively, it is necessary to operate on larger “chunks” of data whenever possible. When there is an operation to be performed on multiple elements, the elements should be passed to the operation together, and the results obtained together as well.

In order to ensure high efficiency, your code should allow the same. This is achieved by calling methods that are designed to work on multiple items in parallel. Where it makes sense, OPC Data Client provides such methods, and they contain the work **Multiple** in their names. For example, for reading a value of an OPC item, a method named ReadItemValue exists on the EasyDAClient object (or, ReadValue method on the EasyUAClient object). There is also a corresponding method named ReadMultipleItemValues on the EasyDAClient object (or, ReadMultipleValues on the EasyUAClient object) which can read multiple OPC items at once. It is strongly recommended that you call the methods that are designed for simultaneous operation wherever possible.

Methods for simultaneous operation return an array of OperationResult objects, or its derivatives. Each element in the output array corresponds to an element in the input array with the same index. Some methods or method overloads take multiple arguments, where some arguments are common for all elements, and one of them is the input array that has parts that are different for each element. There is always one method overload that takes a single argument which is an array of OperationArguments objects; this is the most generic method overload that allows each element be fully different from other elements.

4.9 Identifying information in OPC Classic

Nodes in OPC Classic are identified either by Item IDs, or using browse paths.

4.9.1 Browse Paths for OPC Classic

Browse Paths In General

An alternative way (to using a node ID) of specifying a node in OPC address space is using a browse path. The browse path is a sequence of node names, starting from a known location (e.g. the root of the OPC address space). The browse path can also be expressed as a string, where the individual node names are separated by delimiters (e.g. “/”, a slash).

Browse paths can be absolute (starting from a specified node), or relative.

The advantage of the browse paths is that you can construct them yourself, with just knowledge of the node names, without knowing the full node IDs. In the end, each item must be addressed by its node ID, and OPC Data Client will resolve the browse path to a node ID as necessary.

OPC Classic Browse Paths Specifics

In OPC Classic, absolute browse paths always start at the root of the OPC address space, because the root is the only "pre-defined" or well-known node.

An absolute browse path for OPC Classic is contained in a BrowsePath object. Browse paths are commonly specified by the BrowsePath property of the DANodeDescriptor or DAItemDescriptor object.

If a non-null ItemID is specified in the descriptor, OPC Data Client will use this item Id and ignore the browse path. If ItemID is null, OPC Data Client will attempt to resolve the browse path contained in the BrowsePath property of the descriptor. Either item id, or browse path (or both) must be specified.

Besides the ability to get around usage of full node IDs, the other reason to use browse paths is for OPC browsing with OPC Servers that only support OPC Data Access 1.0 specification. Such OPC servers cannot start the browsing at a node in the address given just by its Item ID; the node must always be reached by browsing from the root level. If the ID of such node it already known, OPC Data Client takes care of supplying the proper browse path automatically, but this cannot be always done. If, however, a browse path is given (which can be done by using the DANodeElement that is the output of the browsing to construct the DANodeDescriptor that is the input of further browsing), the browsing can proceed normally.

4.9.1.1 OPC Classic Browse Path Format

In OPC Classic, the slash at the beginning of the browse path denotes an absolute path that starts from the root of the OPC address space. The string form of a browse path is a concept on the client side (OPC Data Client), and does not appear in OPC specifications.

A browse path can be represented by a string; in such case, it can express either a relative or absolute browse path (the BrowsePath object cannot hold relative browse paths).

As mentioned above, the format of the browse path string is such that the individual node names are separated by slashes ("/"). The slash at the beginning of the browse path denotes an absolute path that starts from the root of the OPC address space. In addition, '.' denotes a current level, and '..' denotes a parent level, similarly to the conventions used in Windows file system.

Because the node names can contain any characters, we need to consider situations in which the node name itself contains a slash ('/') or a dot ('.'). In such case, in the string format of browse paths, these characters are escaped by preceding them with an ampersand ('&'). An ampersand itself needs to be escaped, too.

4.10 Identifying information in OPC XML

OPC Data Client.NET supports the OPC XML-DA specification transparently, with the same objects that are used for COM-based OPC Data Access. The two specifications are quite similar, and for most part, developers will not see much difference. Accesses to OPC-DA and OPC XML-DA can be freely mixed, even in the same method call. This transparency has been achieved by several generalizations in the object model and API.

OPC XML-DA specification identifies certain entities differently. The specifics of those are explained further below.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.10.1 OPC XML Servers

In OPC XML, a client connects to a server with the use of the server URL. OPC Data Client uses the ServerDescriptor class for identifying servers, and you can construct a ServerDescriptor by passing it a string that contains the server URL. The URL is also accessible in the UrlString property of the ServerDescriptor, and there is an implicit conversion available from a (URL) string to a ServerDescriptor. This means that at most places in OPC Data Client API, you can specify an OPC XML server simply by using its URL.

URLs of OPC XML servers can start with "http:", "https:", or "opc.xmlda" (which is internally treated as "http:").

For compatibility between OPC COM and OPC XML, the ServerDescriptor (which, for COM, contained mainly the machine name and server class in the past) is generalized to be able to describe both COM and XML based servers. The primary information in the ServerDescriptor is its UrlString. For OPC COM servers, the URL is composed in such a way that it contains the original Location (i.e. machine name) and ServerClass properties. For OPC XML servers, their URL can be used as the UrlString directly.

Note: We use the term "location" when we need to describe either a "machine name" (used in COM) or a "host name" (used with the Web, and OPC XML).

The ServerDescriptor also contains an additional NetworkSecurity property. This object can optionally specify the network credentials used when connecting to an OPC XML server.

The ServerElement (returned when you browse for OPC servers) has a UrlString property that can be used to connect to OPC XML servers.

4.10.2 OPC XML Nodes and Items

OPC XML-DA does not identify OPC items by a single string as OPC COM. Instead, it uses an additional string, and only the combination of the two identifies an item in an OPC XML-DA server. The original string that roughly corresponds to OPC item ID is called "ItemName" in the OPC XML-DA specification; in OPC Data Client, we use the existing ItemId property for it. The additional string is called "ItemPath" in the OPC XML-DA specification; in OPC Data Client, we have introduced a NodePath property for it. This property appears both in the DANodeDescriptor and DANodeElement. When you use the information received from the OPC XML browsing, both ItemId and NodePath are filled in the DANodeElement, and you can easily convert it to a DANodeDescriptor and it will "just work". If you are getting the nodes (items) from elsewhere, you need to create a DANodeDescriptor that contains both these strings (at least in general case; some OPC XML-DA server do not use them both).

4.10.3 OPC XML Properties

OPC XML-DA identifies properties by a string (a XML qualified name), instead of the numerical code used in OPC COM. For this a reason, a DAPropertyDescriptor and a DAPropertyElement have an additional QualifiedName property. When the component returns the DAPropertyElement from browsing, it fill is the information it knows about. When you pass the DAPropertyDescriptor to the component, you can fill in either PropertyId, or QualifiedName, or both. The component will use whatever is available and at the same time usable by the underlying technology. For the most common case, standard

(well-known) properties can always be identified using their numerical ID, and the component will look up their qualified name automatically.

The DAPropertyElement (returned when you browse for OPC properties) contains an additional ItemPath property for OPC XML. For properties coming from OPC XML servers, if the property can be accessed as an item as well, the property is filled in with the string that together with the ItemId identifies the OPC item that corresponds to the OPC property.

4.11 Identifying Information in OPC UA (Client-Server)

These chapters explain how the most commonly used entities in OPC Unified Architecture, such as servers, nodes, and attributes, are identified in the application code.

Note: Some entities in OPC Unified Architecture are identified by URIs and URLs. .NET Framework has a System.Uri class that provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI. It is generally recommended that programs use this class for URI manipulation internally, instead of working with plain strings containing URIs. In OPC Unified Architecture, however, the URIs and URLs are used in such a way that they do not always have to fully conform to the specifications. For this reason, URI and URL arguments and properties in OPC Data Client-UA are also strings and not System.Uri (and are consistently named so they contain "**UriString**" or "**UrlString**").

4.11.1 OPC UA Server Endpoints

A Server Endpoint is a physical address available on a network that allows clients to access one or more services provided by a server. Server endpoint is specified by its URL string.

Besides standard URLs e.g. with HTTP scheme, OPC Unified Architecture uses its own schemes, such as "opc.tcp". Examples of server endpoint URL strings are:

- http://opcua.demo-this.com:51211/UA/SampleServer
- opc.tcp://opcua.demo-this.com:51210/UA/SampleServer
- http://localhost:51211/UA/SampleServer
- opc.tcp://localhost:51210/UA/SampleServer

Note: For backward compatibility, it is also possible to use "opc.http:" in place of "http:" scheme in OPC UA endpoint descriptors (for OPC Unified Architecture over SOAP/HTTP). You should not, however, use "opc.http:" in new projects.

In OPC Data Client-UA, server endpoint corresponds to a UAEndpointDescriptor object that you create and pass to various methods. An implicit conversion from a string containing the URL of the endpoint exists. It is therefore common to specify UAEndpointDescriptor-s simply as strings, relying on the implicit conversion.

Endpoint Selection Policy

A single OPC-UA server may have multiple endpoints with the same functionality, but using different protocols, or different security settings, and expose the list of its own endpoint using discovery services. By default, the component will attempt to interpret the server endpoint you have specified as server's discovery endpoint, and automatically select the best endpoint.

How precisely the endpoint is selected is determined by Endpoint Selection Policy (UAEndpointSelectionPolicy object) that you can parameterize in the SessionParameters property on the EasyUAClient object. Such parameterization has a global effect. You can also specify the endpoint selection policy individually for just a specific endpoint descriptor. To do so, set the EndpointSelectionPolicy Property of the UAEndpointDescriptor to a non-null instance of UAEndpointSelectionPolicy. For details, see **OPC UA Endpoint Selection Policy (Section 4.11.1.1)**.

User Identity

If a user authentication is required by the OPC UA server, you can specify user identity on the endpoint, in the UAEndpointDescriptor object (alternatively, it can be specified in session parameters - the token infos from these two places are effectively merged together). See **User Identity in QuickOPC-UA (Section 5.1.2.6.1)** for more detailed discussion of this topic.

In order to make specifying the user identity with an endpoint descriptor easy OPC Data Client has extension methods on the UAEndpointDescriptor class. They are:

- WithAnonymousIdentity Method
- WithKerberosIdentity Method
- WithUserNameIdentity Method
- WithX509CertificateIdentity Method

These methods return a copy of the endpoint descriptor with the specified token info in its user identity.

4.11.1.1 OPC UA Endpoint Selection Policy

The OPC UA Endpoint Selection Policy allows you to specify the criteria that OPC Data Client will use when picking up the endpoint from those made available by the target OPC UA server.

In This Topic

Purpose

Why do we need the endpoint selection policy, with so many settings it has? Wouldn't it be better to specify the endpoint precisely, the same way as many other products do?

The answer to this is that in some cases, specifying the endpoint precisely may be possible and suitable. This is when you do know the endpoint parameters upfront, and you are sure that it will not change in the future. OPC Data Client actually supports this as well (using the very same endpoint selection policy mechanism, and specifying precise parameters).

In most cases, however, the aim of the client applications written with OPC Data Client is to be flexible, and be able to connect to the target server not just now, but even in future, when the details of the communication may change. This can happen easily, for example, when certain encryption algorithm becomes outdated, and the server is updated to no longer support it (or to give it less preference). If you had hard-coded the precise endpoint parameters into the client application, the solution will stop working at this point, and the client application will have to be rebuilt (or reconfigured, if you had allowed for that). With endpoint selection policy properly specified to indicate your intent, the client application written with OPC Data Client will automatically select the endpoint that uses the newer encryption algorithm, without anybody having to rewrite or reconfigure it.

If you use the endpoint selection policy just to express what you want to achieve, instead of picking the precise parameters which may change in future, you make your application more flexible and robust. Doing it this way is in line

with the declarative approach used wherever possible in OPC Data Client - that is, you express what you want to achieve, and leave the how of actually doing it to the component.

Effective Endpoint Selection Policy

The endpoint selection policy which is actually used for a concrete connection is called *effective endpoint selection policy*. It is determined in following steps:

1. If the EndpointSelectionPolicy Property on the UAEndpointDescriptor is set to a non-null value, this policy will be used.
2. Otherwise, if the Isolated Property on the EasyUAClient is true, the policy from EasyUAClient.IsolatedParameters.SessionParameters.EndpointSelectionPolicy will be used.
3. Otherwise (if the Isolated Property on the EasyUAClient is false, which is the default), the policy from static EasyUAClient.AdaptableParameters.SessionParameters.EndpointSelectionPolicy will be used.

Endpoint Selection Algorithm

When OPC Data Client selects an endpoint, it first decides whether each of the endpoints provided by the server is eligible for selection, using the effective endpoint selection policy. Settings in the Filter category (see further down) can influence this process.

The endpoint is then selected from those that have passed the filtering. What constitutes the "best" endpoint is determined by settings in the Order category (see further down).

Finally, some of the selected endpoint parameters may be modified, mainly for security reasons. This part is influenced by the settings in the Transform category (see further down).

Settings Category: Filter

Settings in this category influence which endpoints are eligible for selection. You can specify a subset of settings, and depending on how you do it, the filter may allow multiple endpoints, or just one. In some cases, the settings may conflict, and the policy would then not be able to select any endpoint ever.

Precise endpoint selection

You can specify a precise endpoint by setting following three properties of the endpoint selection policy:

- AllowedMessageSecurityModes Property
- SecurityPolicyUriString Property
- TransportProfileUriString Property

More details to the values of these properties are given below.

The AllowedMessageSecurityModes property of the UAEndpointSelectionPolicy gives you a possibility to completely enable or disable certain classes of endpoints from selection, based on their security mode. You can combine the value of this property from following flags:

- SecurityNone: No security is applied.
- SecuritySign: All messages are signed but not encrypted.
- SecuritySignAndEncrypt: All messages are signed and encrypted.

The UAEndpointSelectionPolicy.SecurityPolicyUriString property, when not an empty string (empty string is the default), specifies the required security policy URI string of the endpoint. This allows the developer to select the precise security policy, when so required. The static UASecurityPolicyUriStrings class contains strings that can be used to specify the security policy URI.

The TransportProfileUriString Property (when not an empty string) specifies the required transport profile of the endpoint. The static UATransportProfileUriStrings Class contains strings that can be used to specify the transport profile URI.

Imprecise endpoint selection

The endpoint does not have to be specified precisely. You can leave one or more of the settings described under "Precise endpoint selection" above - for example, keep the SecurityPolicyUriString Property and/or the TransportProfileUriString Property empty. Or, you can use combinations of multiple flags in the AllowedMessageSecurityModes Property. There are also pre-defined symbolic values for combinations of these flags, such as All (which is the default setting), or Secure.

If you want to specify some aspects of the transport profile, but not the precise transport profile as such, you can use:

ProtocolName Property: When not empty, requires the endpoint to use the specified protocol. Valid values are e.g. "HTTP", "HTTPS" or "TCP".

DataEncodingName Property: When not empty, requires that the endpoint uses the specified data encoding. Valid values are e.g. "Binary" and "XML".

Additional properties that influence the endpoint selection are:

- AllowZeroSecurityLevel Property. Endpoints that the server marks with a zero security level are not recommended, and are only made available for backward compatibility. This property determines whether such endpoints will be allowed.
- RejectObsoleteSecurityPolicies Property. Some security policies are already known to be obsolete. This property determines whether endpoints that use an obsolete security policy will be rejected.

The mechanism described here allows you to describe the intended capabilities of the endpoint, but without specifying its configuration precisely. This is the recommended approach, because it has a built-in flexibility to cover changes in server configuration or version, protocol improvements etc. An alternative approach (described above) allows more precision, but does not adapt so well to changes in server configuration.

Requiring security objectives

Following the declarative approach, you may decide to simply state which security objectives you want to achieve, using following properties:

- RequireAuthentication Property. Specifies whether authentication of the communication parties is required.
- RequireConfidentiality Property. Specifies whether communication confidentiality is required (usually provided by encryption).
- RequireIntegrity Property. Specifies whether communication integrity is required (usually provded by signing).

The communication parameters are considered as a whole. For example, even when the OPC UA message security does not provide encryption, the encryption may be provided by the transport. This is the case e.g. with the transports based on the HTTPS protocol.

Requiring certain security objectives may be combined with other settings in the policy.

Settings Category: Order

When more than one endpoint passes the filtering, settings in the Order category influence which of these endpoints will get selected. The endpoint are given a numerical ranking, and the endpoint will the highest ranking will become the winner. The basis for the ranking is the security level of each endpoint, as indicated by the server itself. The security level can then optionally be further manipulated.

The MessageSecurityPreference property of the UAEndpointSelectionPolicy tells the component whether endpoints that provide message security are preferred for selection. It has three possible values:

- Negative: The endpoints without message security will be given a major preference. This is the default setting, for good interoperability.
- None: The endpoint security level provided by the server will be followed without change.
- Positive: The endpoints with message security will be given a major preference.

Settings Category: Transform

Settings in this category influence how the endpoint parameters are modified after the endpoint has been selected. The available settings are:

- EnforceSamePort Property: When enabled, replaces the port number in the selected endpoint by the value used for discovery.
- EnforceSameSite Property: When enabled, replaces the host name in the selected endpoint by the value used for discovery.

The modifications are done mainly for security reasons.

Predefined Endpoint Selection Policies

Instead of setting the individual parameters of the policy, you can also use one of the predefined policies provided by the component. They are made available by static properties on the UAEndpointSelectionPolicy Class. They are:

- NoMessageSecurity Property. Endpoint selection policy that allows only connections with no message security.
- Default Property. Default endpoint selection policy. Allows all kinds of connections, and prefers connections with no message security.
- GuaranteedIntegrity Property. Endpoint selection policy that guarantees communication integrity.
- GuaranteedIntegrityExcludingObsolete Property. Endpoint selection policy that guarantees integrity, excluding obsolete security policies.
- GuaranteedIntegrityAndConfidentiality Property. Endpoint selection policy that guarantees integrity and confidentiality, excluding obsolete security policies.
- FullySecured Property. An endpoint selection policy that guarantees integrity, confidentiality and authentication, with additional safeguards.

Alternatively, parameters of the policy object can be changed to one these predefined policies by settings its StandardName Property to one of the names listed above. This is useful e.g. on the COM platform, where static properties do not exist.

4.11.1.2 Preselected vs. Synthetised Endpoints

> ⚠ This is an advanced topic. You do not need to study it unless you have a specific need.

In This Topic

When OPC Data Client establishes a connection to an OPC UA server, it tries to do so in most effective manner. When necessary, it first calls the GetEndpoints service of the target OPC UA server, and *preselects* the endpoint. In other cases, the call to GetEndpoints can be skipped, and the endpoint is *synthetised* by OPC Data Client.

The endpoint demands preselection if at least of one the conditions below is fulfilled:

- Your target development platform is .NET Standard.
- The AllowedMessageSecurityModes Property in the effective endpoint policy does not specify a precise (single) message security mode.
- The SecurityPolicyUriString Property is empty (i.e. does not specify a precise security policy URI).
- The AllowedMessageSecurityModes is not equal to SecurityNone, or the SecurityPolicyUriString is not equal to None ("http://opcfoundation.org/UA/SecurityPolicy#None"), and at the same time, the server certificate is not given in the ServerCertificate Property (or ServerCertificateByteArray Property) of the UAEndpointDescriptor.

This might seem quite complicated, but it boils down to a simple rule: If OPC Data Client has information available that is precise enough to allow connection without calling GetEndpoints first, it will connect straight away, if supported by the underlying OPC UA stack (= endpoint synthesis). Otherwise, it will call GetEndpoints in order to obtain the list of available endpoints and select from them (= endpoint preselection).

In addition, the endpoint will be preselected if you set the AlwaysPreselectEndpoint Property to true.

Preselected Endpoints

Endpoint preselection is performed by calling the OPC UA GetEndpoints service on the target server, and applying the effective endpoint selection policy to it. The OPC UA server must therefore properly support the GetEndpoints service for endpoint preselection to work. When applying the endpoint selection policy yields an endpoint, the parameters of it are then used to create the OPC UA secure channel.

Synthetised Endpoints

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

Synthetised endpoints do not require use of the OPC UA GetEndpoints service as part of the connection process. They are therefore a bit faster to connect, and (more importantly) allow connections even to servers that do not support the GetEndpoints service (or do not support it properly). The client code in OPC Data Client simply uses the parameters from the effective endpoint selection policy when creating the OPC UA secure channel. The effective endpoint selection policy is then applied to the endpoints returned by the OPC UA server from the CreateSession service, and OPC Data Client checks whether the endpoint specified by the policy is available.

> ⚠ Some servers may have endpoint URLs that are different from the discovery endpoint URL, or differ depending on the security policy. For example, a server built with OPC Foundation .NET legacy stack and using the HTTP protocol will use an endpoint URL with appended "/None" for its insecure endpoint. You need to specify the precise URL of the specific endpoint, for the synthesized endpoint to work.

4.11.1.3 Server Certificate in Endpoint Descriptor

> ⚠ This is an advanced topic. You do not need to study it unless you have a specific need.

If you want the endpoint be synthetised and not preselected (see **Preselected vs. Synthetised Endpoints (Section 4.11.1.2)**) and you want the communication be secure, you must also present a server certificate. This is done using the ServerCertificate Property (or the ServerCertificateByteArray Property) in the UAEndpointDescriptor.

4.11.2 OPC UA Node IDs

Node ID is an identifier for a node in an OPC server's address space.

OPC Unified Architecture allows the OPC server to choose one or more types of node IDs for representation of its nodes. Node IDs can be numeric (a 32-bit integer), string, a GUID (globally unique identifier, 128 bits), or opaque (a binary data blob).

OPC Data Client-UA works with so-called expanded node IDs, which contain the node's identifier together with a complete namespace URI. A single server may contain nodes in multiple namespaces, and the server maintains a namespace table where each namespace can be identified by an index. During a communication session with an OPC client, nodes are identified using indices into the namespace table. The namespace table contents and order of elements may, however, change between sessions.

OPC Data Client-UA does not expose node IDs with indices into a namespace table alone. Instead, it always passes around the complete URI string of a namespace with a node ID. Because this expanded form of a node ID is used, a Node ID in OPC Data Client-UA is stable between sessions, and can also be persistent (stored) and later used without a complexity of additional index remapping.

Depending on the type of Node ID, the expanded node ID string may have one of the following forms:

- nsu=*namespaceUri*;i=*integer*
- nsu=*namespaceUri*;s=*string*
- nsu=*namespaceUri*;g=*guid*
- nsu=*namespaceUri*;b=*base64string*

In OPC Data Client-UA, an expanded node ID corresponds to UANodeId object that you create and pass to various methods. An implicit conversion from a string containing the expanded text of Node ID exists. It is therefore common to specify UANodeId-s simply as strings, relying on the implicit conversion. Also note that UANodeId is usually a part of UANodeDescriptor and can be (implicitly) converted to it easily.

When the UANodeId object contains no data, it is considered a null Node Id, and its IsNull property is true. Beware that this is different from having a null reference to a UANodeId (which, in most places, is not allowed).

> In OPC Data Client, when formatting the expanded text of OPC UA Node IDs and qualified names, the namespace URI (the text after "nsu=") is separated with an additional space from the following semicolon, allowing the URI be easily separated by tools that are not aware of the specific syntax of OPC UA Node IDs and qualified names. Similarly, any trailing whitespace in the namespace URI part (the text after "nsu=") is ignored when parsing the OPC UA Node IDs and qualified names.

Note: During browsing, the OPC server provides so-called browse names (browse IDs), which can be combined into browse paths. The browse names (browse IDs) usually "look" more user- friendly and you may be tempted to think that they would be more appropriate for node identification. The purpose of browse names (browse IDs) is, however, different,

and using them alone for node identification would be inefficient, and sometimes ambiguous.

There are many ways in which you can construct node IDs, depending on which information you have available.

Following examples are intended to show you many of the options that exist for constructing the node IDs.

C#

```
// This example shows different ways of constructing OPC UA node IDs.

using System;
using OpcLabs.BaseLib;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Parsing;
using OpcLabs.EasyOpc.UA.AddressSpace.Parsing.Extensions;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UANodeId
{
    class _Construction
    {
        // A node ID specifies a namespace (either by an URI or by an index), and an
identifier.
        // The identifier can be numeric (an integer), string, GUID, or opaque.
        public static void Main1()
        {

            // A node ID can be specified in string form (so-called expanded text).
            // The code below specifies a namespace URI (nsu=...), and an integer
identifier (i=...).
            UANodeId nodeId1 = new UANodeId("nsu=http://test.org/UA/Data/;i=10853");
            Console.WriteLine(nodeId1);


            // Similarly, with a string identifier (s=...).
            UANodeId nodeId2 = new
UANodeId("nsu=http://test.org/UA/Data/;s=someIdentifier");
            Console.WriteLine(nodeId2);


            // Actually, "s=" can be omitted (not recommended, though)
            UANodeId nodeId3 = new
UANodeId("nsu=http://test.org/UA/Data/;someIdentifier");
            Console.WriteLine(nodeId3);
            // Notice that the output is normalized - the "s=" is added again.


            // Similarly, with a GUID identifier (g=...)
            UANodeId nodeId4 = new UANodeId("nsu=http://test.org/UA/Data/;g=BAEAF004-
1E43-4A06-9EF0-E52010D5CD10");
            Console.WriteLine(nodeId4);
            // Notice that the output is normalized - uppercase letters in the GUI are
converted to lowercase, etc.
```

```
            // Similarly, with an opaque identifier (b=..., in Base64 encoding).
            UANodeId nodeId5 = new UANodeId("nsu=http://test.org/UA/Data/;b=AP8=");
            Console.WriteLine(nodeId5);


            // Namespace index can be used instead of namespace URI. The server is
allowed to change the namespace
            // indices between sessions (except for namespace 0), and for this reason,
you should avoid the use of
            // namespace indices, and rather use the namespace URIs whenever possible.
            UANodeId nodeId6 = new UANodeId("ns=2;i=10853");
            Console.WriteLine(nodeId6);


            // Namespace index can be also specified together with namespace URI. This
is still safe, but may be
            // a bit quicker to perform, because the client can just verify the
namespace URI instead of looking
            // it up.
            UANodeId nodeId7 = new
UANodeId("nsu=http://test.org/UA/Data/;ns=2;i=10853");
            Console.WriteLine(nodeId7);


            // When neither namespace URI nor namespace index are given, the node ID is
assumed to be in namespace
            // with index 0 and URI "http://opcfoundation.org/UA/", which is reserved
by OPC UA standard. There are
            // many standard nodes that live in this reserved namespace, but no nodes
specific to your servers will
            // be in the reserved namespace, and hence the need to specify the
namespace with server-specific nodes.
            UANodeId nodeId8 = new UANodeId("i=2254");
            Console.WriteLine(nodeId8);


            // If you attempt to pass in a string that does not conform to the syntax
rules,
            // a UANodeIdFormatException is thrown.
            try
            {
                UANodeId nodeId9 = new
UANodeId("nsu=http://test.org/UA/Data/;i=notAnInteger");
                Console.WriteLine(nodeId9);
            }
            catch (UANodeIdFormatException nodeIdFormatException)
            {
                Console.WriteLine($"*** Failure {nodeIdFormatException.Message}");
            }


            // There is a parser object that can be used to parse the expanded texts of
node IDs.
            UANodeIdParser nodeIdParser10 = new UANodeIdParser();
            UANodeId nodeId10 =
nodeIdParser10.Parse("nsu=http://test.org/UA/Data/;i=10853");
            Console.WriteLine(nodeId10);
```

```
            // The parser can be used if you want to parse the expanded text of the
node ID but do not want
            // exceptions be thrown.
            UANodeIdParser nodeIdParser11 = new UANodeIdParser();
            IStringParsingError stringParsingError =
                nodeIdParser11.TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger",
out UANodeId nodeId11);
            if (stringParsingError == null)
                Console.WriteLine(nodeId11);
            else
                Console.WriteLine($"*** Failure: {stringParsingError.Message}");


            // You can also use the parser if you have node IDs where you want the
default namespace be different
            // from the standard "http://opcfoundation.org/UA/".
            UANodeIdParser nodeIdParser12 = new
UANodeIdParser("http://test.org/UA/Data/");
            UANodeId nodeId12 = nodeIdParser12.Parse("i=10853");
            Console.WriteLine(nodeId12);


            // The namespace URI string (or the namespace index, or both) and the
identifier can be passed to the
            // constructor separately.
            UANodeId nodeId13 = new UANodeId("http://test.org/UA/Data/", 10853);
            Console.WriteLine(nodeId13);


            // You can create a "null" node ID. Such node ID does not actually identify
any valid node in OPC UA, but
            // is useful as a placeholder or as a starting point for further
modifications of its properties.
            UANodeId nodeId14 = new UANodeId();
            Console.WriteLine(nodeId14);


            // Properties of a node ID can be modified individually. The advantage of
this approach is that you do
            // not have to care about syntax of the node ID expanded text.
            UANodeId nodeId15 = new UANodeId();
            nodeId15.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId15.Identifier = 10853;
            Console.WriteLine(nodeId15);


            // The same as above, but using an object initializer list.
            UANodeId nodeId16 = new UANodeId
            {
                NamespaceUriString = "http://test.org/UA/Data/",
                Identifier = 10853
            };
            Console.WriteLine(nodeId16);
```

```
            // If you know the type of the identifier upfront, it is safer to use typed
properties that correspond
            // to specific types of identifier. Here, with an integer identifier.
            UANodeId nodeId17 = new UANodeId();
            nodeId17.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId17.NumericIdentifier = 10853;
            Console.WriteLine(nodeId17);


            // Similarly, with a string identifier.
            UANodeId nodeId18 = new UANodeId();
            nodeId18.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId18.StringIdentifier = "someIdentifier";
            Console.WriteLine(nodeId18);


            // Similarly, with a GUID identifier.
            UANodeId nodeId19 = new UANodeId();
            nodeId19.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId19.GuidIdentifier = Guid.Parse("BAEAF004-1E43-4A06-9EF0-
E52010D5CD10");
            Console.WriteLine(nodeId19);


            // If you have GUID in its string form, the node ID object can parse it for
you.
            UANodeId nodeId20 = new UANodeId();
            nodeId20.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId20.GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10";
            Console.WriteLine(nodeId20);


            // And, with an opaque identifier.
            UANodeId nodeId21 = new UANodeId();
            nodeId21.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId21.OpaqueIdentifier = new byte[] {0x00, 0xFF};
            Console.WriteLine(nodeId21);


            // Assigning an expanded text to a node ID parses the value being assigned
and sets all corresponding
            // properties accordingly.
            UANodeId nodeId22 = new UANodeId();
            nodeId22.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";
            Console.WriteLine(nodeId22);


            // There is an implicit conversion from a string (representing the expanded
text) to a node ID.
            // You can therefore use the expanded text (string) in place of any node ID
object directly.
            UANodeId nodeId23 = "nsu=http://test.org/UA/Data/;i=10853";
            Console.WriteLine(nodeId23);


            // There is a copy constructor as well, creating a clone of an existing
node ID.
```

```
            UANodeId nodeId24a = new UANodeId("nsu=http://test.org/UA/Data/;i=10853");
            Console.WriteLine(nodeId24a);
            UANodeId nodeId24b = new UANodeId(nodeId24a);
            Console.WriteLine(nodeId24b);


            // We have provided static classes with properties that correspond to all
standard nodes specified by
            // OPC UA. You can simply refer to these node IDs in your code.
            // The class names are UADataTypeIds, UAMethodIds, UAObjectIds,
UAObjectTypeIds, UAReferenceTypeIds,
            // UAVariableIds and UAVariableTypeIds.
            UANodeId nodeId25 = UAObjectIds.TypesFolder;
            Console.WriteLine(nodeId25);
            // When the UANodeId equals to one of the standard nodes, it is output in
the shortened form - as the standard
            // name only.


            // You can also refer to any standard node using its name (in a string
form).
            // Note that assigning a non-existing standard name is not allowed, and
throws ArgumentException.
            UANodeId nodeId26 = new UANodeId();
            nodeId26.StandardName = "TypesFolder";
            Console.WriteLine(nodeId26);


            // When you browse for nodes in the OPC UA server, every returned node
element contains a node ID that
            // you can use further.
            var client27 = new EasyUAClient();
            try
            {
                UANodeElementCollection nodeElementCollection27 = client27.Browse(
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
                if (nodeElementCollection27.Count != 0)
                {
                    UANodeId nodeId27 = nodeElementCollection27[0].NodeId;
                    Console.WriteLine(nodeId27);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("Failure: {0}",
uaException.GetBaseException().Message);
            }


            // As above, but using a constructor that takes a node element as an input.
            var client28 = new EasyUAClient();
            try
            {
                UANodeElementCollection nodeElementCollection28 = client28.Browse(
```

```
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
                if (nodeElementCollection28.Count != 0)
                {
                    UANodeId nodeId28 = new UANodeId(nodeElementCollection28[0]);
                    Console.WriteLine(nodeId28);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("Failure: {0}",
uaException.GetBaseException().Message);
            }


            // Or, there is an explicit conversion from a node element as well.
            var client29 = new EasyUAClient();
            try
            {
                UANodeElementCollection nodeElementCollection29 = client29.Browse(
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
                if (nodeElementCollection29.Count != 0)
                {
                    UANodeId nodeId29 = (UANodeId) nodeElementCollection29[0];
                    Console.WriteLine(nodeId29);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

Object Pascal

```
// This example shows different ways of constructing OPC UA node IDs.

class procedure _Construction.Main;
var
  BrowseParameters: _UABrowseParameters;
  Client27: _EasyUAClient;
  EndpointDescriptor: _UAEndpointDescriptor;
  NodeId1, NodeId2, NodeId3, NodeId4, NodeId5, NodeId6, NodeId7, NodeId8, NodeId9,
NodeId10,
  NodeId11, NodeId12, NodeId14, NodeId15, NodeId17, NodeId18,
  NodeId20, NodeId21, NodeId26, NodeId27,
  ServerNodeId, ReferencesNodeId: OpcLabs_EasyOpcUA_TLB._UANodeId;
  NodeId11Result: OleVariant;
  NodeIdParser10, NodeIdParser11, NodeIdParser12: _UANodeIdParser;
```

```
  NodeElements27: _UANodeElementCollection;
  OpaqueIdentifier21: Variant;
  ServerNodeDescriptor: _UANodeDescriptor;
  StringParsingError: _StringParsingError;
begin
  // A node ID specifies a namespace (either by an URI or by an index), and an
identifier.
  // The identifier can be numeric (an integer), string, GUID, or opaque.

  // A node ID can be specified in string form (so-called expanded text).
  // The code below specifies a namespace URI (nsu=...), and an integer identifier
(i=...).
  // Assigning an expanded text to a node ID parses the value being assigned and sets
all corresponding
  // properties accordingly.
  NodeId1 := CoUANodeId.Create;
  NodeId1.ExpandedText := 'nsu=http://test.org/UA/Data/;i=10853';
  WriteLn(NodeId1.ToString);


  // Similarly, with a string identifier (s=...).
  NodeId2 := CoUANodeId.Create;
  NodeId2.ExpandedText := 'nsu=http://test.org/UA/Data/;s=someIdentifier';
  WriteLn(NodeId2.ToString);


  // Actually, "s=" can be omitted (not recommended, though)
  NodeId3 := CoUANodeId.Create;
  NodeId3.ExpandedText := 'nsu=http://test.org/UA/Data/;someIdentifier';
  WriteLn(NodeId3.ToString);
  // Notice that the output is normalized - the "s=" is added again.


  // Similarly, with a GUID identifier (g=...)
  NodeId4 := CoUANodeId.Create;
  NodeId4.ExpandedText := 'nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-4A06-9EF0-
E52010D5CD10';
  WriteLn(NodeId4.ToString);
  // Notice that the output is normalized - uppercase letters in the GUI are converted
to lowercase, etc.


  // Similarly, with an opaque identifier (b=..., in Base64 encoding).
  NodeId5 := CoUANodeId.Create;
  NodeId5.ExpandedText := 'nsu=http://test.org/UA/Data/;b=AP8=';
  WriteLn(NodeId5.ToString);


  // Namespace index can be used instead of namespace URI. The server is allowed to
change the namespace
  // indices between sessions (except for namespace 0), and for this reason, you should
avoid the use of
  // namespace indices, and rather use the namespace URIs whenever possible.
  NodeId6 := CoUANodeId.Create;
  NodeId6.ExpandedText := 'ns=2;i=10853';
  WriteLn(NodeId6.ToString);
```

```
  // Namespace index can be also specified together with namespace URI. This is still
safe, but may be
  // a bit quicker to perform, because the client can just verify the namespace URI
instead of looking
  // it up.
  NodeId7 := CoUANodeId.Create;
  NodeId7.ExpandedText := 'nsu=http://test.org/UA/Data/;ns=2;i=10853';
  WriteLn(NodeId7.ToString);


  // When neither namespace URI nor namespace index are given, the node ID is assumed
to be in namespace
  // with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by OPC UA
standard. There are
  // many standard nodes that live in this reserved namespace, but no nodes specific to
your servers will
  // be in the reserved namespace, and hence the need to specify the namespace with
server-specific nodes.
  NodeId8 := CoUANodeId.Create;
  NodeId8.ExpandedText := 'i=2254';
  WriteLn(NodeId8.ToString);


  // If you attempt to pass in a string that does not conform to the syntax rules,
  // a UANodeIdFormatException is thrown.
  NodeId9 := CoUANodeId.Create;
  try
    NodeId9.ExpandedText := 'nsu=http://test.org/UA/Data/;i=notAnInteger';
    WriteLn(NodeId9.ToString);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.BaseException.Message]));
    end;
  end;


  // There is a parser object that can be used to parse the expanded texts of node IDs.
  NodeIdParser10 := CoUANodeIdParser.Create;
  NodeId10 := NodeIdParser10.Parse('nsu=http://test.org/UA/Data/;i=10853', False);
  WriteLn(NodeId10.ToString);


  // The parser can be used if you want to parse the expanded text of the node ID but
do not want
  // exceptions be thrown.
  NodeIdParser11 := CoUANodeIdParser.Create;
  StringParsingError :=
NodeIdParser11.TryParse('nsu=http://test.org/UA/Data/;i=notAnInteger', False,
NodeId11Result);
  if StringParsingError = nil then
    begin
      NodeId11 := IUnknown(NodeId11Result) as OpcLabs_EasyOpcUA_TLB._UANodeId;
      WriteLn(NodeId11.ToString);
    end
  else
```

```
    WriteLn(Format('*** Failure: %s', [StringParsingError.Message]));


  // You can also use the parser if you have node IDs where you want the default
namespace be different
  // from the standard "http://opcfoundation.org/UA/".
  NodeIdParser12 := CoUANodeIdParser.Create;
  NodeIdParser12.DefaultNamespaceUriString := 'http://test.org/UA/Data/';
  NodeId12 := NodeIdParser12.Parse('i=10853', False);
  WriteLn(NodeId12.ToString);


  // You can create a "null" node ID. Such node ID does not actually identify any valid
node in OPC UA, but
  // is useful as a placeholder or as a starting point for further modifications of its
properties.
  NodeId14 := CoUANodeId.Create;
  WriteLn(NodeId14.ToString);


  // Properties of a node ID can be modified individually. The advantage of this
approach is that you do
  // not have to care about syntax of the node ID expanded text.
  NodeId15 := CoUANodeId.Create;
  NodeId15.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId15.Identifier := 10853;
  WriteLn(NodeId15.ToString);


  // If you know the type of the identifier upfront, it is safer to use typed
properties that correspond
  // to specific types of identifier. Here, with an integer identifier.
  NodeId17 := CoUANodeId.Create;
  NodeId17.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId17.NumericIdentifier := 10853;
  WriteLn(NodeId17.ToString);


  // Similarly, with a string identifier.
  NodeId18 := CoUANodeId.Create;
  NodeId18.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId18.StringIdentifier := 'someIdentifier';
  WriteLn(NodeId18.ToString);


  // If you have GUID in its string form, the node ID object can parse it for you.
  NodeId20 := CoUANodeId.Create;
  NodeId20.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId20.GuidIdentifierString := 'BAEAF004-1E43-4A06-9EF0-E52010D5CD10';
  WriteLn(NodeId20.ToString);


  // And, with an opaque identifier.
  NodeId21 := CoUANodeId.Create;
  NodeId21.NamespaceUriString := 'http://test.org/UA/Data/';
//  OpaqueIdentifier21 := VarArrayCreate ([0, 1], varByte);
  OpaqueIdentifier21 := VarArrayCreate ([0, 1], varVariant);
```

```
  OpaqueIdentifier21[0] := $00;
  OpaqueIdentifier21[1] := $FF;
  NodeId21.OpaqueIdentifier := PSafeArray (TVarData (OpaqueIdentifier21).VArray);
  WriteLn(NodeId21.ToString);


  // We have built-in a list of all standard nodes specified by OPC UA. You can simply
refer to these node IDs in your code.
  // You can refer to any standard node using its name (in a string form).
  // Note that assigning a non-existing standard name is not allowed, and throws
ArgumentException.
  NodeId26 := CoUANodeId.Create;
  NodeId26.StandardName := 'TypesFolder';
  WriteLn(NodeId26.ToString);
  // When the UANodeId equals to one of the standard nodes, it is output in the
shortened form - as the standard name only.


  // When you browse for nodes in the OPC UA server, every returned node element
contains a node ID that
  // you can use further.
  Client27 := CoEasyUAClient.Create;
  EndpointDescriptor := CoUAEndpointDescriptor.Create;
  EndpointDescriptor.UrlString := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // Browse from the Server node.
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  ServerNodeDescriptor := CoUANodeDescriptor.Create;
  ServerNodeDescriptor.NodeId := ServerNodeId;
  // Browse all References.
  ReferencesNodeId := CoUANodeId.Create;
  ReferencesNodeId.StandardName := 'References';

  BrowseParameters := CoUABrowseParameters.Create;
  BrowseParameters.NodeClasses := UANodeClass_All;  // this is the default, anyway
  BrowseParameters.ReferenceTypeIds.Add(ReferencesNodeId);

  try
    NodeElements27 := Client27.Browse(EndpointDescriptor, ServerNodeDescriptor,
BrowseParameters);
    if NodeElements27.Count <> 0 then
    begin
      NodeId27 := NodeElements27[0].NodeId;
      WriteLn(NodeId27.ToString);
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.BaseException.Message]));
    end;
  end;
end;
```

PHP

```
// This example shows different ways of constructing OPC UA node IDs.
```

```
// A node ID specifies a namespace (either by an URI or by an index), and an
identifier.
// The identifier can be numeric (an integer), string, GUID, or opaque.

const UANodeClass_All = 255;


// A node ID can be specified in string form (so-called expanded text).
// The code below specifies a namespace URI (nsu=...), and an integer identifier
(i=...).
// Assigning an expanded text to a node ID parses the value being assigned and sets all
corresponding
// properties accordingly.
$NodeId1 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId1->ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";
printf("%s\n", $NodeId1);


// Similarly, with a string identifier (s=...).
$NodeId2 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId2->ExpandedText = "nsu=http://test.org/UA/Data/;s=someIdentifier";
printf("%s\n", $NodeId2);


// Actually, "s=" can be omitted (not recommended, though)
$NodeId3 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId3->ExpandedText = "nsu=http://test.org/UA/Data/;someIdentifier";
printf("%s\n", $NodeId3);
// Notice that the output is normalized - the "s=" is added again.


// Similarly, with a GUID identifier (g=...)
$NodeId4 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId4->ExpandedText = "nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-4A06-9EF0-
E52010D5CD10";
printf("%s\n", $NodeId4);
// Notice that the output is normalized - uppercase letters in the GUI are converted to
lowercase, etc.


// Similarly, with an opaque identifier (b=..., in Base64 encoding).
$NodeId5 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId5->ExpandedText = "nsu=http://test.org/UA/Data/;b=AP8=";
printf("%s\n", $NodeId5);


// Namespace index can be used instead of namespace URI. The server is allowed to
change the namespace
// indices between sessions (except for namespace 0), and for this reason, you should
avoid the use of
// namespace indices, and rather use the namespace URIs whenever possible.
$NodeId6 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId6->ExpandedText = "ns=2;i=10853";
printf("%s\n", $NodeId6);


// Namespace index can be also specified together with namespace URI. This is still
```

```
safe, but may be
// a bit quicker to perform, because the client can just verify the namespace URI
instead of looking
// it up.
$NodeId7 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId7->ExpandedText = "nsu=http://test.org/UA/Data/;ns=2;i=10853";
printf("%s\n", $NodeId7);


// When neither namespace URI nor namespace index are given, the node ID is assumed to
be in namespace
// with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by OPC UA
standard. There are
// many standard nodes that live in this reserved namespace, but no nodes specific to
your servers will
// be in the reserved namespace, and hence the need to specify the namespace with
server-specific nodes.
$NodeId8 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId8->ExpandedText = "i=2254";
printf("%s\n", $NodeId8);


// If you attempt to pass in a string that does not conform to the syntax rules,
// a UANodeIdFormatException is thrown.
$NodeId9 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
try
{
    $NodeId9->ExpandedText = "nsu=http://test.org/UA/Data/;i=notAnInteger";
    printf("%s\n", $NodeId9);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}


// There is a parser object that can be used to parse the expanded texts of node IDs.
$NodeIdParser10 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser");
$NodeId10 = $NodeIdParser10->Parse("nsu=http://test.org/UA/Data/;i=10853", False);
printf("%s\n", $NodeId10);


// The parser can be used if you want to parse the expanded text of the node ID but do
not want
// exceptions be thrown.
$NodeId11 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeIdParser11 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser");
$StringParsingError = $NodeIdParser11-
>TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger", False, $NodeId11);
if (is_null($StringParsingError))
    printf("%s\n", $NodeId11);
else
    printf("*** Failure: %s\n", $StringParsingError->Message);


// You can also use the parser if you have node IDs where you want the default
namespace be different
```

```
// from the standard "http://opcfoundation.org/UA/".
$NodeIdParser12 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser");
$NodeIdParser12->DefaultNamespaceUriString = "http://test.org/UA/Data/";
$NodeId12 = $NodeIdParser12->Parse("i=10853", False);
printf("%s\n", $NodeId12);


// You can create a "null" node ID. Such node ID does not actually identify any valid
node in OPC UA, but
// is useful as a placeholder or as a starting point for further modifications of its
properties.
$NodeId14 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
printf("%s\n", $NodeId14);


// Properties of a node ID can be modified individually. The advantage of this approach
is that you do
// not have to care about syntax of the node ID expanded text.
$NodeId15 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId15->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId15->Identifier = 10853;
printf("%s\n", $NodeId15);


// If you know the type of the identifier upfront, it is safer to use typed properties
that correspond
// to specific types of identifier. Here, with an integer identifier.
$NodeId17 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId17->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId17->NumericIdentifier = 10853;
printf("%s\n", $NodeId17);


// Similarly, with a string identifier.
$NodeId18 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId18->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId18->StringIdentifier = "someIdentifier";
printf("%s\n", $NodeId18);


// If you have GUID in its string form, the node ID object can parse it for you.
$NodeId20 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId20->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId20->GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10";
printf("%s\n", $NodeId20);


// And, with an opaque identifier.
$NodeId21 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId21->NamespaceUriString = "http://test.org/UA/Data/";
$OpaqueIdentifier21[0] = 0x00;
$OpaqueIdentifier21[1] = 0xFF;
$NodeId21->OpaqueIdentifier = $OpaqueIdentifier21;
printf("%s\n", $NodeId21);


// We have built-in a list of all standard nodes specified by OPC UA. You can simply
```

```
refer to these node IDs in your code.
// You can refer to any standard node using its name (in a string form).
// Note that assigning a non-existing standard name is not allowed, and throws
ArgumentException.
$NodeId26 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId26->StandardName = "TypesFolder";
printf("%s\n", $NodeId26);
// When the UANodeId equals to one of the standard nodes, it is output in the shortened
form - as the standard name only.


// When you browse for nodes in the OPC UA server, every returned node element contains
a node ID that
// you can use further.
$Client27 = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// Browse from the Server node.
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";
$ServerNodeDescriptor = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
$ServerNodeDescriptor->NodeId = $ServerNodeId;
// Browse all References.
$ReferencesNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ReferencesNodeId->StandardName = "References";

$BrowseParameters = new COM("OpcLabs.EasyOpc.UA.UABrowseParameters");
$BrowseParameters->NodeClasses = UANodeClass_All;  // this is the default, anyway
$BrowseParameters->ReferenceTypeIds->Add($ReferencesNodeId);

try
{
    $NodeElements27 = $Client27->Browse($EndpointDescriptor, $ServerNodeDescriptor,
$BrowseParameters);
    if ($NodeElements27->Count != 0) {
        $NodeId27 = $NodeElements27[0]->NodeId;
        printf("%s\n", $NodeId27);
    }
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}
```

VB.NET

```
' This example shows different ways of constructing OPC UA node IDs.

Imports System
Imports OpcLabs.BaseLib
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Parsing
Imports OpcLabs.EasyOpc.UA.AddressSpace.Parsing.Extensions
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
```

```
Namespace UADocExamples._UANodeId
    Friend Class _Construction
        Public Shared Sub Main1()

            ' A node ID can be specified in string form (so-called expanded text).
            ' The code below specifies a namespace URI (nsu=...), and an integer
identifier (i=...).
            Dim nodeId1 = New UANodeId("nsu=http://test.org/UA/Data/;i=10853")
            Console.WriteLine(nodeId1)


            ' Similarly, with a string identifier (s=...).
            Dim nodeId2 = New UANodeId("nsu=http://test.org/UA/Data/;s=someIdentifier")
            Console.WriteLine(nodeId2)


            ' Actually, "s=" can be omitted (not recommended, though)
            Dim nodeId3 = New UANodeId("nsu=http://test.org/UA/Data/;someIdentifier")
            Console.WriteLine(nodeId3)


            ' Similarly, with a GUID identifier (g=...)
            Dim nodeId4 = New UANodeId("nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-
4A06-9EF0-E52010D5CD10")
            Console.WriteLine(nodeId4)


            ' Similarly, with an opaque identifier (b=..., in Base64 encoding).
            Dim nodeId5 = New UANodeId("nsu=http://test.org/UA/Data/;b=AP8=")
            Console.WriteLine(nodeId5)


            ' Namespace index can be used instead of namespace URI. The server is
allowed to change the namespace
            ' indices between sessions, and for this reason, you should avoid the use
of namespace indices, and
            ' rather use the namespace URIs whenever possible.
            Dim nodeId6 = New UANodeId("ns=2;i=10853")
            Console.WriteLine(nodeId6)


            ' Namespace index can be also specified together with namespace URI. This
is still safe, but may be
            ' a bit quicker to perform, because the client can just verify the
namespace URI instead of looking
            ' it up.
            Dim nodeId7 = New UANodeId("nsu=http://test.org/UA/Data/;ns=2;i=10853")
            Console.WriteLine(nodeId7)


            ' When neither namespace URI nor namespace index are given, the node ID is
assumed to be in namespace
            ' with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by
OPC UA standard. There are
            ' many standard nodes that live in this reserved namespace, but no nodes
specific to your servers will
            ' be in the reserved namespace, and hence the need to specify the namespace
```

```
with server-specific nodes.
            Dim nodeId8 = New UANodeId("i=2254")
            Console.WriteLine(nodeId8)


            ' If you attempt to pass in a string that does not conform to the syntax
rules,
            ' a UANodeIdFormatException is thrown.
            Try
                Dim nodeId9 = New
UANodeId("nsu=http://test.org/UA/Data/;i=notAnInteger")
                Console.WriteLine(nodeId9)
            Catch nodeIdFormatException As UANodeIdFormatException
                Console.WriteLine(nodeIdFormatException.Message)
            End Try


            ' There is a parser object that can be used to parse the expanded textx of
node IDs.
            Dim nodeIdParser10 = New UANodeIdParser()
            Dim nodeId10 = nodeIdParser10.Parse("nsu=http://test.org/UA/Data/;i=10853")
            Console.WriteLine(nodeId10)


            ' The parser can be used if you want to parse the expanded text of the node
ID but do not want
            ' exceptions be thrown.
            Dim nodeIdParser11 = New UANodeIdParser()
            Dim nodeId11 As UANodeId = Nothing
            Dim stringParsingError As IStringParsingError =

nodeIdParser11.TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger", nodeId11)
            If stringParsingError Is Nothing Then
                Console.WriteLine(nodeId11)
            Else
                Console.WriteLine(stringParsingError.Message)
            End If


            ' You can also use the parser if you have node IDs where you want the
default namespace be different
            ' from the standard "http://opcfoundation.org/UA/".
            Dim nodeIdParser12 = New UANodeIdParser("http://test.org/UA/Data/")
            Dim nodeId12 = nodeIdParser12.Parse("i=10853")
            Console.WriteLine(nodeId12)


            ' The namespace URI string (or the namespace index, or both) and the
identifier can be passed to the
            ' constructor separately.
            Dim nodeId13 = New UANodeId("http://test.org/UA/Data/", 10853)
            Console.WriteLine(nodeId13)


            ' You can create a "null" node ID. Such node ID does not actually identify
any valid node in OPC UA, but
            ' is useful as a placeholder or as a starting point for further
```

```
modifications of its properties.
            Dim nodeId14 = New UANodeId()
            Console.WriteLine(nodeId14)


            ' Properties of a node ID can be modified individually. The advantage of
this approach is that you do
            ' not have to care about syntax of the node ID expanded text.
            Dim nodeId15 = New UANodeId()
            nodeId15.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId15.Identifier = 10853
            Console.WriteLine(nodeId15)


            ' The same as above, but using an object initializer list.
            Dim nodeId16 = New UANodeId With
            {
                .NamespaceUriString = "http://test.org/UA/Data/",
                .Identifier = 10853
            }
            Console.WriteLine(nodeId16)


            ' If you know the type of the identifier upfront, it is safer to use typed
properties that correspond
            ' to specific types of identifier. Here, with an integer identifier.
            Dim nodeId17 = New UANodeId()
            nodeId17.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId17.NumericIdentifier = 10853
            Console.WriteLine(nodeId17)


            ' Similarly, with a string identifier.
            Dim nodeId18 = New UANodeId()
            nodeId18.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId18.StringIdentifier = "someIdentifier"
            Console.WriteLine(nodeId18)


            ' Similarly, with a GUID identifier.
            Dim nodeId19 = New UANodeId()
            nodeId19.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId19.GuidIdentifier = Guid.Parse("BAEAF004-1E43-4A06-9EF0-
E52010D5CD10")
            Console.WriteLine(nodeId19)


            ' If you have GUID in its string form, the node ID object can parse it for
you.
            Dim nodeId20 = New UANodeId()
            nodeId20.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId20.GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10"
            Console.WriteLine(nodeId20)


            ' And, with an opaque identifier.
            Dim nodeId21 = New UANodeId()
```

```
            nodeId21.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId21.OpaqueIdentifier = {&H0, &HFF}
            Console.WriteLine(nodeId21)


            ' Assigning an expanded text to a node ID parses the value being assigned
and sets all corresponding
            ' properties accordingly.
            Dim nodeId22 = New UANodeId()
            nodeId22.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"
            Console.WriteLine(nodeId22)


            ' There is an implicit conversion from a string (representing the expanded
text) to a node ID.
            ' You can therefore use the expanded text (string) in place of any node ID
object directly.
            Dim nodeId23 = "nsu=http://test.org/UA/Data/;i=10853"
            Console.WriteLine(nodeId23)


            ' There is a copy constructor as well, creating a clone of an existing node
ID.
            Dim nodeId24a = New UANodeId("nsu=http://test.org/UA/Data/;i=10853")
            Console.WriteLine(nodeId24a)
            Dim nodeId24b = New UANodeId(nodeId24a)
            Console.WriteLine(nodeId24b)


            ' We have provided static classes with properties that correspond to all
standard nodes specified by
            ' OPC UA. You can simply refer to these node IDs in your code.
            ' The class names are UADataTypeIds, UAMethodIds, UAObjectIds,
UAObjectTypeIds, UAReferenceTypeIds,
            ' UAVariableIds and UAVariableTypeIds.
            Dim nodeId25 = UAObjectIds.TypesFolder
            Console.WriteLine(nodeId25)


            ' You can also refer to any standard node using its name (in a string
form).
            ' Note that assigning a non-existing standard name is not allowed, and
throws ArgumentException.
            Dim nodeId26 = New UANodeId()
            nodeId26.StandardName = "TypesFolder"
            Console.WriteLine(nodeId26)


            ' When you browse for nodes in the OPC UA server, every returned node
element contains a node ID that
            ' you can use further.
            Dim client27 = New EasyUAClient()
            Dim nodeElementCollection27 = client27.Browse(
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                UAObjectIds.Server,
                New UABrowseParameters(UANodeClass.All,
{UAReferenceTypeIds.References}))
```

```
            Dim nodeId27 = nodeElementCollection27(0).NodeId
            Console.WriteLine(nodeId27)


            ' As above, but using a constructor that takes a node element as an input.
            Dim client28 = New EasyUAClient()
            Dim nodeElementCollection28 = client28.Browse(
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                UAObjectIds.Server,
                New UABrowseParameters(UANodeClass.All,
{UAReferenceTypeIds.References}))
            Dim nodeId28 = New UANodeId(nodeElementCollection28(0))
            Console.WriteLine(nodeId28)


            ' Or, there is an explicit conversion from a node element as well.
            Dim client29 = New EasyUAClient()
            Dim nodeElementCollection29 = client29.Browse(
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                UAObjectIds.Server,
                New UABrowseParameters(UANodeClass.All,
{UAReferenceTypeIds.References}))
            Dim nodeId29 = CType(nodeElementCollection29(0), UANodeId)
            Console.WriteLine(nodeId29)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows different ways of constructing OPC UA node IDs.

Option Explicit

' A node ID specifies a namespace (either by an URI or by an index), and an identifier.
' The identifier can be numeric (an integer), string, GUID, or opaque.

Const UANodeClass_All = 255


' A node ID can be specified in string form (so-called expanded text).
' The code below specifies a namespace URI (nsu=...), and an integer identifier
(i=...).
' Assigning an expanded text to a node ID parses the value being assigned and sets all
corresponding
' properties accordingly.
Dim NodeId1: Set NodeId1 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId1.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"
WScript.Echo NodeId1


' Similarly, with a string identifier (s=...).
Dim NodeId2: Set NodeId2 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId2.ExpandedText = "nsu=http://test.org/UA/Data/;s=someIdentifier"
WScript.Echo NodeId2
```

```
' Actually, "s=" can be omitted (not recommended, though)
Dim NodeId3: Set NodeId3 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId3.ExpandedText = "nsu=http://test.org/UA/Data/;someIdentifier"
WScript.Echo NodeId3
' Notice that the output is normalized - the "s=" is added again.


' Similarly, with a GUID identifier (g=...)
Dim NodeId4: Set NodeId4 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId4.ExpandedText = "nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-4A06-9EF0-
E52010D5CD10"
WScript.Echo NodeId4
' Notice that the output is normalized - uppercase letters in the GUI are converted to
lowercase, etc.


' Similarly, with an opaque identifier (b=..., in Base64 encoding).
Dim NodeId5: Set NodeId5 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId5.ExpandedText = "nsu=http://test.org/UA/Data/;b=AP8="
WScript.Echo NodeId5


' Namespace index can be used instead of namespace URI. The server is allowed to change
the namespace
' indices between sessions (except for namespace 0), and for this reason, you should
avoid the use of
' namespace indices, and rather use the namespace URIs whenever possible.
Dim NodeId6: Set NodeId6 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId6.ExpandedText = "ns=2;i=10853"
WScript.Echo NodeId6


' Namespace index can be also specified together with namespace URI. This is still
safe, but may be
' a bit quicker to perform, because the client can just verify the namespace URI
instead of looking
' it up.
Dim NodeId7: Set NodeId7 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId7.ExpandedText = "nsu=http://test.org/UA/Data/;ns=2;i=10853"
WScript.Echo NodeId7


' When neither namespace URI nor namespace index are given, the node ID is assumed to
be in namespace
' with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by OPC UA
standard. There are
' many standard nodes that live in this reserved namespace, but no nodes specific to
your servers will
' be in the reserved namespace, and hence the need to specify the namespace with
server-specific nodes.
Dim NodeId8: Set NodeId8 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId8.ExpandedText = "i=2254"
WScript.Echo NodeId8


' If you attempt to pass in a string that does not conform to the syntax rules,
' a UANodeIdFormatException is thrown.
```

```
Dim NodeId9: Set NodeId9 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
On Error Resume Next
NodeId9.ExpandedText = "nsu=http://test.org/UA/Data/;i=notAnInteger"
If Err.Number = 0 Then
    WScript.Echo NodeId9
Else
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
End If
On Error Goto 0


' There is a parser object that can be used to parse the expanded texts of node IDs.
Dim NodeIdParser10: Set NodeIdParser10 =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser")
Dim NodeId10: Set NodeId10 =
NodeIdParser10.Parse("nsu=http://test.org/UA/Data/;i=10853", False)
WScript.Echo NodeId10


' The parser can be used if you want to parse the expanded text of the node ID but do
not want
' exceptions be thrown.
Dim NodeIdParser11: Set NodeIdParser11 =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser")
Dim NodeId11
Dim StringParsingError: Set StringParsingError =
NodeIdParser11.TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger", False, NodeId11)
If StringParsingError Is Nothing Then
    WScript.Echo NodeId11
Else
    WScript.Echo "*** Failure: " & StringParsingError.Message
End If


' You can also use the parser if you have node IDs where you want the default namespace
be different
' from the standard "http://opcfoundation.org/UA/".
Dim NodeIdParser12: Set NodeIdParser12 =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser")
NodeIdParser12.DefaultNamespaceUriString = "http://test.org/UA/Data/"
Dim NodeId12: Set NodeId12 = NodeIdParser12.Parse("i=10853", False)
WScript.Echo NodeId12


' You can create a "null" node ID. Such node ID does not actually identify any valid
node in OPC UA, but
' is useful as a placeholder or as a starting point for further modifications of its
properties.
Dim NodeId14: Set NodeId14 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
WScript.Echo NodeId14


' Properties of a node ID can be modified individually. The advantage of this approach
is that you do
' not have to care about syntax of the node ID expanded text.
Dim NodeId15: Set NodeId15 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId15.NamespaceUriString = "http://test.org/UA/Data/"
```

```
NodeId15.Identifier = 10853
WScript.Echo NodeId15


' If you know the type of the identifier upfront, it is safer to use typed properties
that correspond
' to specific types of identifier. Here, with an integer identifier.
Dim NodeId17: Set NodeId17 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId17.NamespaceUriString = "http://test.org/UA/Data/"
NodeId17.NumericIdentifier = 10853
WScript.Echo NodeId17


' Similarly, with a string identifier.
Dim NodeId18: Set NodeId18 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId18.NamespaceUriString = "http://test.org/UA/Data/"
NodeId18.StringIdentifier = "someIdentifier"
WScript.Echo NodeId18


' If you have GUID in its string form, the node ID object can parse it for you.
Dim NodeId20: Set NodeId20 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId20.NamespaceUriString = "http://test.org/UA/Data/"
NodeId20.GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10"
WScript.Echo NodeId20


' And, with an opaque identifier.
Dim NodeId21: Set NodeId21 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId21.NamespaceUriString = "http://test.org/UA/Data/"
NodeId21.OpaqueIdentifier = Array(&H00, &HFF)
WScript.Echo NodeId21


' We have built-in a list of all standard nodes specified by OPC UA. You can simply
refer to these node IDs in your code.
' You can refer to any standard node using its name (in a string form).
' Note that assigning a non-existing standard name is not allowed, and throws
ArgumentException.
Dim NodeId26: Set NodeId26 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId26.StandardName = "TypesFolder"
WScript.Echo NodeId26
' When the UANodeId equals to one of the standard nodes, it is output in the shortened
form - as the standard name only.


' When you browse for nodes in the OPC UA server, every returned node element contains
a node ID that
' you can use further.
Dim Client27: Set Client27 = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
' Browse from the Server node.
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
```

```
Dim ServerNodeDescriptor: Set ServerNodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UANodeDescriptor")
ServerNodeDescriptor.NodeId = ServerNodeId
' Browse all References.
Dim ReferencesNodeId: Set ReferencesNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ReferencesNodeId.StandardName = "References"
'
Dim BrowseParameters: Set BrowseParameters =
CreateObject("OpcLabs.EasyOpc.UA.UABrowseParameters")
BrowseParameters.NodeClasses = UANodeClass_All  ' this is the default, anyway
BrowseParameters.ReferenceTypeIds.Add ReferencesNodeId
'
On Error Resume Next
Dim NodeElementCollection27: Set NodeElementCollection27 = Client27.Browse( _
    EndpointDescriptor, ServerNodeDescriptor, BrowseParameters)
If Err.Number = 0 Then
    If NodeElementCollection27.Count <> 0 Then
        Dim NodeId27: Set NodeId27 = NodeElementCollection27(0).NodeId
        WScript.Echo NodeId27
    End If
Else
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
End If
On Error Goto 0
```

4.11.2.1 Namespace indices in Node Ids

During most communication between the client and the server, OPC Unified Architecture uses namespace indices, which are integer numbers, to distinguish between namespaces. The namespaces indices reference a namespace table made available by the server, and only the namespace table actually contains the namespace URIs. The order of entries in the server's namespace table can change between sessions. For this reason, namespace indices should not be persisted, as they alone are not generally sufficient to identify a namespace and thus a node.

There are, however, situations where you may want to include the namespace index in the expanded node Id string:

- For optimization. If the namespace index is given, it may be slightly faster to reference the node.
- Sometimes the party you are working with does not fully understand the consequences of using a namespace index, and only gives you node Ids with the namespace indices, without the namespace URIs.

For situations like this, OPC Data Client-UA allows the use of "ns=..." notation for a namespace index inside the expanded node ID strings, and has two additional provisions:

- When expanded node ID strings are provided out of OPC Data Client-UA components, they contain BOTH the namespace URI, and the namespace index. For example, the browsing may return following expanded node ID string to you:

  ```
  nsu=http://test.org/UA/Data/;ns=2;i=10219
  ```

- When OPC Data Client-UA receives an expanded node ID string on input, it uses the namespace URI specified in

the string by the “nsu=…” notation, if present. If the namespace URI is not present, a namespace index, specified by the “ns=…” notation (or defaulted to zero), is used. If both the namespace URI and the namespace index are present, semantically it always behaves the same as if only the namespace URI was specified, but the namespace index may be used for performance optimization. This means that the expanded node ID strings provided out of the component always refer to nodes through the namespace URI, even if they (in addition) contain the namespace index.

Note that namespace index 0 (zero) is reserved for the standard namespace maintained by the OPC Foundation ("http://opcfoundation.org/UA/"), and is not usually specified inside the expanded node ID strings. In addition (as an exception to the rule laid out above), it is safe to use nodes inside namespace index 0 without specifying their namespace URI, because the namespace index is "tied" to the URI by the provision in the standard, and the OPC servers are not allowed to change that.

4.11.2.2 Standard Node IDs

OPC Data Client-UA provides static classes with standard Node IDs defined by OPC Foundation: UADataTypeIds, UAMethodIds, UAObjectIds, UAObjectTypeIds, UAReferenceTypeIds, UAVariableIds, and UAVariableTypeIds. You can use properties of these static classes to easily specify “well-known” nodes you need to refer to, for example, the “ObjectsFolder” node which is a common starting node for browsing.

There are following classes with standard Node IDs for OPC-UA:

- UADataTypeIds: A class that declares constants for all Data Types in the Model Design. This class contains definitions for nodes such as BaseDataType, Number, Integer, Boolean, and so on.
- UAMethodIds: A class that declares constants for all MethodIds. in the Model Design.This class contains definitions for method nodes for condition handling, status machine handling, and others.
- UAObjectIds: A class that declares constants for Objects in the Model Design. This class contains definitions for folder nodes such as DataTypesFolder, EventTypesFolder, ObjectsFolder, ObjectTypesFolder, ReferenceTypesFolder, RootFolder, Server, TypesFolder, VariableTypesFolder, ViewsFolder, and others.
- UAObjectTypeIds: A class that declares constants for Object Types in the Model Design. This class contains definitions for nodes such as BaseObjectType, FolderType, and many other object types.
- UAReferenceTypeIds: A class that declares constants for Reference Types in the Model Design. This class contains definitions for nodes such as Aggregates, HasChild, HasComponent, HasProperty, HierarchicalReferences, Organizes, and more.
- UAVariableIds: A class that declares constants for all Variables in the Model Design.
- UAVariableTypeIds: A class that declares constants for all Variable Types in the Model Design.

When you need to pass a standard node to any method, simply use the corresponding property from the static class, such as UAObjectIds.ObjectsFolder.

4.11.3 OPC UA Qualified Names

In OPC-UA, the qualified name comprises of a namespace, and a text portion (a name). The qualified name is used, for example, as browse name (i.e. it is one of the information pieces that you obtain for a node when browsing nodes in the OPC-UA address space). In the opposite direction (from your application to the server), qualified names are used in OPC-UA browse paths, to specify the target node of each element in the browse path.

In This Topic

OPC Data Client-UA has a UAQualifiedName object for representing the qualified names. You can access individual parts of the qualified name by corresponding properties on this object (namely, the NamespaceUriString and Name properties). The ExpandedText property contains a string that fully specifies the qualified name, borrowing a syntax used for expanded node IDs described in an earlier article (except that in this case, the “identifier” part is always of a string type).

> In OPC Data Client, when formatting the expanded text of OPC UA Node IDs and qualified names, the namespace URI (the text after "nsu=") is separated with an additional space from the following semicolon, allowing the URI be easily separated by tools that are not aware of the specific syntax of OPC UA Node IDs and qualified names. Similarly, any trailing whitespace in the namespace URI part (the text after "nsu=") is ignored when parsing the OPC UA Node IDs

> and qualified names.

Standard qualified names

OPC Data Client-UA provides a static class with standard qualified names defined by OPC Foundation: UABrowseNames. You can use properties of this static class to easily specify “well-known” browse names you need to refer to. For example, the “SessionsDiagnosticsSummary” browse name is defined by the UABrowseNames.SessionsDiagnosticsSummary property, and you can pass the value of this property to any method that expects UAQualifiedName on input.

4.11.3.1 Namespace indices in qualified names

During most communication between the client and the server, OPC Unified Architecture uses namespace indices, which are integer numbers, to distinguish between namespaces. The namespaces indices reference a namespace table made available by the server, and only the namespace table actually contains the namespace URIs. The order of entries in the server’s namespace table can change between sessions. For this reason, namespace indices should not be persisted, as they alone are not generally sufficient to identify a namespace and thus a qualified name.

There are, however, situations where you may want to include the namespace index in the expanded qualified name string:

- For optimization. If the namespace index is given, it may be slightly faster to reference the qualified name.
- Sometimes the party you are working with does not fully understand the consequences of using a namespace index, and only gives you qualified names with the namespace indices, without the namespace URIs.

For situations like this, OPC Data Client-UA allows the use of “ns=…” notation for a namespace index inside the expanded qualified name strings, and has two additional provisions:

- When expanded qualified name strings are provided out of OPC Data Client-UA components, they contain BOTH the namespace URI, and the namespace index. For example, the browsing may return following expanded qualified name string to you:

  ```
  nsu=http://test.org/UA/Data/;ns=2;s=Data
  ```

- When OPC Data Client-UA receives an expanded qualified name string on input, it uses the namespace URI specified in the string by the “nsu=…” notation, if present. If the namespace URI is not present, a namespace index, specified by the “ns=…” notation (or defaulted to zero), is used. If both the namespace URI and the namespace index are present, semantically it always behaves the same as if only the namespace URI was specified, but the namespace index may be used for performance optimization. This means that the expanded qualified name strings provided out of the component always refer to qualified names through the namespace URI, even if they (in addition) contain the namespace index.

Note that namespace index 0 (zero) is reserved for the standard namespace maintained by the OPC Foundation ("http://opcfoundation.org/UA/"), and is not usually specified inside the expanded qualified name strings.

4.11.4 Browse Paths in OPC-UA

Browse Paths In General

An alternative way (to using a node ID) of specifying a node in OPC address space is using a browse path. The browse path is a sequence of node names, starting from a known location (e.g. the root of the OPC address space). The browse path can also be expressed as a string, where the individual node names are separated by delimiters (e.g. "/", a slash).

Browse paths can be absolute (starting from a specified node), or relative.

The advantage of the browse paths is that you can construct them yourself, with just knowledge of the node names, without knowing the full node IDs. In the end, each item must be addressed by its node ID, and OPC Data Client will resolve the browse path to a node ID as necessary.

OPC UA Browse Paths Specifics

In OPC Unified Architecture, each element of the browse path is not a simple string, but instead, it is a structured object (UABrowsePathElement) which contains more information that allows identifying the child node.

The OPC-UA browse path element contains a qualified name of the child node (a name qualified with a namespace URI), in the UABrowsePathElement.TargetName property. This allows ensuring the uniqueness of browse path elements in complex OPC-UA servers, such as when the address space is aggregated from multiple sources.

Because in OPC-UA there may be various references that lead from the parent node to a child node, the OPC-UA browse paths need to specify which type of reference should be used. This is done by the UABrowsePathElement.ReferenceTypeId property; in addition, it is possible to specify whether the reference should be followed in reverse direction (UABrowsePathElement.ReferenceIsInverse), and whether subtypes of the given reference type should also be considered.

An absolute browse path for OPC-UA is contained in a UABrowsePath object. The starting node of an OPC-UA browse path is contained in the UABrowsePath.StartingNodeId property. Browse paths are commonly specified by the BrowsePath property of the UANodeDescriptor object.

The UABrowsePath can be a null browse path, in case its StartingNodeId is a null node ID. A null browse path is used to specify that no browse path is known or given. Note that passing a null browse path object is different from passing a null reference to UABrowsePath (which is not allowed in most places).

If a non-null NodeId is specified in the descriptor, OPC Data Client will use this node Id and ignore the browse path. If NodeId is null, OPC Data Client will attempt to resolve the browse path contained in the BrowsePath property of the descriptor. Either node ID, or browse path (or both) must be specified.

4.11.4.1 OPC UA Browse Path Elements

OPC UA browse path elements are represented by UABrowsePathElement objects. Each browse path element has following data members:

- ReferenceTypeId Property. The type of reference (an instance of UANodeId Class) to follow from the current node. The current path cannot be followed any further if the reference type is not available on the node instance.
- ReferenceIsInverse Property. Indicates whether the inverse Reference should be followed.
- IncludeSubtypes Property. Indicates whether subtypes of the ReferenceType should be followed.
- TargetName Property. The browse name of the target node (an instance of UAQualifiedName Class).

The UABrowsePathElement has various constructor overloads with different combinations of these parameters. In addition, there are static methods on the UABrowsePathElement Class that allow you to easily create commonly used browse path elements:

- The Create Method creates a new OPC-UA browse path element with given reference type and target name, for following in forward direction, and including subtypes.

- The CreateInverse Method creates a new OPC-UA browse path element with given reference type and target name, for following in inverse direction, and including subtypes.
- The CreateSimple Method creates a new OPC-UA browse path element with given target name, and specifying any hierarchical forward reference.
- There are static methods for creation of browse path elements for all standard *concrete* references, taking just the target name as input. For example, the HasProperty Method creates a browse path element for following a specific HasProperty reference in a forward direction.

The created browse path elements can then be added to a UABrowsePathElementCollection Class, forming a relative path.

4.11.4.2 OPC-UA Browse Path Format

In This Topic

OPC Data Client-UA uses the relative browse path string format recommended by the OPC specification, but extends it by an ability to specify the starting node ID (and therefore introducing the absolute browse paths), and by an ability to specify the namespaces not by an index into a namespace table, but by a namespace URI. The precise syntax of these extensions is beyond the current scope of this document, and the description below is just a close approximation.

An OPC-UA browse path can be represented by a string; in such case, it can express either a relative or absolute browse path (the UABrowsePath object cannot hold relative browse paths).

As mentioned above, the format of the browse path string is such that the individual elements (usually, node names) are separated by delimiters. In OPC-UA, the delimiter actually precedes every element, i.e. there is a delimiter also at the beginning of the relative browse path. Different delimiters are available for specifying various reference types. Following delimiters are available:

Delimiter	Description
/	Follow any subtype of HierarchicalReferences (technically, numerical node Id 44 in namespace 0).
.	Follow any subtype of Aggregates reference type (technically, numerical node Id 33 in namespace 0). This is stricter than '/'.
<referenceSpecifier>	Follow the specified reference type. A '#' placed in front of the reference type name indicates that subtypes should not be followed. A '!' in front of the reference type name is used to indicate that the inverse reference should be followed.

If the OPC-UA browse path does not start with any of the above delimiters, it denotes an absolute path that starts from the node specified at the beginning of the string. For example, "[Objects]/Boilers" is an example of such absolute OPC-UA browse path, whereas "/Boilers" in OPC-UA is a relative browse path.

The string form of an OPC-UA browse path is a concept on the client side (OPC Data Client), and the browse paths strings are not directly processed by OPC-UA servers.

Because the node names can contain any characters, we need to consider situations in which the node name itself contains a slash ('/') or a dot ('.'). In such case, in the string format of browse paths, these characters are escaped by preceding them with an ampersand ('&'). An ampersand itself needs to be escaped, too.

Parsing

Parsing an absolute browse path:

C#

```
// Parses an absolute  OPC-UA browse path and displays its starting node and elements.

using System;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class Parse
    {
        public static void Main1()
        {
            var browsePathParser = new UABrowsePathParser();
            UABrowsePath browsePath;
            try
            {
                browsePath = browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar");
            }
            catch (UABrowsePathFormatException browsePathFormatException)
            {
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId);

            foreach (UABrowsePathElement browsePathElement in browsePath.Elements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // StartingNodeId: ObjectsFolder
            // /Data
            // /Static
            // /UserScalar
        }
    }
}
```

Object Pascal

```
// Parses an absolute  OPC-UA browse path and displays its starting node and elements.

class procedure Parse.Main;
var
  BrowsePath: _UABrowsePath;
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  Count: Cardinal;
  Element: OleVariant;
```

```
  ElementEnumerator: IEnumVariant;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;

  try
    BrowsePath := BrowsePathParser.Parse('[ObjectsFolder]/Data/Static/UserScalar');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('StartingNodeId: ', BrowsePath.StartingNodeId.ToString);

  WriteLn('Elements:');
  ElementEnumerator := BrowsePath.Elements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // StartingNodeId: ObjectsFolder
  // Elements:
  // /Data
  // /Static
  // /UserScalar

end;
```

PHP

```
// Parses an absolute  OPC-UA browse path and displays its starting node and elements.


$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

try
{
    $BrowsePath = $BrowsePathParser->Parse("[ObjectsFolder]/Data/Static/UserScalar");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("StartingNodeId: %s\n", $BrowsePath->StartingNodeId);

printf("Elements:\n");
```

```
for ($i = 0; $i < $BrowsePath->Elements->Count; $i++)
{
    printf("%s\n", $BrowsePath->Elements[$i]);
}

// Example output:
// StartingNodeId: ObjectsFolder
// Elements:
// /Data
// /Static
// /UserScalar
```

VB.NET

```
' Parses an absolute  OPC-UA browse path and displays its starting node and elements.

Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class Parse
        Public Shared Sub Main1()
            Dim browsePathParser = New UABrowsePathParser()
            Dim browsePath As UABrowsePath
            Try
                browsePath = browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar")
            Catch browsePathFormatException As UABrowsePathFormatException
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId)

            For Each browsePathElement As UABrowsePathElement In browsePath.Elements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' StartingNodeId: ObjectsFolder
            ' /Data
            ' /Static
            ' /UserScalar
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Parses an absolute  OPC-UA browse path and displays its starting node and elements.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
```

```
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
On Error Resume Next
Dim BrowsePath: Set BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "StartingNodeId: " & BrowsePath.StartingNodeId

WScript.Echo "Elements:"
Dim BrowsePathElement: For Each BrowsePathElement In BrowsePath.Elements
    WScript.Echo BrowsePathElement
Next
```

Attempting to parse an absolute browse path:

C#

```
// Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

using System;
using OpcLabs.BaseLib;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class TryParse
    {
        public static void Main1()
        {
            var browsePathParser = new UABrowsePathParser();

            IStringParsingError stringParsingError = browsePathParser.TryParse(
                "[ObjectsFolder]/Data/Static/UserScalar",
                out UABrowsePath browsePath);

            // Display results
            if (stringParsingError != null)
            {
                Console.WriteLine("*** Error: {0}", stringParsingError);
                return;
            }

            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId);

            foreach (UABrowsePathElement browsePathElement in browsePath.Elements)
                Console.WriteLine(browsePathElement);
```

```
            // Example output:
            // StartingNodeId: ObjectsFolder
            // /Data
            // /Static
            // /UserScalar
        }
    }
}
```

Object Pascal

```
// Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

class procedure TryParse.Main;
var
  BrowsePath: _UABrowsePath;
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  BrowsePathResult: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
  StringParsingError: _StringParsingError;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;

  StringParsingError :=
BrowsePathParser.TryParse('[ObjectsFolder]/Data/Static/UserScalar', BrowsePathResult);

  // Display results
  if StringParsingError <> nil then
  begin
    WriteLn('*** Error: ', StringParsingError.ToString);
    Exit;
  end;

  BrowsePath := IUnknown(BrowsePathResult) as _UABrowsePath;
  WriteLn('StartingNodeId: ', BrowsePath.StartingNodeId.ToString);

  WriteLn('Elements:');
  ElementEnumerator := BrowsePath.Elements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // StartingNodeId: ObjectsFolder
  // Elements:
  // /Data
  // /Static
  // /UserScalar

end;
```

PHP

```
// Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

$BrowsePath = new COM("OpcLabs.EasyOpc.UA.Navigation.UABrowsePath");

$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

$StringParsingError = $BrowsePathParser->TryParse("
[ObjectsFolder]/Data/Static/UserScalar", $BrowsePath);

// Display results
if (!is_null($StringParsingError)) {
    printf("*** Error: %s\n", $StringParsingError);
    exit();
}

printf("StartingNodeId: %s\n", $BrowsePath->StartingNodeId);

printf("Elements:\n");

for ($i = 0; $i < $BrowsePath->Elements->Count; $i++)
{
    printf("%s\n", $BrowsePath->Elements[$i]);
}

// Example output:
// StartingNodeId: ObjectsFolder
// Elements:
// /Data
// /Static
// /UserScalar
```

VB.NET

```
' Attempts to parses an absolute  OPC-UA browse path and displays its starting node and
elements.

Imports System
Imports OpcLabs.BaseLib
Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class TryParse
        Public Shared Sub Main1()
            Dim browsePathParser = New UABrowsePathParser()

            Dim browsePath As UABrowsePath = Nothing
            Dim stringParsingError As IStringParsingError = browsePathParser.TryParse("
[ObjectsFolder]/Data/Static/UserScalar", browsePath)

            ' Display results
```

```
            If Not stringParsingError Is Nothing Then
                Console.WriteLine("*** Error: {0}", stringParsingError)
                Exit Sub
            End If

            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId)

            For Each browsePathElement As UABrowsePathElement In browsePath.Elements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' StartingNodeId: ObjectsFolder
            ' /Data
            ' /Static
            ' /UserScalar
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
Dim BrowsePath
Dim StringParsingError: Set StringParsingError = BrowsePathParser.TryParse("
[ObjectsFolder]/Data/Static/UserScalar", BrowsePath)

' Display results
If Not (StringParsingError Is Nothing) Then
    WScript.Echo "*** Error: " & StringParsingError
    WScript.Quit
End If

WScript.Echo "StartingNodeId: " & BrowsePath.StartingNodeId

WScript.Echo "Elements:"
Dim BrowsePathElement: For Each BrowsePathElement In BrowsePath.Elements
    WScript.Echo BrowsePathElement
Next
```

Parsing a relative browse path:

C#

```
// Parses a relative OPC-UA browse path and displays its elements.

using System;
```

```
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class ParseRelative
    {
        public static void Main1()
        {
            var browsePathParser = new UABrowsePathParser();
            UABrowsePathElementCollection browsePathElements;
            try
            {
                browsePathElements =
browsePathParser.ParseRelative("/Data.Dynamic.Scalar.CycleComplete");
            }
            catch (UABrowsePathFormatException browsePathFormatException)
            {
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UABrowsePathElement browsePathElement in browsePathElements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // /Data
            // .Dynamic
            // .Scalar
            // .CycleComplete
        }
    }
}
```

Object Pascal

```
// Parses a relative OPC-UA browse path and displays its elements.

class procedure ParseRelative.Main;
var
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathElements: _UABrowsePathElementCollection;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;

  try
    BrowsePathElements :=
BrowsePathParser.ParseRelative('/Data.Dynamic.Scalar.CycleComplete');
  except
    on E: EOleException do
    begin
```

```
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  ElementEnumerator := BrowsePathElements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // /Data
  // .Dynamic
  // .Scalar
  // .CycleComplete

end;
```

PHP

```
// Parses a relative OPC-UA browse path and displays its elements.


$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

try
{
    $BrowsePathElements = $BrowsePathParser-
>ParseRelative("/Data.Dynamic.Scalar.CycleComplete");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
for ($i = 0; $i < $BrowsePathElements->Count; $i++)
{
    printf("%s\n", $BrowsePathElements[$i]);
}

// Example output:
// /Data
// .Dynamic
// .Scalar
// .CycleComplete
```

VB.NET

```
' Parses a relative OPC-UA browse path and displays its elements.
```

```
Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class ParseRelative
        Public Shared Sub Main1()
            Dim browsePathParser = New UABrowsePathParser()
            Dim browsePathElements As UABrowsePathElementCollection
            Try
                browsePathElements =
browsePathParser.ParseRelative("/Data.Dynamic.Scalar.CycleComplete")
            Catch browsePathFormatException As UABrowsePathFormatException
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each browsePathElement As UABrowsePathElement In browsePathElements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' /Data
            ' .Dynamic
            ' .Scalar
            ' .CycleComplete
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Parses a relative OPC-UA browse path and displays its elements.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
On Error Resume Next
Dim BrowsePathElements: Set BrowsePathElements =
BrowsePathParser.ParseRelative("/Data.Dynamic.Scalar.CycleComplete")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim BrowsePathElement: For Each BrowsePathElement In BrowsePathElements
    WScript.Echo BrowsePathElement
Next
```

Attempting to parse a relative browse path:

C#

```
// Attempts to parse a relative OPC-UA browse path and displays its elements.

using System;
using OpcLabs.BaseLib;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class TryParseRelative
    {
        public static void Main1()
        {
            var browsePathElements = new UABrowsePathElementCollection();

            var browsePathParser = new UABrowsePathParser();
            IStringParsingError stringParsingError =
browsePathParser.TryParseRelative("/Data.Dynamic.Scalar.CycleComplete",
browsePathElements);

            // Display results
            if (stringParsingError != null)
            {
                Console.WriteLine("*** Error: {0}", stringParsingError);
                return;
            }

            foreach (UABrowsePathElement browsePathElement in browsePathElements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // /Data
            // .Dynamic
            // .Scalar
            // .CycleComplete
        }
    }
}
```

Object Pascal

```
// Attempts to parse a relative OPC-UA browse path and displays its elements.

class procedure TryParseRelative.Main;
var
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathElements: _UABrowsePathElementCollection;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
  StringParsingError: _StringParsingError;
```

```
begin
  BrowsePathElements := CoUABrowsePathElementCollection.Create;

  BrowsePathParser := CoUABrowsePathParser.Create;
  StringParsingError :=
BrowsePathParser.TryParseRelative('/Data.Dynamic.Scalar.CycleComplete',
BrowsePathElements);

  // Display results
  if StringParsingError <> nil then
  begin
    WriteLn('*** Error: ', StringParsingError.ToString);
    Exit;
  end;

  ElementEnumerator := BrowsePathElements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // /Data
  // .Dynamic
  // .Scalar
  // .CycleComplete

end;
```

PHP

```
// Attempts to parse a relative OPC-UA browse path and displays its elements.

$BrowsePathElements = new
COM("OpcLabs.EasyOpc.UA.Navigation.UABrowsePathElementCollection");

$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

$StringParsingError = $BrowsePathParser-
>TryParseRelative("/Data.Dynamic.Scalar.CycleComplete", $BrowsePathElements);

// Display results
if (!is_null($StringParsingError)) {
    printf("*** Error: %s\n", $StringParsingError);
    exit();
}

printf("StartingNodeId: %s\n", $BrowsePath->StartingNodeId);

printf("Elements:\n");

for ($i = 0; $i < $BrowsePathElements->Count; $i++)
{
    printf("%s\n", $BrowsePathElements[$i]);
}
```

```
// Example output:
// /Data
// .Dynamic
// .Scalar
// .CycleComplete
```

VB.NET

```
' Attempts to parse a relative OPC-UA browse path and displays its elements.

Imports System
Imports OpcLabs.BaseLib
Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class TryParseRelative
        Public Shared Sub Main1()
            Dim browsePathElements = New UABrowsePathElementCollection()

            Dim browsePathParser = New UABrowsePathParser()
            Dim stringParsingError As IStringParsingError =
browsePathParser.TryParseRelative("/Data.Dynamic.Scalar.CycleComplete",
browsePathElements)

            ' Display results
            If Not stringParsingError Is Nothing Then
                Console.WriteLine("*** Error: {0}", stringParsingError)
                Exit Sub
            End If

            For Each browsePathElement As UABrowsePathElement In browsePathElements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' /Data
            ' .Dynamic
            ' .Scalar
            ' .CycleComplete
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Attempts to parse a relative OPC-UA browse path and displays its elements.

Option Explicit

Dim BrowsePathElements: Set BrowsePathElements =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.UABrowsePathElementCollection")

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
```

```
Dim StringParsingError: Set StringParsingError =
BrowsePathParser.TryParseRelative("/Data.Dynamic.Scalar.CycleComplete",
BrowsePathElements)

' Display results
If Not (StringParsingError Is Nothing) Then
    WScript.Echo "*** Error: " & StringParsingError
    WScript.Quit
End If

Dim BrowsePathElement: For Each BrowsePathElement In BrowsePathElements
    WScript.Echo BrowsePathElement
Next
```

4.11.5 OPC UA Attribute IDs

Attribute is a primitive characteristic of a Node. All Attributes are defined by OPC UA, and may not be defined by clients or servers (i.e. the attribute set is not vendor-extensible). Attributes are the only elements in the address space permitted to have data values.

In most cases, you will probably be working with the Value attribute, which contains the actual value of a variable. If you need to address a different attribute, use the UAAttributeId enumeration, which contains constants for all attribute IDs available in OPC UA address space.

UAAttributeId can be given as AttributeId property of the UAAttributeArguments object (and derived objects, such as UAReadArguments, UAWriteArguments, UAWriteValueArguments, and EasyUAMonitoredItemArguments). You can then pass this object to a corresponding method on the main EasyUAClient object.

4.11.6 OPC UA Index Range Lists

OPC Unified Architecture has support for single- and multidimensional arrays in data values. In addition, it is also possible to work with (read, write, subscribe to) only a subset of an array, so that the whole (potentially large) array does not have to be transferred between the OPC server and OPC client.

OPC Data Client-UA uses Index Range Lists (UAIndexRangeList objects) to control which parts of array value should be accessed. An index range list can be used to identify the whole array, a single element of a structure or an array, or a single range of indexes for arrays. Index Range List is basically a list of individual Index Ranges (UAIndexRange objects), each containing a range for a single dimension of an array.

There is an implicit conversion from an integer (Int32) to a UAIndexRange, so if your language (such as C#) supports implicit conversions, an index range containing just a single element can be written simply as the element index, without explicitly constructing the UAIndexRange object.

Besides constructing the UAIndexRangeList from individual UAIndexRange objects, it is also possible to specify the index range list by parsing a string, using static UAIndexRangeList.TryParse method.

Examples of index range strings:

- 5:20
- 10:11,1:2

- 1,4:7,90

Dimensions are separated by commas. A minimum and maximum value, separated by a colon, denotes a range of index. If the colon is missing, only a single for an index is selected. An empty string denotes the whole array.

One-dimensional index range lists can be easily constructed using the UAIndexRangeList.OneDimension static method, passing it either the single index, or minimum and maximum indices.

UAIndexRangeList can be given as IndexRangeList property of the UAAttributeArguments object (and derived objects, such as UAReadArguments, UAWriteArguments, UAWriteValueArguments, and EasyUAMonitoredItemArguments). You can then pass this object to a corresponding method on the main EasyUAClient object.

C#

```
// This example shows how to read a range of values from an array.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UAIndexRangeList
{
    class Usage
    {
        public static void ReadValue()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain the value, indicating that just the elements 2 to 4 should be
returned
            object value;
            try
            {
                value = client.ReadValue(
                    new UAReadArguments(
                        endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;ns=2;i=10305",
                        UAIndexRangeList.OneDimension(2, 4)));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Cast to typed array
            var arrayValue = (Int32[]) value;

            // Display results
```

```
            for (int i = 0; i < 3; i++)
                Console.WriteLine("arrayValue[{0}]: {1}", i, arrayValue[i]);


            // Example output:
            //arrayValue[0]: 180410224
            //arrayValue[1]: 1919239969
            //arrayValue[2]: 1700185172
        }
    }
}
```

Object Pascal

```
// This example shows how to read a range of values from an array.

class procedure Usage.ReadValue;
var
  Client: _EasyUAClient;
  EndpointDescriptor: string;
  ReadArguments1: _UAReadArguments;
  Arguments, Results: OleVariant;
  I: Cardinal;
  IndexRange: _UAIndexRange;
  IndexRangeList: OpcLabs_EasyOpcUA_TLB._UAIndexRangeList;
  ArrayValue: OleVariant;
  Value: Integer;
  ValueResult: _ValueResult;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer';
  //or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Prepare the arguments, indicating that just the elements 2 to 4 should be
returned.
  IndexRangeList := CoUAIndexRangeList.Create;
  IndexRange := CoUAIndexRange.Create;
  IndexRange.Minimum := 2;
  IndexRange.Maximum := 4;
  IndexRangeList.Add(IndexRange);

  ReadArguments1 := CoUAReadArguments.Create;
  ReadArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  ReadArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;ns=2;i=10305';
  ReadArguments1.IndexRangeList := IndexRangeList;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := ReadArguments1;

  // Obtain value.
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultipleValues(Arguments));
```

```
  ValueResult := IInterface(Results[0]) as _ValueResult;
  if not ValueResult.Succeeded then
    begin
      WriteLn(' *** Failure: ', ValueResult.Exception.GetBaseException.Message);
      Exit;
    end;

  // Display results
  ArrayValue := ValueResult.Value;
  for I := VarArrayLowBound(ArrayValue, 1) to VarArrayHighBound(ArrayValue, 1) do
  begin
      Value := ArrayValue[I];
      WriteLn('arrayValue[', I, ']: ', Value);
  end;

  // Example output:
  //arrayValue[0]: 180410224
  //arrayValue[1]: 1919239969
  //arrayValue[2]: 1700185172
end;
```

PHP

```
// This example shows how to read a range of values from an array.

$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Prepare the arguments, indicating that just the elements 2 to 4 should be returned.
$IndexRangeList = new COM("OpcLabs.EasyOpc.UA.UAIndexRangeList");
$IndexRange = new COM("OpcLabs.EasyOpc.UA.UAIndexRange");
$IndexRange->Minimum = 2;
$IndexRange->Maximum = 4;
$IndexRangeList->Add($IndexRange);

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;ns=2;i=10305";
$ReadArguments1->IndexRangeList = $IndexRangeList;

$arguments[0] = $ReadArguments1;

// Obtain value.
$results = $Client->ReadMultipleValues($arguments);

$ValueResult = $results[0];

if (!$ValueResult->Succeeded) {
    printf("*** Failure: %s\n", $ValueResult->ErrorMessageBrief);
    Exit();
}
```

```
// Display results
$ArrayValue = $ValueResult->Value;
for ($i = 0; $i <= 2; $i++)
{
    printf("arrayValue[%d]: %s\n", $i, $ArrayValue[$i]);
}

// Example output:
//arrayValue[0]: 180410224
//arrayValue[1]: 1919239969
//arrayValue[2]: 1700185172
```

VB.NET

```
' This example shows how to read a range of values from an array.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._UAIndexRangeList
    Friend Class Usage
        Public Shared Sub ReadValue()

            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain the value, indicating that just the elements 2 to 4 should be
returned
            Dim value As Object
            Try
                value = client.ReadValue( _
                    New UAReadArguments( _
                        endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10305", _
                        UAIndexRangeList.OneDimension(2, 4)))
                ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Cast to typed array
            Dim arrayValue = DirectCast(value, Int32())

            ' Display results
            For i = 0 To 2
                Console.WriteLine("arrayValue[{0}]: {1}", i, arrayValue(i))
            Next
        End Sub
```

```
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to read a range of values from an array.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Prepare the arguments, indicating that just the elements 2 to 4 should be returned.
Dim IndexRangeList: Set IndexRangeList =
CreateObject("OpcLabs.EasyOpc.UA.UAIndexRangeList")
Dim IndexRange: Set IndexRange = CreateObject("OpcLabs.EasyOpc.UA.UAIndexRange")
IndexRange.Minimum = 2
IndexRange.Maximum = 4
IndexRangeList.Add IndexRange
'
Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = endpointDescriptor
ReadArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;ns=2;i=10305"
ReadArguments1.IndexRangeList = IndexRangeList

Dim arguments(0)
Set arguments(0) = ReadArguments1

' Obtain the value.
Dim results: results = Client.ReadMultipleValues(arguments)
Dim ValueResult: Set ValueResult = results(0)
If Not ValueResult.Succeeded Then
    WScript.Echo "*** Failure: " & ValueResult.Exception.GetBaseException().Message
    WScript.Quit
End If
' VBScript can only handle well arrays of VARIANTs; most other COM tool will be able to
use simply the .Value property.
Dim arrayValue: arrayValue = ValueResult.RegularizedValue

' Display results
Dim i: For i = 0 To 2
    WScript.Echo "arrayValue[" & i & "]: " & arrayValue(i)
Next


' Example output:
'arrayValue[0]: 180410224
'arrayValue[1]: 1919239969
'arrayValue[2]: 1700185172
```

4.12 Identifying information in OPC UA PubSub

There are two aspects, or levels, to identifying information in OPC UA PubSub.

The actual information being exchanged relies on **OPC UA PubSub physical identifiers (Section 4.12.1)**. Example of these are publisher Ids and dataset writer Ids. The physical identifiers are those that are actually used to establish connections with the message-oriented middleware, and are those that appear in the network messages transmitted by publishers and received by subscribers. Some physical identifiers are always needed at certain point - without them, OPC UA PubSub cannot work.

OPC UA PubSub also has a configuration model (which can reside in OPC UA Server or in a file), and the configuration model provides logical identifiers (mainly, names) for PubSub objects. The logical identifiers (**OPC UA PubSub logical identifiers (Section 4.12.2)**) are easier to work with for humans, and the configuration model has a hierarchical structure which makes it better to comprehend. The use of logical identifiers also allows the higher levels to accommodate to changes, because they provide a mapping mechanism and "shield" the consumers from the physical details. For example, a dataset writer Id (which is an integer) may change due to configuration changes, but as long as the consumer refers to the dataset writer using its logical name (which does not change), the communication can still be established without a problem. The use of this extra logical level is, however, entirely optional, and depends on the consuming application.

OPC Data Client provides way to work with either identifiers, and combines them together into **OPC UA PubSub Descriptors (Section 4.12.3)**. The transformation from logical to physical identifiers is done "behind the scenes" using a process known as **OPC UA PubSub logical resolution (Section 5.1.5.1.4)**.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

4.12.1 OPC UA PubSub physical identifiers

This article describes the physical identifiers used in OPC UA PubSub communication. For a distinction between physical and logical identifiers, see **Identifying information in OPC UA PubSub (Section 4.12)**.

In This Topic

Network Address and Resource Address

Network address is an abstract concept used to represent the basic information needed to make a PubSub connection to the message oriented middleware. In OPC Data Client, it represented by the NetworkAddress Class. The network address specifies a network interface used for the communication, with its InterfaceName Property. This is an empty string if you want to leave the choice of the network interface up to the component, or a non-empty string specifying the name of the network interface on the computer where the component runs.

Resource address is a concrete derivation of network address, in which the network address information is represented by a resource string - essentially, a URL string. Currently, all network addresses used in OPC UA PubSub are resource

addresses. In OPC Data Client, resource address is represented by the ResourceAddress Class. The actual URL information is stored in its ResourceDescriptor Property.

URL schemes and syntaxes

The URL schemes and syntaxes for OPC UA PubSub are described in OPC specifications. The ones currently supported by core OPC Data Client are:

- opc.udp://<host>[:<port>]
- opc.eth://<host>[:<VID>[.PCP]]

With UDP, both IPv4 and IPv6 addresses are supported.

> OPC Data Client supports Enhanced Host Name Resolution in OPC UA PubSub URLs. This allows you to conveniently and reliably use some pre-defined host names as shortcuts for IP and MAC addresses with the above URL schemes, such as "ip6-allrouters" or "eth-broadcast".

OPC Data Client will, by default, use MQTT protocol with any of the URL schemes below:

- mqtt:
- mqtts:
- ssl:
- tcp:
- ws:
- wss:

The details of the URL syntax are then left on the concrete MQTT communication package used (see MQTT communication packages). Some communication packages define useful extensions to the URL syntax, which are described with each communication package. The basic syntax options, described in the OPC UA PubSub specification, are supported by every communication (as long as the corresponding scheme is supported at all), i.e.:

- mqtts://<domain name>[:<port>][/<path>]
- mqtt://<domain name>[:<port>][/<path>]
- wss://<domain name>[:<port>][/<path>]

Publisher Id

The publisher Id is a unique identifier for a publisher with the message-oriented middleware. A publisher can use a publisher Id of one of these types:

- Byte (8-bit unsigned integer)
- UInt16 (16-bit unsigned integer)
- UInt32 (32-bit unsigned integer)
- UInt64 (64-bit unsigned integer)
- String

> Publisher Ids are only equal if both their type and their value are equal. For example, a UInt16 publisher Id with value 31 is different from UInt32 publisher Id with the same value. You therefore need to be careful when specifying the publisher Ids, and with numerical Ids, pay attention to their precise type. Also, a String publisher Id is different from any numerical publisher Id, even if it carries a string representation of the same numerical value (such as a string "31", in our example).

In OPC Data Client, the publisher Id is represented by the UAPublisherId Class, and the types of publisher Ids are expressed by the UAPublisherIdType Enumeration. The easiest way to create a publisher Id is to use one of the static CreateXXXX methods, defined on the class:

- CreateByte Method
- CreateUInt16 Method
- CreateUInt32 Method
- CreateUInt64 Method
- CreateString Method

In a **Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)**, you can also use a publisher Id with type None. This specifies that no filtering on publisher Ids should be performed.

Writer Group Id

The writer group Id is an unsigned 16-bit integer that identifies the writer group. It is unique across writer groups for a publisher Id. Writer groups can only have non-zero Ids, but a zero writer group Id may be used in a **Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)** to specify that no filtering on writer group Ids should be performed.

Dataset Writer Id

The dataset writer Id identifies the dataset writer in the writer group. It is an unsigned 16-bit integer that is unique across all dataset writers for a publisher Id. Dataset writers can only have non-zero Ids, but a zero dataset writer Id may be used in a **Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)** to specify that no filtering on dataset writer Ids should be performed.

Field Name and Field Index

Each dataset consists of a collection of dataset fields. Each field has a unique name within a dataset. Some message mappings (such as JSON) transfer the field names in the dataset message, others (such as UADP) do not, and they rely on the order of fields, which then must be the same as in the dataset metadata.

When OPC Data Client delivers information to you, it preferably uses field names, but it resorts to field indexes when the field names are not available. When you need to specify a dataset field to OPC Data Client, you can use either form (name or index). Normally, we recommend using field names and not field indexes, but an exception must be made in case when message mapping such as UADP is sued, and metadata (as a source of field names) is not available).

Dataset Metadata (OPC UA PubSub) (Section 5.1.5.1.3) can be provided through **OPC UA PubSub logical resolution (Section 5.1.5.1.4)**, or you can supply it manually (from the code).

4.12.2 OPC UA PubSub logical identifiers

The logical identifiers allow PubSub objects be found in the PubSub configuration.

PubSub Connection Name, Writer Group Name,

In This Topic

and DataSet Writer Name

A PubSub configuration contains a collection of PubSub connections, and each PubSub connection is identified by its name, which is a string. For publishers, a PubSub connection determines the physical publisher Id that is used by that publisher.

A PubSub connection contains a collection of writer groups, and each writer group is also identified by its name (a string). Each writer group has its physical writer group Id specified.

A writer group contains a collection of dataset writers, and each dataset writer is also identified by its name (a string). Each dataset writer has its physical dataset wroiter Id specified.

Published Dataset Name

In a PubSub configuration, each published dataset has a unique name (a string). Published datasets may be organized in a tree of (named) dataset folders. The name of published dataset, however, must still be unique across the whole PubSub configuration, not just in its dataset folder.

A dataset writer may be associated with a published dataset by means of published dataset name.

4.12.3 OPC UA PubSub Descriptors

This article describes the descriptors used in OPC UA PubSub communication. For a relation between physical and logical identifiers and descriptors, see **Identifying information in OPC UA PubSub (Section 4.12)**.

In This Topic

Explanation of Terms

In order to be able to describe how OPC Data Client handles the logical and physical identifiers, especially in relation to **OPC UA PubSub logical resolution (Section 5.1.5.1.4)**, we need to define some terms first. There are also properties and methods that use these terms in their names, so knowing what they mean is beneficial to understand their purpose.

We say that a descriptor or filter *strictly physical* when it does not have a logical identifier in itself - that is, the name of the PubSub object is empty - and all its constituent descriptors (if there are any) are also strictly physical.

We say that a descriptor or filter *requires resolution* when an access to PubSub configuration is necessary in order to interpret the information in it for PubSub communication purposes. For example, a PubSub connection descriptor requires resolution when it has no (physical) network address specified, but it has a (logical) non-empty connection name. Similarly, a writer group descriptor requires resolution when its dataset writer Id is set to zero (i.e. "unknown"), but it has a (logical) non-empty dataset writer name, and so on.

A *physicalization* of a descriptor or filter is an operation that makes it strictly physical. A physicalization is only possible on descriptor or filters that currently do not require resolution. Physicalization is performed by setting the logical name of the object to an empty string, and physicalizing its constituent descriptors (if there are any).

PubSub Connection Descriptor

A PubSub connection descriptor combines together:

- Logical name of the connection (optional)
- Physical network address (optional)
- Connection transport parameters
- Transport profile URI

In order to specify a PubSub connection, you can set the PubSub connection name, or its network address, or both.

> The PubSub connection can be "redirected" so that it uses a packet capture file, instead of true connection to the message oriented middleware (network). This is useful in all kinds of troubleshooting and development tasks. Such redirection is performed by setting up the PubSub connection descriptor in a specific way. For more information, see **Using packet capture files with OPC UA PubSub (Section 14.3)**.

Writer Group Descriptor

A writer group descriptor combines together an optional logical writer group name with an optional writer group Id. That is, in order to specify a writer group, you can set the writer group name, or its Id, or both.

Dataset Writer Descriptor

A dataset writer descriptor combines together an optional logical dataset writer name with an optional dataset writer Id. That is, in order to specify a dataset writer, you can set the dataset writer name, or its Id, or both.

Subscribe Dataset Filter

The subscribe dataset filter is a kind of compound descriptor. It is documented separately in **Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)**.

4.13 Security

In this part of the documentation:

- **OPC UA Security (Client-Server) (Section 4.13.1)**

4.13.1 OPC UA Security (Client-Server)

OPC Unified Architecture is designed as secure solution from the ground up. The result is a solid, secure infrastructure that can be, at the same time, complex to understand, and sometimes difficult to deploy and configure properly.

Security in OPC UA is a broad subject that we cannot and will not cover in detail here. It is recommended that you study available materials. Here are some useful links that can get you started:

- Secure communication with IEC 62541 OPC UA: https://opcfoundation.org/wp-content/uploads/2014/05/OPC-

UA_Security_EN.pdf
- The OPC UA Security Model For Administrators: https://opcfoundation.org/wp-content/uploads/2014/05/OPC-UA_Security_Model_for_Administrators_V1.00.pdf

OPC Data Client-UA attempts to simplify the security configuration where possible, but a reasonable level of understanding OPC-UA security is still needed.

4.13.1.1 Providing OPC UA Client Instance Certificate

Similarly to server instance certificate checking described above, in a secured OPC-UA solution, the OPC client identifies itself to the OPC server using its own instance certificate. The OPC server authenticates the certificate and checks whether the communication with that OPC client is authorized. Typically, the process of checking other party's instance certificate is achieved through use of certificate trust lists, accessible programmatically and by means of UA Configuration Tool (installed with our product, and with OPC-UA downloads provided by OPC Foundation).

Since with OPC Data Client-UA you are developing an OPC-UA client application, this client application must therefore typically provide its instance certificate to the OPC servers it is connecting to. OPC Data Client-UA attempts to make this process easier by automatically selecting parameters for the certificate, and if the certificate does not exist yet, by creating it and storing into the appropriate certificate store on the computer. Depending on your deployment scenario, you may also create the certificate and put it into the certificate store by some other means, and in that case, you would simply let OPC Data Client-UA use that certificate.

In This Topic

C#

```
// This example demonstrates how to set the application name for the client certificate.

using System;
using OpcLabs.BaseLib.Instrumentation;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UAApplicationParameters
{
    class ApplicationName
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Hook static events
            EasyUAClient.LogEntry += EasyUAClientOnLogEntry;

            try
            {
                // Set the application name, which determines the subject of the client
certificate.
```

```
                // Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName =
                    "QuickOPC - CSharp example application";

                // Do something - invoke an OPC read, to trigger some loggable entries.
                var client = new EasyUAClient();
                try
                {
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                }

                // The certificate will be located or created in a directory similar to:
                // C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
                // and its subject will be as given by the application name.

                Console.WriteLine("Processing log entry events for 10 seconds...");
                System.Threading.Thread.Sleep(10 * 1000);

                Console.WriteLine("Done.");
            }
            finally
            {
                // Unhook static events
                EasyUAClient.LogEntry -= EasyUAClientOnLogEntry;
            }
        }

        // Event handler for the LogEntry event.
        // Print the loggable entry containing client certificate parameters.
        private static void EasyUAClientOnLogEntry(object sender, LogEntryEventArgs
logEntryEventArgs)
        {
            if (logEntryEventArgs.EventId == 161)
                Console.WriteLine(logEntryEventArgs);
        }
    }
}
```

Object Pascal

```
// This example demonstrates how to set the application name for the client certificate.

type
  TClientConfigurationEventHandlers103 = class
    procedure OnLogEntry(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _LogEntryEventArgs);
  end;

// Event handler for the LogEntry event.
```

```
// Print the loggable entry containing client certificate parameters.
procedure TClientConfigurationEventHandlers103.OnLogEntry(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _LogEntryEventArgs);
begin
  if eventArgs.EventId = 161 then
      WriteLn(eventArgs.ToString);
end;

class procedure ApplicationName.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  ClientConfigurationEventHandlers: TClientConfigurationEventHandlers103;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior - here, the
  // shared LogEntry event.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfigurationEventHandlers := TClientConfigurationEventHandlers103.Create;
  ClientConfiguration.OnLogEntry := ClientConfigurationEventHandlers.OnLogEntry;
  ClientConfiguration.Connect;

  try
    // Set the application name, which determines the subject of the client certificate.
    // Note that this only works once in each host process.

ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName
:=
      'QuickOPC - Delphi example application';

    // Do something - invoke an OPC read, to trigger some loggable entries.
    Client := CoEasyUAClient.Create;
    try
      Value := Client.ReadValue(
        'http://opcua.demo-this.com:51211/UA/SampleServer',
        'nsu=http://test.org/UA/Data/;i=10853');
    except
      on E: EOleException do
      begin
        WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      end;
    end;

    // The certificate will be located or created in a directory similar to:
    // C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
    // and its subject will be as given by the application name.

    WriteLn('Processing log entry events for 10 seconds...');
    PumpSleep(10*1000);

    WriteLn('Done.');
  finally
    FreeAndNil(ClientConfiguration);
    FreeAndNil(ClientConfigurationEventHandlers);
  end;
end;
```

PHP

```
// This example demonstrates how to set the application name for the client certificate.

class ClientConfigurationEvents {
    // Event handler for the LogEntry event.
    // Print the loggable entry containing client certificate parameters.
    function LogEntry($Sender, $E)
    {
        if ($E->EventId = 161)
            printf("%s\n", $E);
    }
}

// The configuration object allows access to static behavior - here, the
// shared LogEntry event.
$ClientConfiguration = new COM("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration");
$ClientConfigurationEvents = new ClientConfigurationEvents();
com_event_sink($ClientConfiguration, $ClientConfigurationEvents,
"DEasyUAClientConfigurationEvents");

// Set the application name, which determines the subject of the client certificate.
// Note that this only works once in each host process.
$ClientConfiguration->SharedParameters->EngineParameters->ApplicationParameters-
>ApplicationName =
    "QuickOPC - PHP example application";

// Do something - invoke an OPC read, to trigger some loggable entries.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}

// The certificate will be located or created in a directory similar to:
// C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
// and its subject will be as given by the application name.

printf("Processing log entry events for 10 seconds...");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Done.\n");
```

VB.NET

```
' This example demonstrates how to set the application name for the client certificate.

Imports OpcLabs.BaseLib.Instrumentation
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._UAApplicationParameters
    Friend Class ApplicationName
```

```
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Hook static events
            AddHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry

            Try
                ' Set the application name, which determins the subject of the client
certificate.
                ' Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName = _
                    "QuickOPC - VBNet example application"

                ' Do something - invoke an OPC read, to trigger some loggable entries.
                Dim client = New EasyUAClient()
                Try
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Exit Sub
                End Try

                ' The certificate will be located or created in a directory similar to:
                ' C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
                ' and its subject will be as given by the application name.

                Console.WriteLine("Processing log entry events for 10 seconds...")
                Threading.Thread.Sleep(10 * 1000)

                Console.WriteLine("Done.")
            Finally
                ' Unhook static events
                RemoveHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry
            End Try

        End Sub


        ' Event handler for the LogEntry event.
        ' Print the loggable entry containing client certificate parameters.
        Private Shared Sub EasyUAClientOnLogEntry(ByVal sender As Object, ByVal
logEntryEventArgs As LogEntryEventArgs)
            If (logEntryEventArgs.EventId = 161) Then
                Console.WriteLine(logEntryEventArgs)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example demonstrates how to set the application name for the client certificate.

Option Explicit

' The configuration object allows access to static behavior.
Dim ClientConfiguration: Set ClientConfiguration =
CreateObject("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration")
WScript.ConnectObject ClientConfiguration, "ClientConfiguration_"

' Set the application name, which determins the subject of the client certificate.
' Note that this only works once in each host process.
ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName
= "QuickOPC - VBScript example application"

' Do something - invoke an OPC read, to trigger some loggable entries.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' The certificate will be located or created in a directory similar to:
' C:\Users\All Users\OPC Foundation\CertificateStores\UA Applications\certs\
' and its subject will be as given by the application name.

WScript.Echo "Processing log entry events for 10 seconds..."
WScript.Sleep 10*1000



' Event handler for the LogEntry event.
' Print the loggable entry containing client certificate parameters.
Sub ClientConfiguration_LogEntry(Sender, e)
    If e.EventId = 161 Then WScript.Echo e
End Sub
```

The type and location of client instance certificate store can be controlled by properties in EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters, such as the ApplicationCertificateStore property. By default, OPC Data Client-UA uses a standard directory-based store defined by OPC Foundation ("MachineDefault").

The AllowClientCertificatePrompt property in EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters determines whether the component can interact with the user when checking or creating an application instance certificate. If set to false (the default), no dialogs will be displayed.

Certificate Auto-Creation

If you let OPC Data Client-UA create the certificate and add it to the store, it should be noted that it is a security-sensitive

operation that would typically be allowed only to administrators. OPC Data Client-UA will attempt to do this if the certificate is not found, which may be when the application is first started on the computer, but if your application is not running with privileges required for this operation, the certificate will not be added.

In order to allow your application be run under usual privileges, and make its behavior more predictable, it is recommended that you assure the generation of the certificate during the application deployment, when administrative privileges are usually required anyway. One way to do this is call the static method EasyUAClient.Install during the deployment process. For more information, see the "Application Deployment" chapter.

> ⚠ When using the EasyUAClient.Install method during the deployment process, make sure that the application (certificate) name is the same during deployment, and during the normal application. With automatically generated application name (which is the default), a different name might be generated if you use a separate application for the deployment, or if use you use utilities such as **InstallUtil.exe** which effectively act as a host for your application. In such situations, always set the application name explicitly, to the same value during the deployment and in normal application executions.

Peer Auto-Trust

When OPC Data Client creates or checks the client instance certificate (in the application certificate store), it also makes sure that the certificate is to be found in the trusted peers certificate store. If the client instance certificate store is not found there, a copy of it (without the private key) is placed into the trusted peers certificate store automatically.

With this *peer auto-trust* mechanism, OPC UA servers that use the same trusted peers certificate store can automatically trust the client application(s) created with OPC Data Client.

> ⚠ With the default settings effective when targeting .NET Standard, applications use separate certificate stores and not system-wide "shared" certificate stores. The peer auto-trust mechanism is still invoked under .NET Standard, but with the default settings it may have no observable effect, because other applications are using different certificate stores.

Additional information

OPC-UA requires that both parties (client and server) mutually identify themselves using application certificates. The certificates are supposed to be unique for each application (instance), and therefore cannot be a constant part of the "toolkit" (such as OPC Data Client), and need to be generated. In order to make this process invisible (in common cases) to the developer, OPC Data Client

a) determines the parameters of the certificate automatically, using values such as the EXE name or the calling assembly name,

b) attempts to look up the certificate in the certificate store,

c) if not found, it attempts to create it, and save it into the certificate store.

The algorithm described above does not, however fit well with hosted environments such as IIS, for two reasons:

- It is difficult to automatically determine reasonable and unique parameters for the certificate, because the hosting process is the IIS service, not "your" own application EXE.
- The page processing code in IIS typically runs with low privileges that do not allow it to call the necessary CertificateGenerator utility, and even less to save the new certificate to the store.

For usage in such environments, the recommendation is:

- Pre-generate the application certificate manually, and save it to the certificate store. This will remove the need for the application to create and save the certificate.
- In your application, set your own parameters of the certificate. The application will then use these parameters to look up the certificate.

On Windows, application certificate can be generated using **Opc.Ua.CertificateGenerator.exe** utility, available from the OPC Foundation. The utility is typically located at **"C:\Program Files (x86)\Common Files\OPC Foundation\UA\v1.0\Bin"**. Run it with "/?" to obtain usage instructions.

> OPC Data Client normally does not need the Opc.Ua.CertificateGenerator utility be present on the disk. For COM and .NET Framework target platforms, it has its own internal copy of this utility which it loads and executes. For .NET Standard development platform, the Opc.Ua.CertificateGenerator utility is not used.

Setting the certificate parameters is done by modifying properties in EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters (EasyUAClient.SharedParameteres is a static property, and you need to set the values before creating the first instance EasyUAClient, in order for it to have the desired effect). The properties of interest are:

- ApplicationCertificateSubject
- ApplicationName
- ApplicationUriString
- ProductUriString

The values should match those used when the certificate was generated, because they (or at least some of them) are used to look up the certificate in the store.

The default certificate store for OPC-UA applications is not one of the Windows stores, but instead, a file-system (directory) based store. It defaults to **"%CommonApplicationData%\OPC Foundation\CertificateStores\UA Applications"**, where **%CommonApplicationData%** is e.g. **"C:\ProgramData"**. The best way to inspect the UA certificates (on Windows) is to use the OPC UA Configuration tool from the OPC Foundation. After installation, run (e.g.) Start -> All Programs -> OPC Foundation -> UA SDK 1.01 -> UA Configuration Tool. Then, select the "**Manage Certificates**" tab, press the drop-down next to "**Store Path**", select the store I listed above, and press the View Certificates button.

> OPC Data Client also includes the **UA Configuration Tool** in the Bonus Material part of its full installation for Windows. You can access the **UA Configuration Tool** from the **Start** menu (under OPC Data Client program group), or using the OPC Data Client **Launcher** application.

You should not use the same certificate for multiple applications. And, you should not use the same certificate even for multiple instances of the same application (such as when you deploy the application on multiple computers). Each instance of the application (typically, this means each application/computer combination, but can be even more) should have its own, unique certificate. This is how UA security is meant to work, because there needs to be a way for the server to authenticate each client connection separately and possibly deny the connection to unsecure or compromised clients. Technically, if you violate this and use the same certificate, things will probably work OK on the surface, but it is against the OPC-UA security specs. This is the reason why the certificate is generated anew on the computer - and the cause of some troubles.

Actually, while using the CertificateGenerator manually is possible, it would be probably too complicated to try to assemble the right set of parameters so that the generated certificate is then properly found by the client application. Instead, we suggest that you create a small app (e.g. Windows Forms app, or a console app). In it, you set the 4 parameters in EasyUAClient.SharedParameters to meaningful values, and then call a static method EasyUAClient.Install(). You then run this application with administrator/elevated privileges, and it will create the certificate (verify it with the UA Configuration Tool described above). In your actual client app, set the parameters in precisely the same way before instantiating the EasyUAClient object, and doing so should assure that it will find and use the certificate. Still, your app (which may be a Web app) needs to have read access to the directory of the cert store - you may have to modify the permissions). The additional

app you created may become part of your installation procedure.

Obtaining Instance Certificate from the GDS

If your OPC UA setup includes an OPC UA Global Registration Server (GDS), you might be able to obtain the application instance certificate from the GDS (in the role of Certificate Manager, CM). For more information, see **OPC UA Client Application Service (Section 6.3.1)** (or, less preferrably, **OPC UA Certificate Management Client (Section 6.4.2)**).

Troubleshooting

Some common problem scenarios include:

Certificate generation works on one computer, does not on other. For example, when you first run your application your development computer, you may use the development Web server built-in in Visual Studio. This one uses different accounts and permissions from IIS, and the certificate creates just fine. On the production computer, IIS is used, and the certificate generation may fail. See further below for related information.

Certificate generation fails under IIS. Most likely this is due to permissions. IIS settings are VERY restrictive (for good reason), and the defaults won't allow the certification generation or storing it into the store.

4.13.1.2 Trusting OPC UA Server Instance Certificate

In a secured OPC-UA solution, the OPC server identifies itself to the OPC client using its instance certificate. The OPC client authenticates the certificate and checks whether the communication with that OPC server is authorized. Typically, the process of checking other party's instance certificate is achieved through use of certificate trust lists, accessible programmatically and by means of UA Configuration Tool (installed with our product, and with OPC-UA downloads provided by OPC Foundation).

In This Topic

In OPC Data Client-UA, you can control how server instance certificates are trusted using Certificate Acceptance Policy (UACertificateAcceptancePolicy object) that you can parameterize in the EngineParameters property on the EasyUAClient object.

By default:

- Certificate will be trusted if issued by one of certification authorities found in a "Trusted Issuers" certificate store, while default to a directory-based store at `%CommonApplicationData%\OPC Foundation\CertificateStores\UA Certificate Authorities`.
- Certificate will be trusted if found in a "Trusted Peers" certificate store, which defaults to a directory-based store at `%CommonApplicationData%\OPC Foundation\CertificateStores\UA Applications`.
- Certificate not validated by the above rules will be copied to a "Rejected" certificate store, which defaults to a directory-based store at `%CommonApplicationData%\OPC Foundation\CertificateStores\RejectedCertificates`.
- Connections to some of pre-defined trusted endpoint URL strings are always accepted. The trusted endpoint URL list contains endpoints of OPC Data Client public demo servers. Such connections should not be considered secure!
- If the application is running in user interactive mode, the user is presented with a notification allowing him/her to accept or reject the certificate.

Certificates issued by a certification authority (CA) must contain a verifiable certificate chain leading to the CA. **OPC Data Client requires that the root (CA) certificate must be included in the chain as well.**

As you can see from the rules described above, a certificate that is not initially trusted by the client will be copied to the "Rejected" certificate store. If you know that the server certificate should be trusted, you can locate this rejected certificate and manually move it to the "Trusted Peers" certificate store, which will make it trusted next time it is provided by the server to the client.

Trust Settings

The TrustedEndpointUrlStrings property contains an array of endpoint URLs that are always trusted, without regard for the certificate provided. By default, this list contains endpoint URLs of the demo OPC UA Sample Server on the Internet (opcua.demo-this.com) and the local OPC UA Sample Server, meaning that you will always be able to use the sample servers from OPC Data Client-UA, even if configuration of its instance certificate, or configuration of trusted certificate list for OPC Data Client-UA has done been done properly. In a fully secure configuration, this list should be empty.

The AcceptAnyCertificate property determines whether the client accepts any server certificate, even if a certificate validation error occurs. Setting this property to true effectively bypasses an important security feature in OPC Unified Architecture. Use it only for testing and development purposes, or if your application does not require the server certificate check.

The type and location of trusted issuers certificate store can be controlled by EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedIssuersCertificateStore property. By default, OPC Data Client-UA uses a standard directory-based store defined by OPC Foundation ("UA Certificate Authorities").

The type and location of trusted peers certificate store can be controlled by EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedPeersCertificateStore property. By default, OPC Data Client-UA uses a standard directory-based store defined by OPC Foundation ("UA Applications").

The path to the rejected certificate store can be controlled by EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.RejectedCertificateStore property. By default, OPC Data Client-UA uses a standard directory-based store defined by OPC Foundation ("RejectedCertificates").

The AllowUserAcceptCertificate property determines whether the interactive user can be prompted to and accept a server certificate that has failed other validation checks. This setting has effect only when the current process is running in user interactive mode.

Some finer details regarding the server instance checking process can be controlled by a configuration file, if you correspondingly configure the EasyUAClient.SharedParameters.EngineParameters.ConfigurationSources. For more information, see "Application Configuration" chapter under "Advanced Topics".

Refreshing Trust Lists from the GDS

If your OPC UA setup includes an OPC UA Global Registration Server (GDS), you might be able to acquire the trust lists from the GDS (in the role of Certificate Manager, CM) and refresh the certificate stores accordingly. For more information, see **OPC UA Client Application Service (Section 6.3.1)** (or, less preferrably, **OPC UA Certificate Management Client (Section 6.4.2)**).

4.13.1.3 Trusting OPC UA Server HTTPS Certificate

> This article only applies to OPC UA connections made using the HTTPS protocol. You can skip this article if you are not using HTTPS.

An HTTPS connection requires a certificate on (placed on the OPC UA server side) to be used for performing the message encryption. This certificate is different than the application instance certificate of the OPC UA server. The OPC UA client must validate and trust the server's HTTPS certificate for the connection to work.

If you are accustomed to using HTTPS in the Web browser, you may know that the browser uses machine-wide certificate stores to determine whether the HTTPS certificate of the Web server you are connecting to can be trusted. Typically, the HTTPS certificate is issued by a well-known CA (Certification Authority), and certificates of such authorities come with the browser, and are maintained by the operating system (updates) or on an enterprise IT level.

When an HTTPS server certificate is trusted by the machine-wide mechanism, OPC UA client applications created with OPC Data Client will trust it for HTTPS connections to OPC UA servers as well. One option to establish trust for HTTPS server certificates is therefore to use the same mechanism and procedures as for Web browsing. This approach works, but has the disadvantage that it usually requires administrative privileges to manipulate the Internet certificate stores (normally used for Web browsing).

OPC Data Client allows you to use additional rules with server HTTPS certificates. By default, OPC UA client applications created with OPC Data Client will also trust server HTTPS certificates if they are allowed by the rules for server instance certificates, as described in **Trusting OPC UA Server Instance Certificate (Section 4.13.1.2)**. This means that by default:

- Certificate will be trusted if issued by one of certification authorities found in a directory-based store at `%CommonApplicationData%\OPC Foundation\CertificateStores\UA Certificate Authorities`.
- Certificate will be trusted if found in a directory-based store at `%CommonApplicationData%\OPC Foundation\CertificateStores\UA Applications`.
- Certificate not validated by the above rules will be copied to a directory-based store at `%CommonApplicationData%\OPC Foundation\CertificateStores\RejectedCertificates`.
- Connections to some of pre-defined trusted endpoint URL strings are always accepted. The trusted endpoint URL list contains endpoints of OPC Data Client public demo servers. Such connections should not be considered secure!
- If the application is running in user interactive mode, the user is presented with a notification allowing him/her to accept or reject the certificate.

> Certificates issued by a certification authority (CA) must contain a verifiable certificate chain leading to the CA. **OPC Data Client requires that the root (CA) certificate must be included in the chain as well.**

> For HTTPS server certificates, only the scheme (protocol) and authority (host and port) part of the endpoint URL are used for comparisons. This is because on the server side, the certificate is bound to all communications running on a specific port.

In the default state, the HttpsCertificateAcceptancePolicy Property of the UAClientEngineParameters Class contains a null reference. In this state, when you change the policy used for trusting server instance certificates, such changes will also automatically apply to trusting server HTTPS certificates.

You can also specify a different certificate acceptance policy for HTTPS server certificates. To do so, create your own instance of the UACertificateAcceptancePolicy Class, set its properties as needed, and assign it to the HttpsCertificateAcceptancePolicy Property. Note that as with any other static properties of the EasyUAClient Class, this has to be done before any OPC UA operations are invoked.

4.13.1.4 OPC UA Certificate Stores

Certificate Store Types

OPC Data Client can work with certificates located in *certificate stores*. Certificate stores are of different types; they are either:

- **platform-specific** certificate store (Windows, Linux, ...), or
- **directory** in a file system.

For more details on each of these types, see **OPC UA Platform-specific Certificate Stores (Section 4.13.1.4.2)** and **OPC UA Directory Certificate Stores (Section 4.13.1.4.1)**.

In This Topic

Certificate Store Path

The certificate store is identified by a string, called *certificate store path*. The syntax and semantics of this string is as follows:

- If the string starts with **"LocalMachine\"** (case insensitive), it denotes a platform-specific certificate store for the local computer. The store name follows this prefix.
- If the string starts with **"CurrentUser\"** (case insensitive), it denotes a platform-specific certificate store for the current user. The store name follows this prefix.
- Otherwise, the string denotes a certificate store located in a file system directory, and the value is equal to the directory path. The string can contain replaceable *tokens* that refer to specific locations. For details, see **OPC UA Directory Certificate Stores (Section 4.13.1.4.1)**.

Certificate Store Locations

OPC Data Client uses several certificate stores for its operations. The location of the stores is given by various parameters. The stores are:

- **Application certificate store**. A client application created with OPC Data Client looks for or creates its own instance certificate here. The location of the store is controlled by EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore property. For more details, see **Providing OPC UA Client Instance Certificate (Section 4.13.1.1)**. The application instance certificates in this store include the private keys.
- **Rejected certificate store**. Server certificates that fail validation are placed into this store. Normally the certificates in this store do not include the private key, but in some cases they may (when the validation of application's own instance certificate fails - as opposed to validation of certificate belonging to the other communication party).
- **Trusted issuers certificate store**. Contains certificates of Certification Authorities to be trusted.
- **Trusted peers certificate store**. Contains specific OPC UA application instance certificates to be trusted.

For details on rejected certificate store, trusted issuers certificate store and trusted peers certificate store, see **Trusting OPC UA Server Instance Certificate (Section 4.13.1.2)** and **Trusting OPC UA Server HTTPS Certificate (Section 4.13.1.3)**.

Default Settings

When targeting .NET Framework, all certificates that OPC Data Client works with are located in some "shared" directory-

based certificate store by default. Specifically:

- Default application certificate store is **"%CommonApplicationData%\OPC Foundation\CertificateStores\MachineDefault"**.
- Default rejected certificate store is **"%CommonApplicationData%\OPC Foundation\CertificateStores\RejectedCertificates"**.
- Default trusted issuers certificate store is **"%CommonApplicationData%\OPC Foundation\CertificateStores\UA Certificate Authorities"**.
- Default trusted peers certificate store is **"%CommonApplicationData%\OPC Foundation\CertificateStores\UA Applications"**.

On Windows, the **%CommonApplicationData%** token typically resolves to something like **"C:\ProgramData"** (note that this folder is hidden by default). See **OPC UA Directory Certificate Stores (Section 4.13.1.4.1)** for more details.

When targeting .NET Standard, all certificates that OPC Data Client works with are located in a directory-based certificate store under the current working directory by default. Specifically:

- Default application certificate store is **"%LocalFolder%/OPC Foundation/CertificateStores/MachineDefault"**.
- Default rejected certificate store is **"%LocalFolder%/OPC Foundation/CertificateStores/RejectedCertificates"**.
- Default trusted issuers certificate store is **"%LocalFolder%/OPC Foundation/CertificateStores/UA Certificate Authorities"**.
- Default trusted peers certificate store is **"%LocalFolder%/OPC Foundation/CertificateStores/UA Applications"**.

See **OPC UA Directory Certificate Stores (Section 4.13.1.4.1)** for explanation of the **"%LocalFolder"%** token, and more details.

> Other commonly used setting for application certificate store location under .NET Standard is **"CurrentUser\UA_MachineDefault"**.

Certificate Store Security

In a secure deployment, the certificate store itself (that is, read and write access to the certificates it contains) must be secured. OPC Data Client needs appropriate permissions to read from (and sometimes write to) the certificate stores. At the same time, access should be denied to unauthorized actors. This is most critical for the write access to the stores (and for read access to the private key parts). Securing the certificate stores is outside of OPC Data Client scope.

> OPC Data Client does not use passwords to access certificate private keys, and the certificates it creates are not protected by passwords. Be aware that the private keys in OPC UA certificate stores for OPC Data Client usage are not protected by passwords.

4.13.1.4.1 OPC UA Directory Certificate Stores

Structure of the Directory Certificate Store

In This Topic

As described in **OPC UA Certificate Stores (Section 4.13.1.4)**, *directory certificate stores* reside in a file system. In many cases, their structure can remain an opaque implementation detail to you. Knowing the structure comes handy if you need to deal with them externally or manually.

There are no files at the "root" of the directory certificate, only sub-directories, and they are:

- The **"certs"** subdirectory. Contains the certificates without private keys, in .DER format.
- The **"crl"** subdirectory (only in certificate stores which need it). Contains the certificate revocation lists (CRLs) for the certificate store.
- The **"private"** subdirectory (only in certificate stores which need it). Contains the certificate *with* private keys, in .PFX format. Instead of .PFX, it can also contain just the private key, in .PEM format (OPC Data Client recognizes this but does not itself create files in directory certificate stores in this way).

If the certificate store contains a certificate with a private key, it will therefore be present in both **"certs"** and **"private"** subdirectories, each time in a different format.

The file name of each certificate comprises of the certificate common name (from its subject), followed by a space, and the certificate thumbprint in square brackets, e.g. **"My Application [71511464F08D30AF9F9B2BC21CDB78D49BE568B8]"**. Special characters (like **< > : " / \ | ? ***) that cannot appear in file names are stripped off from the common name.

> OPC Foundation has a **UA Configuration Tool** which can be used to manage the certificates related to OPC UA on Windows machines (both in the directory certificate stores, and in Windows certificate stores). OPC Data Client includes this tool in the Bonus Material part of its full installation for Windows. You can access the **UA Configuration Tool** from the **Start** menu (under OPC Data Client program group), or using the OPC Data Client **Launcher** application.

Syntax of Directory Certificate Store Path

In its simplest form, the directory certificate store path is an absolute path to the directory that holds the store, e.g. **"C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault"**. Since absolute paths are not flexible enough and may fail to work when the application is transferred to a computer with different configuration, OPC Data Client allows you to include replaceable *tokens* in the directory certificate store path. A token is enclosed in percentage sign characters, e.g. **%CommonApplicationData%**. For the token to be recognized, the directory certificate store path must begin directly with the token (the percent sign). Tokens are not recognized further down in the string.

The following two types of tokens can be used. OPC Data Client searches for them from top to bottom.

- **Special folders**. Any enumerated constant name from the .NET Environment.SpecialFolder enumeration (https://docs.microsoft.com/en-us/dotnet/api/system.environment.specialfolder) can be used as a token. The token is replaced by the absolute path of the corresponding special folder. Not all available special folders make sense with OPC UA, but the following ones are good candidates: CommonApplicationData, CommonProgramFiles, ProgramFiles, etc.
- **Environment variables**. These are the operating system environment variable names, as available to the current process. The value of the environment variable is retrieved, and replaces the token.
- On .NET Standard development platform only: "**%LocalFolder%**" (case sensitive). The current working directory of the application replaces the token. The string does not end with a backslash (\) or a slash (/).

If the token name is not recognized, the token is replaced by an empty string (i.e. the token is removed).

Platform Differences

The default certificate store paths have very different defaults in .NET Framework and .NET Standard.

In .NET Framework, the default certificate stores paths are "shared"; that is, by default, all applications developed with OPC Data Client share the same certificate stores, which are system-wide.

In .NET Standard, the default certificate stores paths are all in directories that are located under the current working directory of the application.

In .NET Framework

When you target .NET Framework with OPC Data Client, the default directory certificate store paths all start with the **%CommonApplicationData%** token, which on Windows typically resolves to something like **"C:\ProgramData"** (this is independent of whether your development target is .NET Framework or .NET Standard). Note: On Linux, the **%CommonApplicationData%** token typically resolves to **"/usr/share"**, but that would only be of interest in .NET Standard (see below), if you change the default store paths values.

> ⚠ In a default Windows installation, the "**C:\ProgramData**" directory exists, but it is hidden. You need to use appropriate setting or tool in order to be able to inspect the contents of hidden directories.

Example: The default application certificate store path is specified as **"%CommonApplicationData%\OPC Foundation\CertificateStores\MachineDefault"**, and on Windows, it may resolve to **"C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault"** (note that the "C:\ProgramData" folder is hidden).

In .NET Standard

When you target .NET Standard with OPC Data Client, the default directory certificate store paths all start with the **%LocalFolder%** token, which on resolves to the current working directory of the application.

Example: The default application certificate store path is specified as **"%LocalFolder%/OPC Foundation/CertificateStores/MachineDefault"**. You will therefore have an **"OPC Foundation"** sub-directory in your current working directory, and a structrie of the certificate stores below it.

> ⚠ Make sure you secure the directory certificate stores properly. This caution is particularly relevant with the default .NET Standard settings, because you may end up with multiple certificate stores (directories) to be secured, not just the common "shared" one. Also note that the absence of the "shared" and commonly used certificate stores prevents certain inter-application cooperation scenarios from working, such as the practice that the OPC UA application stores its own application certificate to the trusted peers certificate store on the same machine, to make it easier for local connections to be established "out of the box".

4.13.1.4.2 OPC UA Platform-specific Certificate Stores

The *platform-specific certificate stores* are implemented and maintained by the operating system or the runtime. As explained in **OPC UA Certificate Stores (Section 4.13.1.4)**, you specify the platform-specific certificate store by starting the certificate store path by either **"LocalMachine\"** or **"CurrentUser\"**.

- If the string starts with **"LocalMachine\"** (case insensitive), it denotes a platform-specific certificate store for the local computer. Commonly used examples are: **"LocalMachine\My"**,

In This Topic

"LocalMachine\UA Applications" or "**LocalMachine\UA Certificate Authorities**".

- If the string starts with **"CurrentUser\"** (case insensitive), it denotes a platform-specific certificate store for the current user. Commonly used examples are: **"CurrentUser\My"** or **"CurrentUser\Root"**.

The store name follows the prefix.

> ⚠ Some older code or documentation might use the term "Windows certificate store" for certificate stores that can, in fact, now be implemented also on other platforms, such as Linux. This is due to the Windows origins of such code or documentation. As OPC Data Client now supports multiple development platforms and operating systems, in new documents we consistently use the term "platform-specific certificate store" wherever we refer to a general platform-provided certificate store concept. In new documents, we use the term "Windows certificate store" only to refer to a specific implementation of platform-specific certificate store on Windows operating system. Similarly, we would use "Linux certificate store" to refer to a platform-specific certificate store in a way that is implemented in Linux (which may differ by the particular .NET runtime, e.g. .NET Core).

C#

```
// This example demonstrates how to place the client certificate in the platform-specific (Windows,
Linux, ...) certificate
// store.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UAApplicationParameters
{
    class ApplicationCertificateStore
    {
        public static void PlatformSpecific()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Set the application certificate store path, which determines the location of the client
certificate.
            // Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore =
"CurrentUser\\My";

            // Do something - invoke an OPC read, to trigger creation of the certificate.
            var client = new EasyUAClient();
            try
            {
                client.ReadValue(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
            }

            // The certificate will be located or created in the specified platform-specific
certificate store.
            // On Windows, when viewed by the certmgr.msc tool, it will be under
            // Certificates - Current User -> Personal -> Certificates.

            Console.WriteLine("Done.");
        }
    }
}
```

Object Pascal

```
// This example demonstrates how to place the client certificate
// in the platform-specific (Windows, Linux, ...) certificate store.

class procedure ApplicationCertificateStore.PlatformSpecific;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Set the application certificate store path, which determines the location of the client
certificate.
  // Note that this only works once in each host process.

ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore
:=
    'CurrentUser\My';

  // Do something - invoke an OPC read, to trigger creation of the certificate.
  Client := CoEasyUAClient.Create;
  try
    Value := Client.ReadValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // The certificate will be located or created in the specified platform-specific certificate store.
  // On Windows, when viewed by the certmgr.msc tool, it will be under
  // Certificates - Current User -> Personal -> Certificates.

  WriteLn('Done.');

  FreeAndNil(ClientConfiguration);
end;
```

PHP

```
// This example demonstrates how to place the client certificate
// in the platform-specific (Windows, Linux, ...) certificate store.


// The configuration object allows access to static behavior.
$ClientConfiguration = new COM("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration");

// Set the application certificate store path, which determines the location of the client
certificate.
// Note that this only works once in each host process.
$ClientConfiguration->SharedParameters->EngineParameters->ApplicationParameters-
>ApplicationCertificateStore =
    "CurrentUser\My";

// Do something - invoke an OPC read, to trigger creation of the certificate.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
try
{
```

```
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}

// The certificate will be located or created in the specified platform-specific certificate store.
// On Windows, when viewed by the certmgr.msc tool, it will be under
// Certificates - Current User -> Personal -> Certificates.

printf("Done.\n");
```

VB.NET

```
' This example demonstrates how to place the client certificate in the platform-specific (Windows,
Linux, ...) certificate store.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._UAApplicationParameters
    Friend Class ApplicationCertificateStore
        Public Shared Sub PlatformSpecific()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Set the application certificate store path, which determines the location of the client
certificate.
            ' Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore =
"CurrentUser\\My"

            ' Do something - invoke an OPC read, to trigger creation of the certificate.
            Dim client = New EasyUAClient()
            Try
                client.ReadValue(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
            End Try

            ' The certificate will be located or created in the specified platform-specific
certificate store.
            ' On Windows, when viewed by the certmgr.msc tool, it will be under
            ' Certificates - Current User -> Personal -> Certificates.

            Console.WriteLine("Done.")
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example demonstrates how to place the client certificate in the platform-specific (Windows,
Linux, ...) certificate
Rem store.
Rem Note: COM is only available on Windows.

Option Explicit
```

```
' The configuration object allows access to static behavior.
Dim ClientConfiguration: Set ClientConfiguration =
CreateObject("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration")
WScript.ConnectObject ClientConfiguration, "ClientConfiguration_"

' Set the application certificate store path, which determines the location of the client certificate.
' Note that this only works once in each host process.
ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore
= "CurrentUser\My"

' Do something - invoke an OPC read, to trigger some loggable entries.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' The certificate will be located or created in the specified platform-specific certificate store.
' On Windows, when viewed by the certmgr.msc tool, it will be under
' Certificates - Current User -> Personal -> Certificates.

WScript.Echo "Done."
```

Windows Certificate Stores

Windows has a support for certificate stores built into the operating system, and corresponding APIs and tools to access the certificate stores. On Windows, OPC Data Client simply uses the mechanisms provided by Windows to support platform-specific certificate stores. For more information about Windows certificate stores, see e.g. Managing Certificates with Certificate Stores and How to Use the Certificates Console.

To manage the local computer certificates on Windows, type ***certlm.msc*** into the Windows search box, and press ***Enter***. You will need administrative privileges to manage the local computer certificates.

To manage the certificates for the current user on Windows, type ***certmgr.msc*** into the Windows search box, and press ***Enter***.

Note, however, that the logical store names displayed by the management console are not the same as the physical certificate store names, and that some stores may not be displayed at all.

OPC Foundation has a **UA Configuration Tool** which can be used to manage the certificates related to OPC UA on Windows machines (both in the directory certificate stores, and in Windows certificate stores). OPC Data Client includes this tool in the Bonus Material part of its full installation for Windows. You can access the **UA Configuration Tool** from the **Start** menu (under OPC Data Client program group), or using the OPC Data Client **Launcher** application.

Linux Certificate Stores

On Linux under .NET Core, the platform-specific certificate stores are implemented as follow:

- The certificates for the local computer are stored according to the rules valid on the particular Linux distro.
- The certificates for the current user are stored using an internal .NET Core mechanism (which currently appears to be a dedicated directory under ~/.dotnet, but that is considered an implementation details that may change in future versions).

For more information, see e.g. Provide a way for sysadmins to manage the .Net Core "My" certificate store on non-Windows platforms .

4.13.1.5 Security in OPC UA Endpoint Selection

As explained earlier, a single OPC-UA server may have multiple endpoints with the same functionality, but using different protocols, or different security settings (for example, an endpoint that provide message security may exist alongside with endpoint that does not provide message security). When OPC Data Client is given an endpoint URL, it does not yet (in general case) precisely determine the actual OPC UA server endpoint that the client wil lbe connecting to.

In order to influence which endpoint will get used, you can configure an OPC UA Endpoint Selection Policy. The policy can specifies various aspects of the endpoint you wish to connect to. Some of them are not related to security (for example, data encoding). Many aspects of the endpoint selection policy have, however, direct effect on the resulting security of the communication between the OPC UA client and the OPC UA server.

See **OPC UA Server Endpoints (Section 4.11.1)** for information about how to set the endpoint selection policy either globally, or just for a single endpoint descriptor. See **OPC UA Endpoint Selection Policy (Section 4.11.1.1)** for information about the contents of the policy, and meaning of its various parameters.

5 Development Models

OPC Data Client offers several development models. Each development model is a kind of approach to solving the problem. Some development models involve imperative programming (traditional coding), while others do not require programming in the usual sense, or the programming approach is dubstantially different. The various development models are described in the chapters further below.

The following table shows which development models are available for different development platforms:

	COM	**.NET Framework**	**.NET Standard**	**Excel** (with **Excel Option (Section 12.1)**)
Imperative Programming Model	✓	✓	✓	× (1)
Live Binding Model (Windows Forms, WPF)	×	✓	×	×
Live Mapping Model	×	✓	✓	×
Reactive Programming Model	×	✓	✓	×
Real-time Data Spreadsheet	×	×	×	✓

Notes: (1) This functionality is not directly available on the development platform, but can be achieved by combining with VBA code (through COM platform).

Multiple different development models can be freely mixed in the same project.

In OPC Data Client-COM, only the imperative programming model is available.

The User Interface object belong logically under the Imperative Programming Model, although they are described separately, because they are simply an extension for a specific purpose, and not all developers need to use them.

5.1 Imperative Programming Model

In the imperative programming development model, you write code that calls methods on OPC Data Client objects. You prepare and pass in the arguments necessary, and get back the results that you can process further.

This is the traditional model, “natural” to software developers. Its advantage is that you have high level of control over what is happening, and you can structure the code the way you like it. The disadvantage is that you may end up writing more code than it is necessary, compared to other models.

5.1.1 Operational Methods Overview

"Operational methods" are methods that communicate with the target environment (OPC servers etc.), directly or indirectly. Auxiliary methods, which may also exist on the objects, are not listed here.

Many methods exist in two forms:

- Single-operation.
- Multiple-operations.

In OPC, operations are much more effective if performed together in a group, at once. For example, doing a read of 100 values is much faster if performed at once, using the multiple-operation method, instead of repeatedly calling the single-operation method in a loop. For this reason, you should use the multiple-operation methods whenever possible. An exception to this rule are methods that act as substitute for event handlers (these methods are consistently named **Pull*XXXX***), where the recommendation is the opposite - you should normally use the single-operation method form instead.

In This Topic

Operational method naming conventions

The name of the operational method usually consists of a verb that denotes what the method does, and sometimes a noun which describes either the affected entity (such as in "ReadItem", or the outcome of the method (such as in "BrowseObjects").

Usually, the name of the multi-operation method is derived from the name of the single-operation method by using a plural instead of singular form of the noun (if it is present), and preceding the noun it with "Multiple" (or adding it at the end, when the noun is missing). For example, "ReadItem" becomes "ReadMultipleItems", and "Write" becomes "WriteMultiple". Such methods use an array of argument objects (one for each operation) as an input, and return an array of results (one for each operation again); the indexes in the output array correspond to the indexes in the input array.

When an input or output array is replaced by a list in the method signature, a word "List" appears at the end of the method name. This happens with all multiple-operation methods (where the word "Multiple" is removed then), and also with some single-operation methods where the argument or result happens to an array already.

> ⚠ Conceptually, the methods with "List" in their name are performing the same operations as those without it. Except where it is specifically needed, this documentation therefore does not refer to these methods explicitly. The name of the base method is used instead. For example, when this document talks about the "ReadMultipleValue" method, the method "ReadValueList" is also implied.

Legend to the tables

This chapter explains some of the notation used in the tables that follow:

- **bold**
- *italics*
- grayed out

When two or more methods appear in the table cell on the same line, separated by a comma, they perform precisely the same function, but differ only in the form of arguments accepted or results returned. Typically, this is with method that work with lists, that ar in addition to methods that work with arrays.

When multiple lines with different methods are listed inside a single table cell, then other methods with slightly different purpose are all derived (by extension) from the primary method listed in **bold**. This grouping helps you to understand the internal relations of the methods, where only a handful of primary methods is used to provide richer functionality, by means of method overloads, or methods with different names. The primary method is always the most general one and can always be used. It is, however, often more convenient to use the derived methods, which are more specialized, and give shorter and more understandable code.

Methods listed in the tables here are considered (unless stated otherwise), for the purpose of the explanation in this document to be methods of the objects listed even though (in .NET) in reality, they may be implemented as extension methods on the object or interface. They are also generally available under COM development platform, on the respective COM interfaces. There are other sets of extension methods, for .NET only, which further enhance the functionality of listed objects. These extension methods are not listed here, but rather under **Layered Extensions for .NET (Section 7.1)**.

Methods that are only available on the COM development platform are listed in *italics*. They are currently methods that accept or return lists instead of arrays, which is a necessity for some COM tools or languages that lack a sufficient support for COM safe arrays.

Methods that primarily exist for internal implementation (infrastructure) reasons are grayed out, and you should not normally used them.

EasyDAClient operation methods

This object is for OPC Data Access (OPC "Classic", COM/DCOM-based) and for OPC XML-DA.

Purpose	Single-operation Method	Multiple-operations Method
Read a named item (value/timestamp/quality).	ReadItem	**ReadMultipleItems**, *ReadItemList*
Read a named item value.	ReadItemValue	**ReadMultipleItemValues**, *ReadItemValueList*
Get value of OPC property.	GetPropertyValue	**GetMultiplePropertyValues**, *GetPropertyValueList*
Write a named item (value/timestamp/quality).	WriteItem	**WriteMultipleItems**, *WriteItemList*
Write value into a named item.	WriteItemValue	**WriteMultipleItemValues**, *WriteItemValueList*
Browse the computer for OPC (DA) servers.	**BrowseServers**	
Browse for child nodes. Browse for child branches. Browse for child leaves.	**BrowseNodes** BrowseBranches BrowseLeaves	
Browse for access paths.	**BrowseAccessPaths**, *BrowseAccessPathList*	
Browse for available item properties.	**BrowseProperties**	
Subscribe to changes of an OPC item.	SubscribeItem	**SubscribeMultipleItems**, *SubscribeItemList*

Change parameters of a subscription to an OPC item.	ChangeItemSubscription	**ChangeMultipleItemSubscriptions**, *ChangeItemSubscriptionList*
Unsubscribe from changes of an OPC item.	UnsubscribeItem	**UnsubscribeMultipleItems**, *UnsubscribeItemList*
Unsubscribe from changes of all OPC items.	**UnsubscribeAllItems**	
Attempt to pull an OPC item change.	**PullItemChanged**	**PullMultipleItemChanges**, *PullItemChangeList*

EasyAEClient operation methods

This objects is for OPC Alarms and Events (OPC "Classic", COM/DCOM-based).

Purpose	Single-operation Method	Multiple-operations Method
Get the current state information for a condition instance.	**GetConditionState**, *GetConditionStateList*	
Acknowledge a condition in the event server.	**AcknowledgeCondition**	
Browse the computer for OPC (A&E) servers.	**BrowseServers**	
Browse for child nodes. Browse for event areas. Browse for event sources.	**BrowseNodes** BrowseAreas BrowseSources	
Find event categories supported by the server. Find event conditions supported by an event category. Find attributes associated with an event category.	**QueryEventCategories** QueryCategoryConditions QueryCategoryAttributes	
Find event conditions associated with the source.	**QuerySourceConditions**	
Subscribe to OPC events.	**SubscribeEvents**	
Change parameters of an event subscription.	**ChangeEventSubscription**	
Force a refresh of active and inactive unacknowledged conditions.	**RefreshEventSubscription**	
Unsubscribe from particular OPC events.	**UnsubscribeEvents**	
Unsubscribe from changes of all OPC events.	**UnsubscribeAllEvents**	
Attempt to pull an OPC event.	**PullNotification**	PullMultipleNotifications, PullNotificationList

EasyUAClient operation methods

This object is for OPC Unified Architecture (OPC UA).

Purpose	Single-operation Method	Multiple-operations Method
Browse nodes in server's address space. Browse data nodes. Browse data variables. Browse event sources. Browse OPC methods. Browse notifiers. Browse OPC objects. Browse OPC object types. Browse OPC properties. Browse OPC variables.	Browse BrowseDataNodes BrowseDataVariables BrowseEventSources BrowseMethods BrowseNotifiers BrowseObjects BrowseObjectTypes BrowseProperties BrowseVariables	**BrowseMultiple**, *BrowseList*
Call OPC method.	CallMethod	**CallMultipleMethods**, *CallMethodList*
Change parameters of a subscription. Change parameters of a data change subscription.	ChangeMonitoredItemSubscription ChangeDataChangeSubscription	**ChangeMultipleMonitoredItemSubscriptions**, *ChangeMonitoredItemSubscriptionList* ChangeMultipleDataChangeSubscriptions
Discover OPC elements using a generalized query. Discover available OPC applications (LDS). Discover available endpoints of a server (LDS). Discover available OPC servers on a host (LDS). Discover available OPC servers on a network (LDS-ME). Find OPC applications, given LDS endpoints. Globally discovers OPC applications (GDS). Globally discovers OPC servers (GDS).	**Discover** DiscoverLocalApplications DiscoverLocalEndpoints DiscoverLocalServers DiscoverNetworkServers FindLocalApplications DiscoverGlobalApplications DiscoverGlobalServers	
Attempt to pull OPC data change notification.	**PullDataChangeNotification**	**PullMultipleDataChangeNotifications**, *PullDataChangeNotificationList*

Attempt to pull OPC event notification.	**PullEventNotification**	**PullMultipleEventNotifications**, *PullEventNotificationList*
Attempt to pull OPC server condition change notification.	**PullServerConditionChange**	**PullMultipleServerConditionChanges**, *PullServerConditionChangeList*
Read data of a given node's attribute. Read value of a given node's attribute.	Read ReadValue	**ReadMultiple**, *ReadList* ReadMultipleValues, *ReadValueList*
Subscribe to changes of a monitored item. Subscribe to data changes. Subscribe to events.	SubscribeMonitoredItem SubscribeDataChange SubscribeEvent	**SubscribeMultipleMonitoredItems**, *SubscribeMonitoredItemList*
Unsubscribe from changes of a monitored item.	UnsubscribeMonitoredItem	**UnsubscribeMultipleMonitoredItems**, *UnsubscribeMonitoredItemList*
Unsubscribe from changes of all monitored items.	**UnsubscribeAllMonitoredItems**	
Write data into a node. Write value into a node.	Write WriteValue	**WriteMultiple**, *WriteList* WriteMultipleValues, *WriteValueList*

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

5.1.1.1 Multiple-operation Methods

As described in **Operational Methods Overview (Section 5.1.1)**, OPC operations are much more effective if performed together - multiple operations at once. Multiple-operation methods allow you to perform such simultaneous operations, and you should use whenever possible, over single-operation methods.

By convention, multiple-operation methods always contain the word "Multiple" in their name.

In This Topic

Arguments and return values as arrays

In its basic form, the multiple-operation method takes an array (of multiple arrays) of arguments as an input, and produces (as an output) a return value which is an array of results.

The size of the output array is the same as the size of the input array(s), and an element at each index in the result array contains a result that corresponds to the arguments at the same index in the input array(s).

Arguments and return values as lists

This functionality is not available under (or the text does not apply to) .NET development platform (.NET Framework, .NET Standard).

Some COM tools and languages (e.g. PowerScript in PowerBuilder) lack the ability to properly provide or process arrays of COM objects. In such case, using multiple-operation methods would be impossible using their standard array-based form.

In order to make the use of multiple-operations methods in such tools and language possible, OPC Data Client introduces an alternative form of these methods, where arrays are replaced by lists (objects). You can then use methods and properties on the list object in order to access its element, instead of indexing an array. By convention, the list-based methods always contain the word "List" in their name. For correspondence between the array-based and list-based methods, see **Operational Methods Overview (Section 5.1.1)**. Of course, if you prefer, you can use the list-based methods from other tools and languages as well.

The order of elements in the return value corresponds with the order of elements in the method arguments. List elements can also be accessed by index, and therefore an element of the returned list corresponds at certain index corresponds to the element of the arguments list at the same index.

There are also single-operation methods that accepts list instead of array, in case that the argument or return value is an array already, even for a single-operation method. In such case, most of the discussion in this topic can also be applied, except for the correspondence of the order and/or indexes between the arguments and the return value.

Creating and accessing the list

A list object that is passed to a list-based multiple-operation method can be any object that implement the IList Interface. A list object returned from the method also derived from the same interface; in fact, it may provide even broader guarantees as to what interfaces are supported.

You will typically use following members of the list object in order to meaningfully create the list and acess its elements:

- Count Property. Gets the number of elements contained in the list.
- Item[Int32] Property. Gets or sets the element at the specified index.
- Add Method. Adds an item to the list.

In principle, a list can also be enumerated using standard COM collection enumeration approach, although various COM tools and languages also fail implementing it properly. You can, however, use a standard "for" loop, available in practically all languages, with an integer counter, and access the list's element by index. List indexes are zero-based, i.e. the range of valid indexes (for already existing elements of the list) is between 0 and (Count - 1).

The ElasticVector object

OPC Data Client provides an ElasticVector Class, which you can conveniently use to provide list-based arguments to multiple-operation methods. The ElasticVector implements the IList Interface, and has a handful of additional useful members as well.

Above that, it is truly "elastic", in the sense that it can grow automatically with certain operations. It does not shrink by itself. The growing constitutes of lowering the lower bound or increasing the upper bound as necessary for the operation. The absolute element indexes remain unchanged, even when the lower bound changes.

Setting an element at specified index is, for example, always a valid operation with the ElasticVector, regardless of whether the index falls between the current lower and upper bounds of the vector. If it falls outside the range, the bounds are automatically adjusted as needed. This allows the object be used without having to explicitly set the vector dimensions first.

Following examples shows usage of the ElasticVector Class for passing a list of arguments into a multiple-operation method. The example also shows how the return value from the multiple-operation method, which is a list again, can be processed and its elements extracted.

PowerScript

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
```

```
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments. By default, the Value attributes of the nodes will be read.

OLEObject readArguments1
readArguments1 = CREATE OLEObject
readArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10845"

OLEObject readArguments2
readArguments2 = CREATE OLEObject
readArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"

OLEObject readArguments3
readArguments3 = CREATE OLEObject
readArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments3.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10855"

OLEObject readArgumentsList
readArgumentsList = CREATE OLEObject
readArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readArgumentsList.Add(readArguments1)
readArgumentsList.Add(readArguments2)
readArgumentsList.Add(readArguments3)

// Obtain values.
OLEObject valueResultList
valueResultList = client.ReadValueList(readArgumentsList)

// Display results
Int i
FOR i = 0 TO valueResultList.Count - 1
    OLEObject valueResult
    valueResult = valueResultList.Item[i]
    IF valueResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Value: " + String(valueResult.Value) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + valueResult.ErrorMessageBrief
+ "~r~n"
    END IF
NEXT

// Example output:
//
//Value: 8
//Value: -8.06803E+21
//Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

5.1.2 Imperative Programming Model for OPC Data (Classic and UA)

Data Access deals with the representation and use of automation data in OPC Servers. This chapter gives you guidance in

how to implement the common tasks that are needed when dealing with OPC Data Access server from the client side. You achieve these tasks by calling methods on the EasyDAClient or the EasyUAClient object.

5.1.2.1 Obtaining Information (OPC Data)

Methods described in this chapter allow your application to obtain information from the underlying data source that the OPC server connect to (reading OPC Classic items, or value attributes of OPC UA nodes), or from the OPC server itself (getting OPC Classic property values). It is assumed that your application already somehow knows how to identify the data it is interested in. If the location of the data is not known upfront, use methods described in Browsing for Information chapter first.

5.1.2.1.1 Reading from OPC Classic Items

In OPC Data Access, reading data from OPC items is one of the most common tasks. The OPC server generally provides current data for any OPC item in form of a Value, Timestamp and Quality combination (VTQ).

In This Topic

A single item

If you want to read the current VTQ from a specific OPC item, call the ReadItem method. You pass in individual arguments for machine name, server class, ItemID, and an optional data type. You will receive back a DAVtq object holding the current value, timestamp, and quality of the OPC item. The ReadItem method returns the current VTQ, regardless of the quality. You may receive an **Uncertain** or even **Bad** quality (and no usable data value), and your code needs to deal with such situations accordingly.

C#

```
// This example shows how to read a single item, and display its value, timestamp and quality.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadItem
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            DAVtq vtq;
            try
            {
                vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Vtq: {0}", vtq);
        }
    }
}
```

PHP

```
// This example shows how to read a single item, and display its value, timestamp and quality.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $Vtq = $Client->ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

printf("Vtq: %s\n", $Vtq);
```

VB.NET

```
' This example shows how to read a single item, and display its value, timestamp and quality.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItem
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim vtq As DAVtq
            Try
                vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Vtq: {0}", vtq)
        End Sub
    End Class
End Namespace
```

JScript

```
// This example shows how to read a single item, and display its value, timestamp and quality.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
var VTQ = Client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
WScript.Echo("VTQ.ToString(): " + VTQ.ToString());
```

Object Pascal

```
// This example shows how to read a single item, and display its value, timestamp and quality.

class procedure ReadItem.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Vtq: _DAVtq;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
```

```
    Vtq := Client.ReadItem('', 'OPCLabs.KitServer.2', 'Simulation.Random');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('Vtq: ', Vtq.ToString);
end;
```

PowerScript

```
// This example shows how to read a single item, and display its value, timestamp and quality.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Obtain value/timestamp/quality
OLEObject vtq
TRY
    vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Demo.Ramp")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + oleRuntimeError.Description +
"~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + "Vtq: " + vtq.DisplayString + "~r~n"
```

VBScript

```
Rem This example shows how to read a single item, and display its value, timestamp and quality.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim Vtq: Set Vtq = Client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "Vtq: " & Vtq
```

In OPC Data Client.NET, you can also pass ServerDescriptor and DAItemDescriptor objects in place of individual arguments to the ReadItem method.

C#

```
// This example shows how to read a single item using a browse path, and display its value,
timestamp and quality.

using System;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
```

```
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadItem
    {
        public static void BrowsePath()
        {
            var client = new EasyDAClient();

            DAVtq vtq;
            try
            {
                vtq = client.ReadItem(
                    new ServerDescriptor("", "OPCLabs.KitServer.2"),
                    new DAItemDescriptor(null, "/Simulation/Random"));
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Vtq: {0}", vtq);
        }
    }
}
```

VB.NET

```
' This example shows how to read a single item using a browse path, and display its value, timestamp
and quality.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItem
        Public Shared Sub BrowsePath()
            Dim client = New EasyDAClient()

            Dim vtq As DAVtq
            Try
                vtq = client.ReadItem(New ServerDescriptor("", "OPCLabs.KitServer.2"), New
DAItemDescriptor(Nothing, "/Simulation/Random"))
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Vtq: {0}", vtq)
        End Sub
    End Class
End Namespace
```

Multiple items

For reading VTQs of multiple items simultaneously in an efficient manner, call the ReadMultipleItems method (instead of multiple ReadItem calls in a loop). You will receive back an array of DAVtqResult objects.

In OPC Data Client.NET, you can pass in a ServerDescriptor object and an array of DAItemDescriptor objects, or an array of DAItemArguments objects, to the ReadMultipleItems method.

C#

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadMultipleItems
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            DAVtqResult[] vtqResults = client.ReadMultipleItems("OPCLabs.KitServer.2",
                new DAItemDescriptor[]
                    {
                        "Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1 min)",
"Simulation.Register_I4"
                    });

            for (int i = 0; i < vtqResults.Length; i++)
            {
                Debug.Assert(vtqResults[i] != null);

                if (vtqResults[i].Succeeded)
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults[i].Vtq);
                else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults[i].ErrorMessageBrief);
            }
        }
    }
}
```

PHP

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

$ReadItemArguments1 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments1->ItemDescriptor->ItemID = "Simulation.Random";

$ReadItemArguments2 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments2->ItemDescriptor->ItemID = "Trends.Ramp (1 min)";

$ReadItemArguments3 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments3->ItemDescriptor->ItemID = "Trends.Sine (1 min)";

$ReadItemArguments4 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments4->ItemDescriptor->ItemID = "Simulation.Register_I4";

$arguments[0] = $ReadItemArguments1;
$arguments[1] = $ReadItemArguments2;
```

```
$arguments[2] = $ReadItemArguments3;
$arguments[3] = $ReadItemArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

$results = $Client->ReadMultipleItems($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $VtqResult = $results[$i];
    if ($VtqResult->Succeeded)
        printf("results[%d].Vtq.ToString(): %s\n", $i, $VtqResult->Vtq->ToString);
    else
        printf("results[%d]: *** Failure: %s\n", $i, $VtqResult->ErrorMessageBrief);
}
```

VB.NET

```
' This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadMultipleItems
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems(
                "OPCLabs.KitServer.2",
                New DAItemDescriptor() {"Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1
min)", "Simulation.Register_I4"})

            For i = 0 To vtqResults.Length - 1
                Debug.Assert(vtqResults(i) IsNot Nothing)

                If vtqResults(i).Succeeded Then
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults(i).Vtq)
                Else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults(i).ErrorMessageBrief)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

class procedure ReadMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ReadItemArguments1: _DAReadItemArguments;
  ReadItemArguments2: _DAReadItemArguments;
  ReadItemArguments3: _DAReadItemArguments;
  ReadItemArguments4: _DAReadItemArguments;
  VtqResult: _DAVtqResult;
  Results: OleVariant;
begin
```

```
  ReadItemArguments1 := CoDAReadItemArguments.Create;
  ReadItemArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments1.ItemDescriptor.ItemID := 'Simulation.Random';

  ReadItemArguments2 := CoDAReadItemArguments.Create;
  ReadItemArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments2.ItemDescriptor.ItemID := 'Trends.Ramp (1 min)';

  ReadItemArguments3 := CoDAReadItemArguments.Create;
  ReadItemArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments3.ItemDescriptor.ItemID := 'Trends.Sine (1 min)';

  ReadItemArguments4 := CoDAReadItemArguments.Create;
  ReadItemArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments4.ItemDescriptor.ItemID := 'Simulation.Register_I4';

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ReadItemArguments1;
  Arguments[1] := ReadItemArguments2;
  Arguments[2] := ReadItemArguments3;
  Arguments[3] := ReadItemArguments4;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.ReadMultipleItems(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
      VtqResult := IInterface(Results[I]) as _DAVtqResult;
      if VtqResult.Succeeded then
        WriteLn('results(', i, ').Vtq.ToString(): ', VtqResult.Vtq.ToString)
      else
        WriteLn('results(', i, ') *** Failure: ', VtqResult.ErrorMessageBrief);
  end;
end;
```

PowerScript

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare arguments.

OLEObject readItemArguments1
readItemArguments1 = CREATE OLEObject
readItemArguments1.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

OLEObject readItemArguments2
readItemArguments2 = CREATE OLEObject
readItemArguments2.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
```

```
readItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

OLEObject readItemArguments3
readItemArguments3 = CREATE OLEObject
readItemArguments3.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

OLEObject readItemArguments4
readItemArguments4 = CREATE OLEObject
readItemArguments4.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

OLEObject readItemArgumentsList
readItemArgumentsList = CREATE OLEObject
readItemArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readItemArgumentsList.Add(readItemArguments1)
readItemArgumentsList.Add(readItemArguments2)
readItemArgumentsList.Add(readItemArguments3)
readItemArgumentsList.Add(readItemArguments4)

// Obtain values/timestamps/qualities.
OLEObject vtqResultList
vtqResultList = client.ReadItemList(readItemArgumentsList)

// Display results
Int i
FOR i = 0 TO vtqResultList.Count - 1
    OLEObject vtqResult
    vtqResult = vtqResultList.Item[i]
    IF vtqResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "vtqResult[" + String(i) + "].Vtq: " +
String(vtqResult.Vtq) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "vtqResult[" + String(i) + "] *** Failure: " +
vtqResult.ErrorMessageBrief + "~r~n"
    END IF
NEXT
```

VBScript

```
Rem This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
```

```
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

VBScript

```
Rem This example reads a large number of items.

Option Explicit

Const repeatCount = 10
Const numberOfItems = 1000

WScript.Echo "Creating array of arguments..."
Dim arguments(): ReDim arguments(numberOfItems - 1)
Dim i: For i = 0 To numberOfItems - 1
    Dim copy: copy = Int(i / 100) + 1
    Dim phase: phase = i Mod 100
    Dim itemId: itemId = "Simulation.Incrementing.Copy_" & copy & ".Phase_" & phase
    WScript.Echo itemId

    Dim ReadItemArguments: Set ReadItemArguments =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
    ReadItemArguments.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
    ReadItemArguments.ItemDescriptor.ItemID = itemId

    Set arguments(i) = ReadItemArguments
Next

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim iRepeat: For iRepeat = 1 To repeatCount
    WScript.Echo "Reading items..."
    Dim results: results = Client.ReadMultipleItems(arguments)

    Dim successCount: successCount = 0
    For i = LBound(results) To UBound(results)
        Dim VtqResult: Set VtqResult = results(i)
        If VtqResult.Succeeded Then successCount = successCount + 1
    Next
    WScript.Echo "Success count: " & successCount
Next
```

VBScript

```
Rem This example shows how to read 4 items at once synchronously, and display their values,
timestamps and qualities.
```

```
Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

' Specify that only synchronous method is allowed. By default, both synchronous and asynchronous
methods are allowed, and
' the components picks a suitable method automatically. Disallowing asynchronous method leaves only
the synchronous method
' available for selection.
Client.InstanceParameters.Mode.AllowAsynchronousMethod = False

Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

VBScript

```
Rem This example shows how to read 2 items (first valid, second invalid), test for success of each
read and display either
Rem the DAVtq or the Exception.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "UnknownItem"
```

```
Dim arguments(1)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

VBScript

```
Rem This example attempts to read 3 items (first valid, second invalid, third valid again), and
display their values,
Rem timestamps and qualities. Without testing for a success, a run-time error occurs when accessing
the Vtq property
Rem of a failed result.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "UnknownItem"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim arguments(2)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    ' This is an anti-example - you should not access the Vtq property without testing the success
first!
    WScript.Echo "results(" & i & ").Vtq.ToString(): " & results(i).Vtq.ToString()
Next
```

Read parameters

Note: In OPC Data Client.NET, you can use the optional argument of type DAReadParameters, and then set the ValueAge property in it to control how "old" may be the values you receive by reading from OPC items. With the DataSource property, you can also specify that you want values from the device, or from cache. You can also use the pre-defined DAReadParameters.CacheSource and DAReadParameters.DeviceSource constants to specify the read parameters. There are also overloads of ReadXXXX methods that accept an integer valueAge argument.

Be aware that it is physically impossible for any system to always obtain fully up-to-date values all the time.

VBScript

```
Rem This example shows how to read 4 items from the device, and display their values, timestamps and
qualities.

Option Explicit

' Selects the data source for OPC reads (from device, from OPC cache, or dynamically determined).
' The data source (memory, OPC cache or OPC device) selection will be based on the desired value age
and current status of
' data received from the server.
Const DADataSource_ByValueAge = 0
' OPC reads will be fulfilled from the cache in the OPC server.
Const DADataSource_Cache = 1
' OPC reads will be fulfilled from the device by the OPC server.
Const DADataSource_Device = 2

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ReadItemArguments1.ReadParameters.DataSource = DADataSource_Device  ' read will be from device

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ReadItemArguments2.ReadParameters.DataSource = DADataSource_Device  ' read will be from device

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ReadItemArguments3.ReadParameters.DataSource = DADataSource_Device  ' read will be from device

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ReadItemArguments4.ReadParameters.DataSource = DADataSource_Device  ' read will be from device

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

5.1.2.1.1.1 Reading just the value (OPC Classic)

Some applications need the actual data value for further processing (e.g. for computations that need be performed on the values), even if it involves waiting a little for the quality to become **Good**.

In This Topic

A single item

For such usage, call the ReadItemValue method, passing it the same arguments as to the ReadItem method. The method will wait until the OPC item's quality becomes **Good** (or until a timeout expires), and you will receive back an Object (a VARIANT in OPC Data Client-COM) holding the actual data value.

Note: In OPC XML-DA, the ReadItemValue method does not wait for the **Good** quality. It makes a single read, and if the value obtained does not have a **Good** quality, the method returns an error.

C#

```
// This example shows how to read and display value of a single item.
// One of the shortest examples possible.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    class ReadItemValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Reading item value...");
            object value;
            try
            {
                value = client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Ramp");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine(value);
        }
    }
}
```

PHP

```
// This example shows how to read value of a single item, and display it.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $value = $Client->ReadItemValue("", "OPCLabs.KitServer.2", "Simulation.Random");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}
```

```
printf("value: %s\n", $value);
```

VB.NET

```
' This example shows how to read and display value of a single item.
' One of the shortest examples possible.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItemValue
        Public Shared Sub Main1()
            Dim client As New EasyDAClient()

            Console.WriteLine("Reading item...")
            Try
                Console.WriteLine(client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Ramp"))
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

JScript

```
// This example shows how to read values of a single item, and display it.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
var value = Client.ReadItemValue("", "OPCLabs.KitServer.2", "Simulation.Random");
WScript.Echo("value: " + value);
```

Object Pascal

```
// This example shows how to read value of a single item, and display it.

class procedure ReadItemValue.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Value: OleVariant;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    Value := Client.ReadItemValue('', 'OPCLabs.KitServer.2', 'Simulation.Random');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('Value: ', Value);
end;
```

PowerScript

```
// This example shows how to read and display value of a single item.
// One of the shortest examples possible.
```

```
mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Obtain value of a node
mle_outputtext.Text = mle_outputtext.Text + "Reading item value..." + "~r~n"
Any value
TRY
    value = client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Ramp")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + oleRuntimeError.Description +
"~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + String(value) + "~r~n"
```

Python

```
# This example shows how to read value of a single item, and display it.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.DataAccess.EasyDAClient')

# Perform the operation
value = client.ReadItemValue('', 'OPCLabs.KitServer.2', 'Demo.Single')

# Display results
print('value: ', value)
```

VBScript

```
Rem This example shows how to read value of a single item, and display it.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim value: value = Client.ReadItemValue("", "OPCLabs.KitServer.2", "Simulation.Random")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "value: " & value
```

VBScript

```
Rem This example shows how to read value of a single item, and display it, using CLSID instead of
ProgID of the OPC Server.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim value: value = Client.ReadItemValue("", "{C8A12F17-1E03-401E-B53D-6C654DD576DA}",
"Simulation.Random")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
```

```
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "value: " & value
```

Multiple items

For reading just the data values of multiple data values (with wait for **Good** quality) in an efficient manner, call the ReadMultipleItemValues method (instead of multiple ReadItemValue calls in a loop). You will receive back an array of ValueResult objects.

Note: In OPC XML-DA, the ReadMultipleItemValues method does not wait for the **Good** quality. It makes a single read, and if the value obtained does not have a **Good** quality, the method returns an error for such an item.

C#

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadMultipleItems
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            DAVtqResult[] vtqResults = client.ReadMultipleItems("OPCLabs.KitServer.2",
                new DAItemDescriptor[]
                    {
                        "Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1 min)",
"Simulation.Register_I4"
                    });

            for (int i = 0; i < vtqResults.Length; i++)
            {
                Debug.Assert(vtqResults[i] != null);

                if (vtqResults[i].Succeeded)
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults[i].Vtq);
                else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults[i].ErrorMessageBrief);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
```

```
    Partial Friend Class ReadMultipleItems
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems(
                "OPCLabs.KitServer.2",
                New DAItemDescriptor() {"Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1
min)", "Simulation.Register_I4"})

            For i = 0 To vtqResults.Length - 1
                Debug.Assert(vtqResults(i) IsNot Nothing)

                If vtqResults(i).Succeeded Then
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults(i).Vtq)
                Else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults(i).ErrorMessageBrief)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

class procedure ReadMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ReadItemArguments1: _DAReadItemArguments;
  ReadItemArguments2: _DAReadItemArguments;
  ReadItemArguments3: _DAReadItemArguments;
  ReadItemArguments4: _DAReadItemArguments;
  VtqResult: _DAVtqResult;
  Results: OleVariant;
begin
  ReadItemArguments1 := CoDAReadItemArguments.Create;
  ReadItemArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments1.ItemDescriptor.ItemID := 'Simulation.Random';

  ReadItemArguments2 := CoDAReadItemArguments.Create;
  ReadItemArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments2.ItemDescriptor.ItemID := 'Trends.Ramp (1 min)';

  ReadItemArguments3 := CoDAReadItemArguments.Create;
  ReadItemArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments3.ItemDescriptor.ItemID := 'Trends.Sine (1 min)';

  ReadItemArguments4 := CoDAReadItemArguments.Create;
  ReadItemArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments4.ItemDescriptor.ItemID := 'Simulation.Register_I4';

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ReadItemArguments1;
  Arguments[1] := ReadItemArguments2;
  Arguments[2] := ReadItemArguments3;
  Arguments[3] := ReadItemArguments4;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  TVarData(Results).VType := varArray or varVariant;
```

```
  TVarData(Results).VArray := PVarArray(
    Client.ReadMultipleItems(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
      VtqResult := IInterface(Results[I]) as _DAVtqResult;
      if VtqResult.Succeeded then
        WriteLn('results(', i, ').Vtq.ToString(): ', VtqResult.Vtq.ToString)
      else
        WriteLn('results(', i, ') *** Failure: ', VtqResult.ErrorMessageBrief);
  end;
end;
```

PowerScript

```
// This example shows how to read the values of 4 different items at once.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare arguments.

OLEObject readItemArguments1
readItemArguments1 = CREATE OLEObject
readItemArguments1.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

OLEObject readItemArguments2
readItemArguments2 = CREATE OLEObject
readItemArguments2.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

OLEObject readItemArguments3
readItemArguments3 = CREATE OLEObject
readItemArguments3.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

OLEObject readItemArguments4
readItemArguments4 = CREATE OLEObject
readItemArguments4.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

OLEObject readItemArgumentsList
readItemArgumentsList = CREATE OLEObject
readItemArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readItemArgumentsList.Add(readItemArguments1)
readItemArgumentsList.Add(readItemArguments2)
readItemArgumentsList.Add(readItemArguments3)
readItemArgumentsList.Add(readItemArguments4)

// Obtain values.
OLEObject valueResultList
valueResultList = client.ReadItemValueList(readItemArgumentsList)
```

```
// Display results
Int i
FOR i = 0 TO valueResultList.Count - 1
    OLEObject valueResult
    valueResult = valueResultList.Item[i]
    IF valueResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "valueResult[" + String(i) + "].Value: " +
String(valueResult.Value) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "valueResult[" + String(i) + "] *** Failure: " +
valueResult.ErrorMessageBrief + "~r~n"
    END IF
NEXT
```

VBScript

```
Rem This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

In OPC Data Client.NET, you can pass in a ServerDescriptor object and an array of DAItemDescriptor objects, or an array of DAItemArguments objects, to the ReadMultipleItemValues method.

5.1.2.1.1.2 Reading in OPC XML-DA

In OPC XML-DA, reading or writing is in principle no different from COM-based OPC. You just need to specify the OPC server using its URL instead of the ProgID or CLSID, and obey the rules for identificaton of the items (which are server-dependent). Sometimes, you may need an additional piece of information to identify nodes in OPC XML-DA address space; for more information see **Identifying information in OPC XML (Section 4.10)**.

C#

```
// This example shows how to read 4 items from an OPC XML-DA server at once, and
display their values, timestamps
// and qualities.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class ReadMultipleItems
    {
        public static void Main1Xml()
        {
            var client = new EasyDAClient();

            DAVtqResult[] vtqResults = client.ReadMultipleItems(
                new ServerDescriptor { UrlString = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx" },
                new DAItemDescriptor[]
                    {
                        "Dynamic/Analog Types/Double",
                        "Dynamic/Analog Types/Double[]",
                        "Dynamic/Analog Types/Int",
                        "SomeUnknownItem"
                    });

            for (int i = 0; i < vtqResults.Length; i++)
            {
                Debug.Assert(vtqResults[i] != null);

                if (vtqResults[i].Exception != null)
                {
                    Console.WriteLine("vtqResults[{0}] *** Failure: {1}", i,
vtqResults[i].ErrorMessageBrief);
                    continue;
                }
                Console.WriteLine("vtqResults[{0}].Vtq: {1}", i, vtqResults[i].Vtq);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to read 4 items from an OPC XML-DA server at once, and display
their values, timestamps
' and qualities.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class ReadMultipleItems
        Public Shared Sub Main1Xml()
            Dim client = New EasyDAClient()

            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems(
                    New ServerDescriptor() With {.UrlString = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx"},
                    New DAItemDescriptor() _
                    { _
                        "Dynamic/Analog Types/Double",
                        "Dynamic/Analog Types/Double[]",
                        "Dynamic/Analog Types/Int",
                        "SomeUnknownItem"
                    })

            For i = 0 To vtqResults.Length - 1
                Debug.Assert(vtqResults(i) IsNot Nothing)

                If vtqResults(i).Exception IsNot Nothing Then
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults(i).ErrorMessageBrief)
                    Continue For
                End If
                Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults(i).Vtq)
            Next i
        End Sub
    End Class
End Namespace
```

5.1.2.1.2 Getting OPC Classic Property Values

Each OPC item has typically associated a set of OPC properties with it. OPC properties contain additional information related to the item.

> ⚠ Except in special cases with clear justification, do not use the OPC Classic property mechanism to get "dynamic" properties such as Value, Quality, or Timestamp. Doing so is very inefficient and has additional issues associated with it (reliability, consistency). Use normal read or subscriptions instead.

In This Topic

A single property

If you want to obtain the value of specific OPC property, call the GetPropertyValue method, passing it the machine name, server class, the ItemId, and a PropertyId. You will receive back an Object (a VARIANT in OPC Data Client-COM) containing the value of the requested property.

C#

```
// This example shows how to get a value of a single OPC property.
//
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to obtain
better property values. Your code may
// also subscribe to the item in order to assure that it remains active.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class GetPropertyValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            object value;
            try
            {
                value = client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random",
                    DAPropertyIds.Timestamp);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine(value);
        }
    }
}
```

PHP

```
// This example shows how to get a value of a single OPC property.
//
// Note that some properties may not have a useful value initially (e.g. until the item
```

```
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to obtain
better property values. Your code may
// also subscribe to the item in order to assure that it remains active.

const Timestamp = 4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $value = $Client->GetPropertyValue("", "OPCLabs.KitServer.2", "Simulation.Random",
Timestamp);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

printf("%s\n", $value);
```

VB.NET

```
' This example shows how to get a value of a single OPC property.
'
' Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
' case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
' IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to obtain
better property values. Your code may
' also subscribe to the item in order to assure that it remains active.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class GetPropertyValue
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim value As Object
            Try
                value = client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random", DAPropertyIds.Timestamp)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine(value)
        End Sub
    End Class
```

```
End Namespace
```

VBScript

```
Rem This example shows how to get a value of a single OPC property.
Rem
Rem Note that some properties may not have a useful value initially (e.g. until the
item is activated in a group), which also the
Rem case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
Rem IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to
obtain better property values. Your code may
Rem also subscribe to the item in order to assure that it remains active.

Option Explicit

Const Timestamp = 4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

On Error Resume Next
Dim value: value = Client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random", Timestamp)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo value
```

VBScript

```
Rem This example measures the time needed to get values of all OPC properties of a
single OPC item "one by one".

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.ReadValue_I4"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim PropertyElementCollection
On Error Resume Next
Set PropertyElementCollection = Client.BrowseProperties(ServerDescriptor,
NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

```
'EasyDAClient.ReadItemValue "", "OPCLabs.KitServer.2", "Simulation.ReadValue_I4"
Dim startTime: startTime = Timer
Dim PropertyElement: For Each PropertyElement In PropertyElementCollection
    Dim propertyID: Set propertyID = PropertyElement.PropertyID
    On Error Resume Next
    Dim value: value = Client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.ReadValue_I4", propertyID.NumericalValue)
    If Err.Number <> 0 Then
        WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
        WScript.Quit
    End If
    On Error Goto 0
    'WScript.Echo value
Next
WScript.Echo "Time taken (milliseconds): " & (Timer - startTime)*1000
```

The GetPropertyValue Method returns a generic object, regardless of the property involved. You may therefore need to cast the returned value to the expected type, and possibly do further processing on the value, as in the the example below.

C#

```
// This example shows how to obtain a data type of an OPC item.

using System;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class GetPropertyValue
    {
        public static void DataType()
        {
            var client = new EasyDAClient();

            // Get the value of DataType property; it is a 16-bit signed integer
            short dataType;
            try
            {
                dataType = (short)client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random",
                    DAPropertyIds.DataType);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
            // Convert the data type to VarType
            var varType = (VarType)dataType;

            // Display the obtained data type
```

```
            Console.WriteLine("DataType: {0}", dataType);    // Display data type as
numerical value
            Console.WriteLine("VarType: {0}", varType);      // Display data type
symbolically

            // Code below illustrates how decisions can be made based on type
            switch (varType.InternalValue)
            {
                case VarTypes.R8:
                    Console.WriteLine("The data type is VarTypes.R8, as we expected.");
                    break;

                // other cases may come here ...

                default:
                    Console.WriteLine("The data type is not as we expected!");
                    break;
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of an OPC item.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class GetPropertyValue
        Public Shared Sub DataType()
            Dim client = New EasyDAClient()

            ' Get the value of DataType property; it is a 16-bit signed integer
            Dim aDataType As Short
            Try
                aDataType = CShort(Fix(client.GetPropertyValue("",
"OPCLabs.KitServer.2", "Simulation.Random", DAPropertyIds.DataType)))
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Convert the data type to VarType
            Dim varType = CType(aDataType, VarType)

            ' Display the obtained data type
            Console.WriteLine("DataType: {0}", aDataType) ' Display data type as
numerical value
            Console.WriteLine("VarType: {0}", varType) ' Display data type symbolically

            ' Code below illustrates how decisions can be made based on type
            Select Case varType
                Case VarTypes.R8
```

```
                    Console.WriteLine("The data type is VarTypes.R8, as we expected.")

                    ' other cases may come here ...

                Case Else
                    Console.WriteLine("The data type is not as we expected!")
            End Select
        End Sub
    End Class
End Namespace
```

In OPC Data Client.NET, you can also pass in the ServerDescriptor inplace of the machine name and server class strings.

In addition to basic methods, there are also many property-related methods. Please refer to a chapter in this document that describes EasyOPC.NET Extensions.

Multiple properties

For obtaining multiple properties simultaneously in an efficient manner, call the GetMultiplePropertyValues method (instead of multiple GetPropertyValue calls in a loop). The arguments are similar, except that in place of a single PropertyId you pass in an array of them. You will receive back an array of Object values (a SAFEARRAY of VARIANT values in OPC Data Client-COM).

OPC Data Access

Object Pascal

```
// This example shows how to get value of multiple OPC properties.
//
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadMultipleItemValues.Main.vbs shortly before this example, in order
to obtain better property values. Your
// code may also subscribe to the items in order to assure that they remain active.

class procedure GetMultiplePropertyValues.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  PropertyArguments1: _DAPropertyArguments;
  PropertyArguments2: _DAPropertyArguments;
  PropertyArguments3: _DAPropertyArguments;
  PropertyArguments4: _DAPropertyArguments;
  ValueResult: _ValueResult;
  Results: OleVariant;
begin
  PropertyArguments1 := CoDAPropertyArguments.Create;
```

```
  PropertyArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments1.NodeDescriptor.ItemID := 'Simulation.Random';
  PropertyArguments1.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_Timestamp;

  PropertyArguments2 := CoDAPropertyArguments.Create;
  PropertyArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments2.NodeDescriptor.ItemID := 'Simulation.Random';
  PropertyArguments2.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_AccessRights;

  PropertyArguments3 := CoDAPropertyArguments.Create;
  PropertyArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments3.NodeDescriptor.ItemID := 'Trends.Ramp (1 min)';
  PropertyArguments3.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_Timestamp;

  PropertyArguments4 := CoDAPropertyArguments.Create;
  PropertyArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments4.NodeDescriptor.ItemID := 'Trends.Ramp (1 min)';
  PropertyArguments4.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_AccessRights;

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := PropertyArguments1;
  Arguments[1] := PropertyArguments2;
  Arguments[2] := PropertyArguments3;
  Arguments[3] := PropertyArguments4;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.GetMultiplePropertyValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
      ValueResult := IInterface(Results[I]) as _ValueResult;

      // Check if there has been an error getting the property value
      if ValueResult.Exception <> nil then
      begin
        WriteLn(Format('%s *** Failure: %s', [Arguments[I].NodeDescriptor.NodeId,
ValueResult.Exception.Message]));
        Continue;
      end;

      WriteLn('results(', i, ').Value: ', ValueResult.Value);
  end;
end;
```

PHP

```
// This example shows how to get value of multiple OPC properties, and handle errors.
//
```

```
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadMultipleItemValues.Main.vbs shortly before this example, in order
to obtain better property values. Your
// code may also subscribe to the items in order to assure that they remain active.

const Timestamp = 4;
const AccessRights = 5;

$PropertyArguments1 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments1->NodeDescriptor->ItemID = "Simulation.Random";
$PropertyArguments1->PropertyDescriptor->PropertyID->NumericalValue = Timestamp;

$PropertyArguments2 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments2->NodeDescriptor->ItemID = "Simulation.Random";
$PropertyArguments2->PropertyDescriptor->PropertyID->NumericalValue = AccessRights;

$PropertyArguments3 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments3->NodeDescriptor->ItemID = "Trends.Ramp (1 min)";
$PropertyArguments3->PropertyDescriptor->PropertyID->NumericalValue = Timestamp;

$PropertyArguments4 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments4->NodeDescriptor->ItemID = "Trends.Ramp (1 min)";
$PropertyArguments4->PropertyDescriptor->PropertyID->NumericalValue = AccessRights;

$arguments[0] = $PropertyArguments1;
$arguments[1] = $PropertyArguments2;
$arguments[2] = $PropertyArguments3;
$arguments[3] = $PropertyArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

$results = $Client->GetMultiplePropertyValues($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $attributeDataResult = $results[$i];
    if ($results[$i]->Succeeded)
        printf("results[%d].Value: %s\n", $i, $results[$i]->Value);
    else
        printf("results[%d]: *** Failure: %s\n", $i, $results[$i]->ErrorMessageBrief);
}
```

VBScript

```
Rem This example shows how to get value of multiple OPC properties, and handle errors.
Rem
```

```
Rem Note that some properties may not have a useful value initially (e.g. until the
item is activated in a group), which also the
Rem case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
Rem IEasyDAClient.ReadMultipleItemValues.Main.vbs shortly before this example, in order
to obtain better property values. Your
Rem code may also subscribe to the items in order to assure that they remain active.

Option Explicit

Const Timestamp = 4
Const AccessRights = 5

Dim PropertyArguments1: Set PropertyArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments1.NodeDescriptor.ItemID = "Simulation.Random"
PropertyArguments1.PropertyDescriptor.PropertyID.NumericalValue = Timestamp

Dim PropertyArguments2: Set PropertyArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments2.NodeDescriptor.ItemID = "Simulation.Random"
PropertyArguments2.PropertyDescriptor.PropertyID.NumericalValue = AccessRights

Dim PropertyArguments3: Set PropertyArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments3.NodeDescriptor.ItemID = "Trends.Ramp (1 min)"
PropertyArguments3.PropertyDescriptor.PropertyID.NumericalValue = Timestamp

Dim PropertyArguments4: Set PropertyArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments4.NodeDescriptor.ItemID = "Trends.Ramp (1 min)"
PropertyArguments4.PropertyDescriptor.PropertyID.NumericalValue = AccessRights

Dim arguments(3)
Set arguments(0) = PropertyArguments1
Set arguments(1) = PropertyArguments2
Set arguments(2) = PropertyArguments3
Set arguments(3) = PropertyArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.GetMultiplePropertyValues(arguments)

Dim i: For i = LBound(results) To UBound(results)
    If results(i).Exception Is Nothing Then
        WScript.Echo "results(" & i & ").Value: " & results(i).Value
    Else
        WScript.Echo "results(" & i & ").Exception.Message: " &
results(i).Exception.Message
    End If
Next
```

VBScript

```
Rem This example measures the time needed to get values of all OPC properties of a
single OPC item all at once.
Rem This example shows how to get value of multiple OPC properties.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.ReadValue_I4"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim PropertyElementCollection
On Error Resume Next
Set PropertyElementCollection = Client.BrowseProperties(ServerDescriptor,
NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim count: count = PropertyElementCollection.Count

Dim arguments(): Redim arguments(count - 1)
Dim i: i = 0
Dim PropertyElement: For Each PropertyElement In PropertyElementCollection
    Dim PropertyArguments: Set PropertyArguments =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
    PropertyArguments.ServerDescriptor = ServerDescriptor
    PropertyArguments.NodeDescriptor = NodeDescriptor
    PropertyArguments.PropertyDescriptor.PropertyID = PropertyElement.PropertyId

    Set arguments(i) = PropertyArguments
    i = i + 1
Next

'EasyDAClient.ReadItemValue "", "OPCLabs.KitServer.2", "Simulation.ReadValue_I4"
Dim startTime: startTime = Timer
Dim results: results = Client.GetMultiplePropertyValues(arguments)
WScript.Echo "Time taken (milliseconds): " & (Timer - startTime)*1000

'For i = LBound(results) To UBound(results)
'    If results(i).Exception Is Nothing Then
'        WScript.Echo "results(" & i & ").Value: " & results(i).Value
'    Else
'        WScript.Echo "results(" & i & ").Exception.Message: " &
results(i).Exception.Message
'    End If
''Next
```

The example below is a bit more complex, and combines the ability to browse for items with getting the data type

property for each of the items obtained.

C#

```
// This example shows how to obtain a data type of all OPC items under a branch.

using System;
using System.Linq;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;

namespace DocExamples.DataAccess._EasyDAClient
{
    class GetMultiplePropertyValues
    {
        public static void DataType()
        {
            var client = new EasyDAClient();
            ServerDescriptor serverDescriptor = "OPCLabs.KitServer.2";

            // Browse for all leaves under the "Simulation" branch
            DANodeElementCollection nodeElementCollection =
client.BrowseLeaves(serverDescriptor, "Simulation");

            // Create list of node descriptors, one for each leaf obtained
            DANodeDescriptor[] nodeDescriptorArray = nodeElementCollection
                .Where(element => !element.IsHint)  // filter out hint leafs that do
not represent real OPC items (rare)
                .Select(element => new DANodeDescriptor(element))
                .ToArray();

            // Get the value of DataType property; it is a 16-bit signed integer
            ValueResult[] valueResultArray =
client.GetMultiplePropertyValues(serverDescriptor,
                nodeDescriptorArray, DAPropertyIds.DataType);

            for (int i = 0; i < valueResultArray.Length; i++)
            {
                DANodeDescriptor nodeDescriptor = nodeDescriptorArray[i];

                // Check if there has been an error getting the property value
                ValueResult valueResult = valueResultArray[i];
                if (valueResult.Exception != null)
                {
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message);
                    continue;
                }

                // Convert the data type to VarType
                var varType = (VarType)(short)valueResult.Value;

                // Display the obtained data type
```

```
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of all OPC items under a branch.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess._EasyDAClient
    Friend Class GetMultiplePropertyValues
        Public Shared Sub DataType()
            Dim client = New EasyDAClient()
            Dim serverDescriptor As ServerDescriptor = "OPCLabs.KitServer.2"

            ' Browse for all leaves under the "Simulation" branch
            Dim nodeElementCollection = client.BrowseLeaves(serverDescriptor,
"Simulation")

            ' Create list of node descriptors, one for each leaf obtained
            ' filter out hint leafs that do not represent real OPC items (rare)
            Dim nodeDescriptorArray() As DANodeDescriptor = nodeElementCollection _
                .Where(Function(element) Not element.IsHint) _
                .Select(Function(element) New DANodeDescriptor(element)) _
                .ToArray()

            ' Get the value of DataType property; it is a 16-bit signed integer
            Dim valueResultArray() As ValueResult =
client.GetMultiplePropertyValues(serverDescriptor,
                    nodeDescriptorArray, DAPropertyIds.DataType)

            For i = 0 To valueResultArray.Length - 1
                Dim nodeDescriptor = nodeDescriptorArray(i)

                ' Check if there has been an error getting the property value
                Dim valueResult As ValueResult = valueResultArray(i)
                If valueResult.Exception IsNot Nothing Then
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message)
                    Continue For
                End If

                ' Convert the data type to VarType
                Dim varType = CType(CShort(valueResult.Value), VarType)

                ' Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType)
            Next i
        End Sub
    End Class
End Namespace
```

OPC XML-DA

The example below combines the ability to browse for items with getting the data type property for each of the items obtained, for OPC XML-DA Servers.

C#

```
// This example shows how to obtain a data type of all OPC XML-DA items under a branch.

using System;
using System.Linq;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;

namespace DocExamples.DataAccess.Xml
{
    class GetMultiplePropertyValues
    {
        public static void DataTypeXml()
        {
            var client = new EasyDAClient();
            ServerDescriptor serverDescriptor = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx";

            // Browse for all leaves under the "Simulation" branch
            DANodeElementCollection nodeElementCollection =
client.BrowseLeaves(serverDescriptor, "Static/Analog Types");

            // Create list of node descriptors, one for each leaf obtained
            DANodeDescriptor[] nodeDescriptorArray = nodeElementCollection
                .Where(element => !element.IsHint)  // filter out hint leafs that do
not represent real OPC items (rare)
                .Select(element => new DANodeDescriptor(element))
                .ToArray();

            // Get the value of DataType property; it is a 16-bit signed integer
            ValueResult[] valueResultArray =
client.GetMultiplePropertyValues(serverDescriptor,
                nodeDescriptorArray, DAPropertyIds.DataType);

            for (int i = 0; i < valueResultArray.Length; i++)
            {
                DANodeDescriptor nodeDescriptor = nodeDescriptorArray[i];

                // Check if there has been an error getting the property value
                ValueResult valueResult = valueResultArray[i];
                if (valueResult.Exception != null)
                {
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message);
                    continue;
                }
```

```
                // Convert the data type to VarType
                var varType = (VarType)(short)valueResult.Value;

                // Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of all OPC XML-DA items under a branch.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess.Xml
    Friend Class GetMultiplePropertyValues
        Public Shared Sub DataTypeXml()
            Dim client = New EasyDAClient()
            Dim serverDescriptor As ServerDescriptor = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx"

            ' Browse for all leaves under the "Simulation" branch
            Dim nodeElementCollection = client.BrowseLeaves(serverDescriptor,
"Static/Analog Types")

            ' Create list of node descriptors, one for each leaf obtained
            ' filter out hint leafs that do not represent real OPC items (rare)
            Dim nodeDescriptorArray() As DANodeDescriptor = nodeElementCollection _
                .Where(Function(element) Not element.IsHint) _
                .Select(Function(element) New DANodeDescriptor(element)) _
                .ToArray()

            ' Get the value of DataType property; it is a 16-bit signed integer
            Dim valueResultArray() As ValueResult =
client.GetMultiplePropertyValues(serverDescriptor,
                    nodeDescriptorArray, DAPropertyIds.DataType)

            For i = 0 To valueResultArray.Length - 1
                Dim nodeDescriptor = nodeDescriptorArray(i)

                ' Check if there has been an error getting the property value
                Dim valueResult As ValueResult = valueResultArray(i)
                If valueResult.Exception IsNot Nothing Then
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message)
                    Continue For
                End If

                ' Convert the data type to VarType
                Dim varType = CType(CShort(valueResult.Value), VarType)
```

```
                ' Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType)
            Next i
        End Sub
    End Class
End Namespace
```

5.1.2.1.3 Reading Attributes of OPC UA Nodes

In OPC UA Data Access, reading data from attributes of OPC nodes is one of the most common tasks. The OPC server generally provides data for any OPC attribute in form of a Value, Timestamps and Status Code combination (see chapter "OPC Attribute Data").

In This Topic

A single node and attribute

If you want to read the current attribute data from an attribute of an OPC node, call the Read method. You pass in individual arguments for the endpoint descriptor and the node ID. You will receive back a UAAttributeData object holding the current value, timestamps, and status code for the specified OPC node and attribute. The Read method returns the current attribute data, regardless of the actual status code. You may receive an Uncertain or even Bad status (and no usable data value), and your code needs to deal with such situations accordingly.

C#

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class Read
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain attribute data. By default, the Value attribute of a node will be
read.
            UAAttributeData attributeData;
```

```
            try
            {
                attributeData = client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);

            // Example output:
            //
            //Value: -2.230064E-31
            //ServerTimestamp: 11/6/2011 1:34:30 PM
            //SourceTimestamp: 11/6/2011 1:34:30 PM
            //StatusCode: Good
        }
    }
}
```

VB.NET

```
' This example shows how to read and display data of an attribute (value, timestamps,
and status code).

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class Read
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain attribute data. By default, the Value attribute of a node will be
read.
            Dim attributeData As UAAttributeData
            Try
                attributeData = client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
```

```
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)

            ' Example output:
            '
            'Value: -2.230064E-31
            'ServerTimestamp: 11/6/2011 1:34:30 PM
            'SourceTimestamp: 11/6/2011 1:34:30 PM
            'StatusCode: Good
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "Read.h"

namespace _EasyUAClient
{
    void Read::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Obtain attribute data. By default, the Value attribute of a node will be
read.
            _UAAttributeDataPtr AttributeDataPtr = ClientPtr->Read(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10853"); // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"

            // Display results
            _variant_t vString;
            vString.ChangeType(VT_BSTR, &AttributeDataPtr->Value);
            _tprintf(_T("Value: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            vString.ChangeType(VT_BSTR, &_variant_t(AttributeDataPtr->ServerTimestamp,
VT_DATE));
            _tprintf(_T("ServerTimestamp: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            vString.ChangeType(VT_BSTR, &_variant_t(AttributeDataPtr->SourceTimestamp,
VT_DATE));
            _tprintf(_T("SourceTimestamp: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            _tprintf(_T("StatusCode: %s\n"), (LPCTSTR)CW2CT(AttributeDataPtr-
```

```
>StatusCode->ToString));

            // Example output:
            //
            //Value: -2.230064E-31
            //ServerTimestamp: 11/6/2011 1:34:30 PM
            //SourceTimestamp: 11/6/2011 1:34:30 PM
            //StatusCode: Good

        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

PHP

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain attribute data. By default, the Value attribute of a node will be read.
try
{
    $AttributeData = $Client->Read(
        "http://opcua.demo-this.com:51211/UA/SampleServer",
        "nsu=http://test.org/UA/Data/;i=10853"); // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display results
printf("Value: %s\n", $AttributeData->Value);
printf("ServerTimestamp: %s\n", $AttributeData->ServerTimestamp);
printf("SourceTimestamp: %s\n", $AttributeData->SourceTimestamp);
printf("StatusCode: %s\n", $AttributeData->StatusCode);

// Example output:
//
//Value: -2.230064E-31
//ServerTimestamp: 11/6/2011 1:34:30 PM
//SourceTimestamp: 11/6/2011 1:34:30 PM
//StatusCode: Good
```

PowerScript

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

mle_outputtext.Text = ""
```

```
// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Obtain attribute data. By default, the Value attribute of a node will be read.
OLEObject attributeData
TRY
    attributeData = client.Read("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")        // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + "Value: " + String(attributeData.value) +
"~r~n"
mle_outputtext.Text = mle_outputtext.Text + "ServerTimestamp: " +
String(attributeData.ServerTimestamp) + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "SourceTimestamp: " +
String(attributeData.SourceTimestamp) + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "StatusCode: " +
String(attributeData.StatusCode) + "~r~n"

// Example output:
//
//Value: -2.230064E-31
//ServerTimestamp: 11/6/2011 1:34:30 PM
//SourceTimestamp: 11/6/2011 1:34:30 PM
//StatusCode: Good
```

Python

```
# This example shows how to read and display data of an attribute (value, timestamps,
and status code).

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Obtain attribute data. By default, the Value attribute of a node will be read.
attributeData = client.Read('http://opcua.demo-this.com:51211/UA/SampleServer', # or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
                            'nsu=http://test.org/UA/Data/;i=10853')

# Display results
print('Value: ', attributeData.Value)
print('ServerTimestamp: ', attributeData.ServerTimestamp)
print('SourceTimestamp: ', attributeData.SourceTimestamp)
print('StatusCode: ', attributeData.StatusCode)
```

```
# Example output:
#
#Value: -2.230064E-31
#ServerTimestamp: 11/6/2011 1:34:30 PM
#SourceTimestamp: 11/6/2011 1:34:30 PM
#StatusCode: Good
```

Visual Basic (VB 6.)

```
Rem This example shows how to read and display data of an attribute (value, timestamps,
and status code).

Private Sub Read_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Obtain attribute data. By default, the Value attribute of a node will be read.
    On Error Resume Next
    Dim AttributeData As UAAttributeData
    Set AttributeData = Client.Read("http://opcua.demo-this.com:51211/UA/SampleServer",
_
                                    "nsu=http://test.org/UA/Data/;i=10853") ' or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    OutputText = OutputText & "Value: " & AttributeData.value & vbCrLf
    OutputText = OutputText & "ServerTimestamp: " & AttributeData.ServerTimestamp &
vbCrLf
    OutputText = OutputText & "SourceTimestamp: " & AttributeData.SourceTimestamp &
vbCrLf
    OutputText = OutputText & "StatusCode: " & AttributeData.StatusCode & vbCrLf

    ' Example output:
    '
    'Value: -2.230064E-31
    'ServerTimestamp: 11/6/2011 1:34:30 PM
    'SourceTimestamp: 11/6/2011 1:34:30 PM
    'StatusCode: Good
End Sub
```

VBScript

```
Rem This example shows how to read and display data of an attribute (value, timestamps,
and status code).

Option Explicit

' Instantiate the client object
```

```
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain attribute data. By default, the Value attribute of a node will be read.
On Error Resume Next
Dim AttributeData: Set AttributeData = Client.Read("http://opcua.demo-
this.com:51211/UA/SampleServer", _

"nsu=http://test.org/UA/Data/;i=10853") ' or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Value: " & AttributeData.Value
WScript.Echo "ServerTimestamp: " & AttributeData.ServerTimestamp
WScript.Echo "SourceTimestamp: " & AttributeData.SourceTimestamp
WScript.Echo "StatusCode: " & AttributeData.StatusCode

' Example output:
'
'Value: -2.230064E-31
'ServerTimestamp: 11/6/2011 1:34:30 PM
'SourceTimestamp: 11/6/2011 1:34:30 PM
'StatusCode: Good
```

If the attribute ID is not specified in the method call, the method will read from the **Value** attribute. There are also other overloads of the Read method that allow you to pass in the arguments in a different way, and with more information in them. For example, you can pass in a UANodeArguments object, and an attribute ID.

Multiple nodes or attributes

For reading attribute data of multiple attributes (of the same node or of different nodes) simultaneously in an efficient manner, call the ReadMultiple method (instead of multiple Read calls in a loop). You pass in an array of UAReadArguments objects, and you will receive back an array of UAAttributeDataResult objects.

C#

```
// This example shows how to read data (value, timestamps, and status code) of 3
attributes at once. In this example,
// we are reading a Value attribute of 3 different nodes, but the method can also be
used to read multiple attributes
// of the same node.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
```

```
{
    partial class ReadMultiple
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain attribute data. By default, the Value attributes of the nodes
will be read.
            UAAttributeDataResult[] attributeDataResultArray =
client.ReadMultiple(new[]
                {
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845"),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853"),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855")
                });

            // Display results
            foreach (UAAttributeDataResult attributeDataResult in
attributeDataResultArray)
            {
                if (attributeDataResult.Succeeded)
                    Console.WriteLine("AttributeData: {0}",
attributeDataResult.AttributeData);
                else
                    Console.WriteLine("*** Failure: {0}",
attributeDataResult.ErrorMessageBrief);
            }
        }

        // Example output:
        //
        //AttributeData: 51 {System.Int16} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM;
Good
        //AttributeData: -1993984 {System.Single} @11/6/2011 1:49:19 PM @11/6/2011
1:49:19 PM; Good
        //AttributeData: Yellow% Dragon Cat) White Blue Dog# Green Banana-
{System.String} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
    }
}
```

VB.NET

```
' This example shows how to read data (value, timestamps, and status code) of 3
attributes at once. In this example,
' we are reading a Value attribute of 3 different nodes, but the method can also be
used to read multiple attributes
```

```
' of the same node.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultiple
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain attribute data. By default, the Value attributes of the nodes will
be read.
            Dim attributeDataResultArray() As UAAttributeDataResult =
client.ReadMultiple(New UAReadArguments() _
                {
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845"),
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853"),
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855")
                })

            ' Display results
            For Each attributeDataResult As UAAttributeDataResult In
attributeDataResultArray
                If attributeDataResult.Succeeded Then
                    Console.WriteLine("AttributeData: {0}",
attributeDataResult.AttributeData)
                Else
                    Console.WriteLine("*** Failure: {0}",
attributeDataResult.ErrorMessageBrief)
                End If
            Next attributeDataResult

            ' Example output:
            '
            'AttributeData: 51 {System.Int16} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19
PM; Good
            'AttributeData: -1993984 {System.Single} @11/6/2011 1:49:19 PM @11/6/2011
1:49:19 PM; Good
            'AttributeData: Yellow% Dragon Cat) White Blue Dog# Green Banana-
{System.String} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read the attributes of 4 OPC-UA nodes at once, and display
the results.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "ReadMultiple.h"

namespace _EasyUAClient
{
    void ReadMultiple::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            _UAReadArgumentsPtr ReadArguments1Ptr(__uuidof(UAReadArguments));
            ReadArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10853";

            _UAReadArgumentsPtr ReadArguments2Ptr(__uuidof(UAReadArguments));
            ReadArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10845";

            _UAReadArgumentsPtr ReadArguments3Ptr(__uuidof(UAReadArguments));
            ReadArguments3Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10304";

            _UAReadArgumentsPtr ReadArguments4Ptr(__uuidof(UAReadArguments));
            ReadArguments4Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments4Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10389";

            CComSafeArray<VARIANT> arguments(4);
            arguments.SetAt(0, _variant_t((IDispatch*)ReadArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)ReadArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)ReadArguments3Ptr));
            arguments.SetAt(3, _variant_t((IDispatch*)ReadArguments4Ptr));
            CComVariant vArguments(arguments);

            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Obtain values. By default, the Value attributes of the nodes will be
read.
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->ReadMultiple(&vArguments));

            // Display results
```

```
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _UAAttributeDataResultPtr ResultPtr = results[i];

                _variant_t vString;
                vString.ChangeType(VT_BSTR, &_variant_t((IDispatch*)ResultPtr-
>AttributeData));
                _tprintf(_T("results(%d).AttributeData: %s\n"), i,
(LPCTSTR)CW2CT((_bstr_t)vString));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

PHP

```
// This example shows how to read the attributes of 4 OPC-UA nodes at once, and display
the results.

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";

$ReadArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";

$ReadArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10304";

$ReadArguments4 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments4->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments4->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10389";

$arguments[0] = $ReadArguments1;
$arguments[1] = $ReadArguments2;
$arguments[2] = $ReadArguments3;
$arguments[3] = $ReadArguments4;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
$results = $Client->ReadMultiple($arguments);

// Display results
```

```
for ($i = 0; $i < count($results); $i++)
{
    $attributeDataResult = $results[$i];
    if ($attributeDataResult->Succeeded)
        printf("results[%d].AttributeData: %s\n", $i, $attributeDataResult-
>AttributeData);
    else
        printf("results[%d]: *** Failure: %s\n", $i, $attributeDataResult-
>ErrorMessageBrief);
}
```

PowerScript

```
// This example shows how to read data (value, timestamps, and status code) of 3
attributes at once. In this example,
// we are reading a Value attribute of 3 different nodes, but the method can also be
used to read multiple attributes
// of the same node.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments. By default, the Value attributes of the nodes will be read.

OLEObject readArguments1
readArguments1 = CREATE OLEObject
readArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
readArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

OLEObject readArguments2
readArguments2 = CREATE OLEObject
readArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
readArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

OLEObject readArguments3
readArguments3 = CREATE OLEObject
readArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
readArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"

OLEObject readArgumentsList
readArgumentsList = CREATE OLEObject
readArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readArgumentsList.Add(readArguments1)
readArgumentsList.Add(readArguments2)
```

```
readArgumentsList.Add(readArguments3)

// Obtain attribute data.
OLEObject attributeDataResultList
attributeDataResultList = client.ReadList(readArgumentsList)

// Display results
Int i
FOR i = 0 TO attributeDataResultList.Count - 1
    OLEObject attributeDataResult
    attributeDataResult = attributeDataResultList.Item[i]
    IF attributeDataResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "AttributeData: " +
String(attributeDataResult.AttributeData) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
attributeDataResult.ErrorMessageBrief + "~r~n"
    END IF
NEXT

// Example output:
//
//AttributeData: 51 {System.Int16} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
//AttributeData: -1993984 {System.Single} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM;
Good
//AttributeData: Yellow% Dragon Cat) White Blue Dog# Green Banana- {System.String}
@11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
```

Visual Basic (VB 6.)

```
Rem This example shows how to read the attributes of 4 OPC-UA nodes at once, and
display the results.

Private Sub ReadMultiple_Main_Command_Click()
    OutputText = ""

    Dim ReadArguments1 As New UAReadArguments
    ReadArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"

    Dim ReadArguments2 As New UAReadArguments
    ReadArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"

    Dim ReadArguments3 As New UAReadArguments
    ReadArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10304"

    Dim ReadArguments4 As New UAReadArguments
    ReadArguments4.endpointDescriptor.UrlString = "http://opcua.demo-
```

```
this.com:51211/UA/SampleServer"
    ReadArguments4.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10389"

    Dim arguments(3) As Variant
    Set arguments(0) = ReadArguments1
    Set arguments(1) = ReadArguments2
    Set arguments(2) = ReadArguments3
    Set arguments(3) = ReadArguments4

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Obtain values. By default, the Value attributes of the nodes will be read.
    Dim results() As Variant
    results = Client.ReadMultiple(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        Dim Result As UAAttributeDataResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "results(" & i & ").AttributeData: " &
Result.AttributeData & vbCrLf
        Else
            OutputText = OutputText & "results(" & i & ") *** Failure: " &
Result.ErrorMessageBrief & vbCrLf
        End If
    Next
End Sub
```

VBScript

```
Rem This example shows how to read the attributes of 4 OPC-UA nodes at once, and
display the results.

Option Explicit

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10304"
```

```
Dim ReadArguments4: Set ReadArguments4 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments4.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments4.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10389"

Dim arguments(3)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3
Set arguments(3) = ReadArguments4

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
Dim results: results = Client.ReadMultiple(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim AttributeDataResult: Set AttributeDataResult = results(i)
    If AttributeDataResult.Succeeded Then
        WScript.Echo "results(" & i & ").AttributeData: " &
AttributeDataResult.AttributeData
    Else
        WScript.Echo "results(" & i & ") *** Failure: " &
AttributeDataResult.ErrorMessageBrief
    End If
Next
```

Read parameters

It is possible to specify the maximum age of the value to be read. In order to do that, set the MaximumAge property of UAReadArguments.ReadParameters object passed to the read method, to the appropriate number of milliseconds. By default, this property is set to UAReadParameters.FromCache, which denotes reading from the cache.

Note: OPC “Classic” has separate functions for reading so-called OPC properties. OPC properties contain additional information related to a node. In OPC Unified Architecture, properties are accessed in the same way as other information. That is, if you have a node ID (obtain e.g. by browsing) of a property, you can use one of the ReadXXXX methods described in this chapter to get a value of that property.

Using browse paths

C#

```
// This example shows how to read the attributes of 4 OPC-UA nodes specified by browse
paths at once, and display the
// results.
```

```
using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultiple
    {
        public static void BrowsePath()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            var browsePathParser = new UABrowsePathParser {DefaultNamespaceUriString =
"http://test.org/UA/Data/"};

            // Prepare arguments
            // Note: Add error handling around the following statement if the browse
paths are not guaranteed to be
            // syntactically valid.
            var readArgumentsArray = new[]
            {
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue")),
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue")),
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/Array/UInt16Value")),
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar/Int32Value"))
            };

            // Obtain attribute data.
            UAAttributeDataResult[] resultArray =
client.ReadMultiple(readArgumentsArray);

            // Display results
            for (int i = 0; i < resultArray.Length; i++)
            {
                UAAttributeDataResult attributeDataResult = resultArray[i];
                if (attributeDataResult.Succeeded)
                    Console.WriteLine($"results[{i}].AttributeData:
{attributeDataResult.AttributeData}");
                else
                    Console.WriteLine($"results[{i}]: *** Failure:
```

```
{attributeDataResult.ErrorMessageBrief}");
            }
        }

        // Example output:
        //results[0].AttributeData: 4.187603E+21 {System.Single} @2019-11-
09T14:05:46.268 @@2019-11-09T14:05:46.268; Good
        //results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268 @@2019-
11-09T14:05:46.268; Good
        //results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
        //results[3].AttributeData: 1280120396 {System.Int32} @2019-11-09T14:00:07.590
@@2019-11-09T14:05:46.268; Good
    }
}
```

VB.NET

```
' This example shows how to read the attributes of 4 OPC-UA nodes specified by browse
paths at once, and display the
' results.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultiple
        Public Shared Sub BrowsePath()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Dim browsePathParser = New UABrowsePathParser()
            browsePathParser.DefaultNamespaceUriString = "http://test.org/UA/Data/"

            ' Prepare arguments
            ' Note: Add error handling around the following statement if the browse
paths are Not guaranteed to be
            ' syntactically valid.
            Dim readArgumentsArray = New UAReadArguments() _
            {
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue")),
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue")),
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
```

```
[ObjectsFolder]/Data/Static/Array/UInt16Value")),
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar/Int32Value"))
            }


            ' Obtain attribute data. By default, the Value attributes of the nodes will
be read.
            Dim resultArray() As UAAttributeDataResult =
client.ReadMultiple(readArgumentsArray)

            ' Display results
            For i As Integer = 0 To resultArray.Length - 1
                Dim attributeDataResult As UAAttributeDataResult = resultArray(i)
                If attributeDataResult.Succeeded Then
                    Console.WriteLine("results[{0}].AttributeData: {1}", i,
attributeDataResult.AttributeData)
                Else
                    Console.WriteLine("results[{0}]: *** Failure: {1}", i,
attributeDataResult.ErrorMessageBrief)
                End If
            Next i

            ' Example output:
            'results[0].AttributeData 4.187603E+21 {System.Single} @2019-11-
09T14:05:46.268 @@2019-11-09T14:05:46.268; Good
            'results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268
@@2019-11-09T14:05:46.268; Good
            'results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
            'results[3].AttributeData: 1280120396 {System.Int32} @2019-11-
09T14:00:07.590 @@2019-11-09T14:05:46.268; Good
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to read the attributes of 4 OPC-UA nodes specified
// by browse paths at once, and display the results.

class procedure ReadMultiple.BrowsePath;
var
  Arguments: OleVariant;
  BrowsePathParser: _UABrowsePathParser;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  ReadArguments1, ReadArguments2, ReadArguments3, ReadArguments4: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  BrowsePathParser.DefaultNamespaceUriString := 'http://test.org/UA/Data/';

  ReadArguments1 := CoUAReadArguments.Create;
  ReadArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
```

```
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments1.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue');

  ReadArguments2 := CoUAReadArguments.Create;
  ReadArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments2.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue');

  ReadArguments3 := CoUAReadArguments.Create;
  ReadArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments3.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Static/Array/UInt16Value');

  ReadArguments4 := CoUAReadArguments.Create;
  ReadArguments4.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments4.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Static/UserScalar/Int32Value');

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ReadArguments1;
  Arguments[1] := ReadArguments2;
  Arguments[2] := ReadArguments3;
  Arguments[3] := ReadArguments4;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    Result := IInterface(Results[I]) as _UAAttributeDataResult;
    if Result.Succeeded then
      WriteLn('results[', I, '].AttributeData: ', Result.AttributeData.ToString)
    else
      WriteLn('results[', I, '] *** Failure: ', Result.ErrorMessageBrief);
  end;

  // Example output:
  //results[0].AttributeData: 4.187603E+21 {System.Single} @2019-11-09T14:05:46.268
@@2019-11-09T14:05:46.268; Good
  //results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268 @@2019-11-
```

```
09T14:05:46.268; Good
  //results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
  //results[3].AttributeData: 1280120396 {System.Int32} @2019-11-09T14:00:07.590
@@2019-11-09T14:05:46.268; Good

end;
```

VBScript

```
Rem This example shows how to read the attributes of 4 OPC-UA nodes specified by browse
paths at once, and display the
Rem results.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
BrowsePathParser.DefaultNamespaceUriString = "http://test.org/UA/Data/"

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments1.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue")

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments2.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue")

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments3.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/Array/UInt16Value")

Dim ReadArguments4: Set ReadArguments4 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments4.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments4.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar/Int32Value")

Dim arguments(3)
```

```
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3
Set arguments(3) = ReadArguments4

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
Dim results: results = Client.ReadMultiple(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim AttributeDataResult: Set AttributeDataResult = results(i)
    If AttributeDataResult.Succeeded Then
        WScript.Echo "results[" & i & "].AttributeData: " &
AttributeDataResult.AttributeData
    Else
        WScript.Echo "results[" & i & "] *** Failure: " &
AttributeDataResult.ErrorMessageBrief
    End If
Next

' Example output:
'results[0].AttributeData: 4.187603E+21 {System.Single} @2019-11-09T14:05:46.268
@@2019-11-09T14:05:46.268; Good
'results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268 @@2019-11-
09T14:05:46.268; Good
'results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
'results[3].AttributeData: 1280120396 {System.Int32} @2019-11-09T14:00:07.590 @@2019-
11-09T14:05:46.268; Good
```

5.1.2.1.3.1 Reading just the value (OPC UA)

Some applications need the actual data value for further processing (e.g. for computations that need be performed on the values), i.e. the status code must be Good and a valid value must be provided by the server, otherwise it is considered an error.

In This Topic

A single node and attribute

For such usage, call the ReadValue method, passing it the same arguments as to the Read method. The method will read the attribute data, check if the status is Good, and you will receive back an Object holding the actual data value. If the status code is not Good, the method will throw a UAStatusCodeException.

C#

```
// This example shows how to read value of a single node, and display it.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples._EasyUAClient
{
    partial class ReadValue
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain value of a node
            object value;
            try
            {
                value = client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("value: {0}", value);
        }
    }
}
```

VB.NET

```
' This example shows how to read value of a single node, and display it.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadValue
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain value of a node
            Dim value As Object
            Try
                value = client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
```

```
            Console.WriteLine("value: {0}", value)
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read value of a single node, and display it.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ReadValue.h"

namespace _EasyUAClient
{
    void ReadValue::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            _variant_t value = ClientPtr->ReadValue(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10853");

            // Display results
            _variant_t vString;
            vString.ChangeType(VT_BSTR, &value);
            _tprintf(_T("value: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

JScript

```
// This example shows how to read value of a single node, and display it.
var Client = new ActiveXObject("OpcLabs.EasyOpc.UA.EasyUAClient");
var value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
WScript.Echo("value: " + value);
```

Free Pascal

```
// This example shows how to read value of a single node, and display it.

class procedure ReadValue.Main;
var
  Client: EasyUAClient;
  Value: OleVariant;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  Value := Client.ReadValue(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853');
  WriteLn('value: ', Value);
end;
```

Object Pascal

```
// This example shows how to read value of a single node, and display it.

class procedure ReadValue.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Value: OleVariant;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain value of a node
  try
    Value := Client.ReadValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('value: ', Value);
end;
```

PHP

```
// This example shows how to read value of a single node, and display it.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}


// Display results
printf("Value: %s\n", $value);
```

PowerScript

```
// This example shows how to read value of a single node, and display it.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")
```

```
// Obtain value of a node
mle_outputtext.Text = mle_outputtext.Text + "Reading node value..." + "~r~n"
Any value
TRY
    value = client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + oleRuntimeError.Description +
"~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + String(value) + "~r~n"
```

Python

```
# This example shows how to read value of a single node, and display it.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Perform the operation
value = client.ReadValue('http://opcua.demo-this.com:51211/UA/SampleServer',
'nsu=http://test.org/UA/Data/;i=10853')

# Display results
print('value: ', value)
```

Visual Basic (VB 6.)

```
Rem This example shows how to read value of a single node, and display it.

Private Sub ReadValue_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Dim value As Variant
    value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description & vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    OutputText = OutputText & "value: " & value & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to read value of a single node, and display it.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
```

```
' Perform the operation
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "value: " & value
```

C#

```
// This example shows how to read a value of a specific attribute of a single node, and display it.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadValue
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            UANodeDescriptor nodeDescriptor = "nsu=http://test.org/UA/Data/;i=10853";

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain value of a DataType attribute
            object value;
            try
            {
                value = client.ReadValue(endpointDescriptor, nodeDescriptor,
UAAttributeId.DataType);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("value type: {0}", value?.GetType());
            Console.WriteLine("value: {0}", value);
        }
    }
}
```

VB.NET

```
' This example shows how to read a value of a specific attribute of a single node, and display it.
```

```
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadValue
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Dim nodeDescriptor As UANodeDescriptor = "nsu=http://test.org/UA/Data/;i=10853"

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            ' Obtain value of a DataType attribute.
            Dim value As Object
            Try
                value = client.ReadValue(endpointDescriptor, nodeDescriptor,
UAAttributeId.DataType)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("value type: {0}", value.GetType())
            Console.WriteLine("value: {0}", value)
        End Sub
    End Class
End Namespace
```

Multiple nodes or attributes

For reading just the data values of multiple data values (with Good status) in an efficient manner, call the ReadMultipleValues method (instead of multiple ReadValue calls in a loop). You pass in an array of UAReadArguments objects, and you will receive back an array of ValueResult objects.

C#

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultipleValues
    {
        public static void Main1()
        {
```

```
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain values. By default, the Value attributes of the nodes will be read.
            ValueResult[] valueResultArray = client.ReadMultipleValues(new[]
                {
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845"),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853"),
                    new UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10855")
                });

            // Display results
            foreach (ValueResult valueResult in valueResultArray)
            {
                if (valueResult.Succeeded)
                    Console.WriteLine("Value: {0}", valueResult.Value);
                else
                    Console.WriteLine("*** Failure: {0}", valueResult.ErrorMessageBrief);
            }

            // Example output:
            //
            //Value: 8
            //Value: -8.06803E+21
            //Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
        }
    }
}
```

VB.NET

```
' This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
' to read multiple attributes of the same node.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultipleValues
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain values. By default, the Value attributes of the nodes will be read.
            Dim valueResultArray() As ValueResult = client.ReadMultipleValues(New UAReadArguments()
_
                {
```

```
                New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10845"),
                New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853"),
                New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10855")
            }
        )

        ' Display results
        For Each valueResult As ValueResult In valueResultArray
            If valueResult.Succeeded Then
                Console.WriteLine("Value: {0}", valueResult.Value)
            Else
                Console.WriteLine("*** Failure: {0}", valueResult.ErrorMessageBrief)
            End If
        Next valueResult

        ' Example output:
        '
        'Value: 8
        'Value: -8.06803E+21
        'Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
    End Sub
End Class
End Namespace
```

C++

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "ReadMultipleValues.h"

namespace _EasyUAClient
{
    void ReadMultipleValues::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            _UAReadArgumentsPtr ReadArguments1Ptr(_uuidof(UAReadArguments));
            ReadArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10845";

            _UAReadArgumentsPtr ReadArguments2Ptr(_uuidof(UAReadArguments));
            ReadArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10853";

            _UAReadArgumentsPtr ReadArguments3Ptr(_uuidof(UAReadArguments));
            ReadArguments3Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10855";

            CComSafeArray<VARIANT> arguments(3);
            arguments.SetAt(0, _variant_t((IDispatch*)ReadArguments1Ptr));
```

```
            arguments.SetAt(1, _variant_t((IDispatch*)ReadArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)ReadArguments3Ptr));
            CComVariant vArguments(arguments);

            // Obtain values. By default, the Value attributes of the nodes will be read.
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->ReadMultipleValues(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _ValueResultPtr ValueResultPtr = results[i];

                _variant_t vString;
                vString.ChangeType(VT_BSTR, &ValueResultPtr->Value);
                _tprintf(_T("Value: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            }

            // Example output:
            //
            //Value: 8
            //Value: -8.06803E+21
            //Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

PHP

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10845";

$ReadArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments2->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";

$ReadArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments3->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10855";

$arguments[0] = $ReadArguments1;
$arguments[1] = $ReadArguments2;
$arguments[2] = $ReadArguments3;

// Obtain values. By default, the Value attributes of the nodes will be read.
$results = $Client->ReadMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $ValueResult = $results[$i];
```

```
    if ($ValueResult->Succeeded)
        printf("Value: %s\n", $ValueResult->Value);
    else
        printf("*** Failure: %s\n", $ValueResult->ErrorMessageBrief);
}

// Example output:
//
//Value: 8
//Value: -8.06803E+21
//Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

PowerScript

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments. By default, the Value attributes of the nodes will be read.

OLEObject readArguments1
readArguments1 = CREATE OLEObject
readArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10845"

OLEObject readArguments2
readArguments2 = CREATE OLEObject
readArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"

OLEObject readArguments3
readArguments3 = CREATE OLEObject
readArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments3.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10855"

OLEObject readArgumentsList
readArgumentsList = CREATE OLEObject
readArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readArgumentsList.Add(readArguments1)
readArgumentsList.Add(readArguments2)
readArgumentsList.Add(readArguments3)

// Obtain values.
OLEObject valueResultList
valueResultList = client.ReadValueList(readArgumentsList)

// Display results
Int i
FOR i = 0 TO valueResultList.Count - 1
    OLEObject valueResult
    valueResult = valueResultList.Item[i]
    IF valueResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Value: " + String(valueResult.Value) + "~r~n"
```

```
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + valueResult.ErrorMessageBrief
+ "~r~n"
    END IF
NEXT

// Example output:
//
//Value: 8
//Value: -8.06803E+21
//Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

Visual Basic (VB 6.)

```
Rem This example shows how to read the Value attributes of 3 different nodes at once. Using the
same method, it is also possible
Rem to read multiple attributes of the same node.

Private Sub ReadMultipleValues_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    Dim ReadArguments1 As New UAReadArguments
    ReadArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments1.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10845"

    Dim ReadArguments2 As New UAReadArguments
    ReadArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments2.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10853"

    Dim ReadArguments3 As New UAReadArguments
    ReadArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments3.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10855"

    Dim arguments(2) As Variant
    Set arguments(0) = ReadArguments1
    Set arguments(1) = ReadArguments2
    Set arguments(2) = ReadArguments3

    ' Obtain values. By default, the Value attributes of the nodes will be read.
    Dim results() As Variant
    results = Client.ReadMultipleValues(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        Dim Result As ValueResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "Value: " & Result.value & vbCrLf
        Else
            OutputText = OutputText & "*** Failure: " & Result.ErrorMessageBrief & vbCrLf
        End If
    Next

    ' Example output:
    '
    'Value: 8
    'Value: -8.06803E+21
    'Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

```
End Sub
```

VBScript

```
Rem This example shows how to read the Value attributes of 3 different nodes at once. Using the
same method, it is also possible
Rem to read multiple attributes of the same node.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10845"

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments3.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10855"

Dim arguments(2)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3

' Obtain values. By default, the Value attributes of the nodes will be read.
Dim results: results = Client.ReadMultipleValues(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim ValueResult: Set ValueResult = results(i)
    If ValueResult.Succeeded Then
        WScript.Echo "Value: " & ValueResult.Value
    Else
        WScript.Echo "*** Failure: " & ValueResult.ErrorMessageBrief
    End If
Next

' Example output:
'
'Value: 8
'Value: -8.06803E+21
'Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

Reading DataType attributes

C#

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
```

```
// to read multiple attributes of the same node.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultipleValues
    {
        public static void DataType()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain values.
            ValueResult[] valueResultArray = client.ReadMultipleValues(new[]
                {
                    new UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10845",
UAAttributeId.DataType),
                    new UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853",
UAAttributeId.DataType),
                    new UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10855",
UAAttributeId.DataType)
                });

            // Display results
            foreach (ValueResult valueResult in valueResultArray)
            {
                Console.WriteLine();

                if (valueResult.Succeeded)
                {
                    Console.WriteLine($"Value: {valueResult.Value}");
                    var dataTypeId = valueResult.Value as UANodeId;
                    if (!(dataTypeId is null))
                    {
                        Console.WriteLine($"Value.ExpandedText: {dataTypeId.ExpandedText}");
                        Console.WriteLine($"Value.NamespaceUriString:
{dataTypeId.NamespaceUriString}");
                        Console.WriteLine($"Value.NamespaceIndex: {dataTypeId.NamespaceIndex}");
                        Console.WriteLine($"Value.NumericIdentifier:
{dataTypeId.NumericIdentifier}");
                    }
                }
                else
                    Console.WriteLine($"*** Failure: {valueResult.ErrorMessageBrief}");
            }

            // Example output:
            //
            //Value: SByte
            //Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2
            //Value.NamespaceUriString: http://opcfoundation.org/UA/
            //Value.NamespaceIndex: 0
            //Value.NumericIdentifier: 2
            //
            //Value: Float
```

```
            //Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=10
            //Value.NamespaceUriString: http://opcfoundation.org/UA/
            //Value.NamespaceIndex: 0
            //Value.NumericIdentifier: 10
            //
            //Value: String
            //Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=12
            //Value.NamespaceUriString: http://opcfoundation.org/UA/
            //Value.NamespaceIndex: 0
            //Value.NumericIdentifier: 12
        }
    }
}
```

VB.NET

```
' This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
' to read multiple attributes of the same node.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultipleValues
        Public Shared Sub DataType()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain values.
            Dim valueResultArray() As ValueResult = client.ReadMultipleValues(New UAReadArguments()
_
               {
                   New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10845",
UAAttributeId.DataType),
                   New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853",
UAAttributeId.DataType),
                   New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10855",
UAAttributeId.DataType)
               }
            )

            ' Display results
            For Each valueResult As ValueResult In valueResultArray
                Console.WriteLine()

                If valueResult.Succeeded Then
                    Console.WriteLine("Value: {0}", valueResult.Value)
                    Dim dataTypeId = CType(valueResult.Value, UANodeId)
                    If Not dataTypeId Is Nothing Then
                        Console.WriteLine("Value.ExpandedText: {0}", dataTypeId.ExpandedText)
                        Console.WriteLine("Value.NamespaceUriString: {0}",
dataTypeId.NamespaceUriString)
                        Console.WriteLine("Value.NamespaceIndex: {0}", dataTypeId.NamespaceIndex)
```

```
                    Console.WriteLine("Value.NumericIdentifier: {0}",
dataTypeId.NumericIdentifier)
                End If
            Else
                Console.WriteLine("*** Failure: {0}", valueResult.ErrorMessageBrief)
            End If
        Next valueResult

        ' Example output:
        '
        'Value: SByte
        'Value.ExpandedText: nsu = http : //opcfoundation.org/UA/ ;i=2
        'Value.NamespaceUriString: http : //opcfoundation.org/UA/
        'Value.NamespaceIndex: 0
        'Value.NumericIdentifier: 2
        '
        'Value: Float
        'Value.ExpandedText: nsu = http : //opcfoundation.org/UA/ ;i=10
        'Value.NamespaceUriString: http : //opcfoundation.org/UA/
        'Value.NamespaceIndex: 0
        'Value.NumericIdentifier: 10
        '
        'Value: String
        'Value.ExpandedText: nsu = http : //opcfoundation.org/UA/ ;i=12
        'Value.NamespaceUriString: http : //opcfoundation.org/UA/
        'Value.NamespaceIndex: 0
        'Value.NumericIdentifier: 12
    End Sub
End Class
End Namespace
```

PHP

```
// This example shows how to read the DataType attributes of 3 different nodes at
// once. Using the same method, it is also possible to read multiple attributes
// of the same node.

const UAAttributeId_DataType = 14;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10845";
$ReadArguments1->AttributeId = UAAttributeId_DataType;

$ReadArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments2->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";
$ReadArguments2->AttributeId = UAAttributeId_DataType;

$ReadArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments3->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10855";
$ReadArguments3->AttributeId = UAAttributeId_DataType;

$arguments[0] = $ReadArguments1;
$arguments[1] = $ReadArguments2;
$arguments[2] = $ReadArguments3;
```

```
// Obtain values. By default, the Value attributes of the nodes will be read.
$results = $Client->ReadMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $ValueResult = $results[$i];
    printf("\n");
    if ($ValueResult->Succeeded)
    {
        printf("Value: %s\n", $ValueResult->Value);
        printf("Value.ExpandedText: %s\n", $ValueResult->Value->ExpandedText);
        printf("Value.NamespaceUriString: %s\n", $ValueResult->Value->NamespaceUriString);
        printf("Value.NamespaceIndex: %s\n", $ValueResult->Value->NamespaceIndex);
        printf("Value.NumericIdentifier: %s\n", $ValueResult->Value->NumericIdentifier);
    }
    else
        printf("*** Failure: %s\n", $ValueResult->ErrorMessageBrief);
}

// Example output:
//
//
//Value: SByte
//Value.ExpandedText: nsu=http://opcfoundation.org/UA/;i=2
//Value.NamespaceUriString: http://opcfoundation.org/UA/
//Value.NamespaceIndex: 0
//Value.NumericIdentifier: 2
//
//Value: Float
//Value.ExpandedText: nsu=http://opcfoundation.org/UA/;i=10
//Value.NamespaceUriString: http://opcfoundation.org/UA/
//Value.NamespaceIndex: 0
//Value.NumericIdentifier: 10
//
//Value: String
//Value.ExpandedText: nsu=http://opcfoundation.org/UA/;i=12
//Value.NamespaceUriString: http://opcfoundation.org/UA/
//Value.NamespaceIndex: 0
//Value.NumericIdentifier: 12
```

Visual Basic (VB 6.)

```
Rem This example shows how to read the Value attributes of 3 different nodes at once. Using the
same method, it is also possible
Rem to read multiple attributes of the same node.

Private Sub ReadMultipleValues_DataType_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    Dim ReadArguments1 As New UAReadArguments
    ReadArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments1.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10845"
    ReadArguments1.AttributeId = UAAttributeId_DataType

    Dim ReadArguments2 As New UAReadArguments
    ReadArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments2.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10853"
```

```
    ReadArguments2.AttributeId = UAAttributeId_DataType

    Dim ReadArguments3 As New UAReadArguments
    ReadArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments3.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10855"
    ReadArguments3.AttributeId = UAAttributeId_DataType

    Dim arguments(2) As Variant
    Set arguments(0) = ReadArguments1
    Set arguments(1) = ReadArguments2
    Set arguments(2) = ReadArguments3

    ' Obtain values.
    Dim results() As Variant
    results = Client.ReadMultipleValues(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        OutputText = OutputText & vbCrLf

        Dim Result As ValueResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "Value: " & Result.value & vbCrLf
            On Error Resume Next
            OutputText = OutputText & "Value.ExpandedText: " & Result.value.expandedText & vbCrLf
            OutputText = OutputText & "Value.NamespaceUriString: " &
Result.value.NamespaceUriString & vbCrLf
            OutputText = OutputText & "Value.NamespaceIndex: " & Result.value.NamespaceIndex &
vbCrLf
            OutputText = OutputText & "Value.NumericIdentifier: " & Result.value.NumericIdentifier
& vbCrLf
            On Error GoTo 0
        Else
            OutputText = OutputText & "*** Failure: " & Result.ErrorMessageBrief & vbCrLf
        End If
    Next

    ' Example output:
    '
    'Value: SByte
    'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2
    'Value.NamespaceUriString: http://opcfoundation.org/UA/
    'Value.NamespaceIndex: 0
    'Value.NumericIdentifier: 2
    '
    'Value: Float
    'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=10
    'Value.NamespaceUriString: http://opcfoundation.org/UA/
    'Value.NamespaceIndex: 0
    'Value.NumericIdentifier: 10
    '
    'Value: String
    'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=12
    'Value.NamespaceUriString: http://opcfoundation.org/UA/
    'Value.NamespaceIndex: 0
    'Value.NumericIdentifier: 12
End Sub
```

VBScript

```
Rem This example shows how to read the DataType attributes of 3 different nodes at once. Using the
same method, it is also possible
Rem to read multiple attributes of the same node.
```

```
Option Explicit

Const UAAttributeId_DataType = 14

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10845"
ReadArguments1.AttributeId = UAAttributeId_DataType

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"
ReadArguments2.AttributeId = UAAttributeId_DataType

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments3.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10855"
ReadArguments3.AttributeId = UAAttributeId_DataType

Dim arguments(2)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3

' Obtain values.
Dim results: results = Client.ReadMultipleValues(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    WScript.Echo

    Dim ValueResult: Set ValueResult = results(i)
    If ValueResult.Succeeded Then
        WScript.Echo "Value: " & ValueResult.Value
        On Error Resume Next
        WScript.Echo "Value.ExpandedText: " & ValueResult.Value.ExpandedText
        WScript.Echo "Value.NamespaceUriString: " & ValueResult.Value.NamespaceUriString
        WScript.Echo "Value.NamespaceIndex: " & ValueResult.Value.NamespaceIndex
        WScript.Echo "Value.NumericIdentifier: " & ValueResult.Value.NumericIdentifier
        On Error Goto 0
    Else
        WScript.Echo "*** Failure: " & ValueResult.ErrorMessageBrief
    End If
Next

' Example output:
'
'Value: SByte
'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2
'Value.NamespaceUriString: http://opcfoundation.org/UA/
'Value.NamespaceIndex: 0
'Value.NumericIdentifier: 2
'
'Value: Float
'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=10
'Value.NamespaceUriString: http://opcfoundation.org/UA/
'Value.NamespaceIndex: 0
'Value.NumericIdentifier: 10
```

```
'
'Value: String
'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=12
'Value.NamespaceUriString: http://opcfoundation.org/UA/
'Value.NamespaceIndex: 0
'Value.NumericIdentifier: 12
```

5.1.2.2 Modifying Information (OPC Data)

Methods described in this chapter allow your application to modify information in the underlying data source that the OPC server connects to (writing OPC Classic items, or attributes of OPC UA nodes). It is assumed that your application already somehow knows how to identify the data it is interested in. If the location of the data is not known upfront, use methods described Browsing for Information chapter first.

5.1.2.2.1 Writing to OPC Classic Items

A single item

If you want to write a data value into a specific OPC item, call the WriteItemValue method, passing it the data value you want to write, arguments for machine name, server class, ItemID, and an optional data type.

C#

```
// This example shows how to write a value into a single item.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteItemValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            try
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message);
            }
        }
    }
}
```

PHP

```
// This example shows how to write a value into a single item.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
```

```
{
    $value = $Client->WriteItemValue("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}
```

VB.NET

```
' Shows how to write into an OPC item that is of array type, and read the array value back.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteItemValue
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to write a value into a single item.

class procedure WriteItemValue.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    Client.WriteItemValue('', 'OPCLabs.KitServer.2', 'Simulation.Register_I4', 12345);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PowerScript

```
// This example shows how to write a value into a single item.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Modify value of an item
```

```
TRY
    client.WriteItemValue("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345)
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

VBScript

```
Rem This example shows how to write a value into a single item.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Client.WriteItemValue "", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

In OPC Data Client.NET, you can also pass in the ServerDescriptor and DAItemDescriptor objects in place of corresponding individual arguments.

C#

```
// This example shows how to write a value into a single OPC XML-DA item.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class WriteItemValue
    {
        public static void Main1Xml()
        {
            var client = new EasyDAClient();

            try
            {
                client.WriteItemValue("http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
"Static/Analog Types/Int", 12345);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single OPC XML-DA item.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess.Xml
```

```
    Partial Friend Class WriteItemValue
        Public Shared Sub Main1Xml()
            Dim client = New EasyDAClient()

            Try
                client.WriteItemValue("http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
"Static/Analog Types/Int", 12345)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}", opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

Multiple items

For writing data values into multiple OPC items in an efficient manner, call the WriteMultipleItemValues method.

In OPC Data Client.NET, you pass in an array of DAItemValueArguments objects, each specifying the location of OPC item, and the value to be written.

For an efficient writing into several items, use the WriteMultipleItemValues method.

C#

```
// Shows how to write into multiple OPC items using a single method call, and read multiple item values
back.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteMultipleItemValues
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Writing multiple item values...");
            OperationResult[] resultArray = client.WriteMultipleItemValues(
                new[] {
                    new DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_I4",
12345),
                    new DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_BOOL",
true),
                    new DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_R4",
234.56)
                });


            for (int i = 0; i < resultArray.Length; i++)
            {
                Debug.Assert(resultArray[i] != null);
                if (resultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                {
                    Debug.Assert(resultArray[i].Exception != null);
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray[i].ErrorMessageBrief);
```

```
            }
        }

        Console.WriteLine("Reading multiple item values...");
        ValueResult[] valueResultArray = client.ReadMultipleItemValues("OPCLabs.KitServer.2",
            new DAItemDescriptor[] {
                    "Simulation.Register_I4",
                    "Simulation.Register_BOOL",
                    "Simulation.Register_R4" });

        for (int i = 0; i < valueResultArray.Length; i++)
        {
            Debug.Assert(valueResultArray[i] != null);
            Console.WriteLine("valueResultArray[{0}]: {1}", i, valueResultArray[i]);
        }
    }
  }
}
```

VB.NET

```
' This example shows how to write values into multiple items.

Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteMultipleItemValues
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim argumentsArray = New DAItemValueArguments() { _
                New DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345), _
                New DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_BOOL", True),
_
                New DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_R4", 234.56) _
            }

            Dim resultArray As OperationResult() = client.WriteMultipleItemValues(argumentsArray)

            For i = 0 To resultArray.Length - 1
                Debug.Assert(resultArray(i) IsNot Nothing)

                If resultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray(i).ErrorMessageBrief)
                End If
            Next i

            Console.WriteLine("Reading multiple item values...")
            Dim valueResultArray() As ValueResult =
client.ReadMultipleItemValues("OPCLabs.KitServer.2", _
                New DAItemDescriptor() { _
                    "Simulation.Register_I4",
                    "Simulation.Register_BOOL",
                    "Simulation.Register_R4"})

            For i = 0 To valueResultArray.Length - 1
                Debug.Assert(valueResultArray(i) IsNot Nothing)
                Console.WriteLine("valueResultArray[{0}]: {1}", i, valueResultArray(i))
            Next i

        End Sub
    End Class
```

```
End Namespace
```

Object Pascal

```
// This example shows how to write values into 3 items at once.

class procedure WriteMultipleItemValues.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ItemValueArguments1: _DAItemValueArguments;
  ItemValueArguments2: _DAItemValueArguments;
  ItemValueArguments3: _DAItemValueArguments;
  Results: OleVariant;
  OperationResult: _OperationResult;

begin
  ItemValueArguments1 := CoDAItemValueArguments.Create;
  ItemValueArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemValueArguments1.ItemDescriptor.ItemID := 'Simulation.Register_I4';
  ItemValueArguments1.Value := 23456;

  ItemValueArguments2 := CoDAItemValueArguments.Create;
  ItemValueArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemValueArguments2.ItemDescriptor.ItemID := 'Simulation.Register_R8';
  ItemValueArguments2.Value := 2.34567890;

  ItemValueArguments3 := CoDAItemValueArguments.Create;
  ItemValueArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemValueArguments3.ItemDescriptor.ItemID := 'Simulation.Register_BSTR';
  ItemValueArguments3.Value := 'ABC';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := ItemValueArguments1;
  Arguments[1] := ItemValueArguments2;
  Arguments[2] := ItemValueArguments3;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.WriteMultipleItemValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    OperationResult := IInterface(Results[I]) as _OperationResult;
    if OperationResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ' *** Failure: ', OperationResult.Exception.GetBaseException.Message);
  end;

end;
```

PHP

```
// This example shows how to write values into 3 items at once.

$ItemValueArguments1 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments1->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemValueArguments1->Value = 23456;
```

```
$ItemValueArguments2 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments2->ItemDescriptor->ItemID = "Simulation.Register_R8";
$ItemValueArguments2->Value = 2.34567890;

$ItemValueArguments3 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments3->ItemDescriptor->ItemID = "Simulation.Register_BSTR";
$ItemValueArguments3->Value = "ABC";

$arguments[0] = $ItemValueArguments1;
$arguments[1] = $ItemValueArguments2;
$arguments[2] = $ItemValueArguments3;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$results = $Client->WriteMultipleItemValues($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $OperationResult = $results[$i];
    if ($OperationResult->Succeeded)
        printf("Result %d: success\n", $i);
    else
        printf("Result %d: %s\n", $i, $OperationResult->ErrorMessageBrief);
}
```

PowerScript

```
// Shows how to write into multiple OPC items using a single method call, and itemValue multiple item
values back.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare arguments.

OLEObject itemValueArguments1
itemValueArguments1 = CREATE OLEObject
itemValueArguments1.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")

itemValueArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
itemValueArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
itemValueArguments1.Value = 12345

OLEObject itemValueArguments2
itemValueArguments2 = CREATE OLEObject
itemValueArguments2.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")

itemValueArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
itemValueArguments2.ItemDescriptor.ItemID = "Simulation.Register_BOOL"
itemValueArguments2.Value = TRUE

OLEObject itemValueArguments3
itemValueArguments3 = CREATE OLEObject
itemValueArguments3.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")

itemValueArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
itemValueArguments3.ItemDescriptor.ItemID = "Simulation.Register_R4"
itemValueArguments3.Value = 234.56

OLEObject itemValueArgumentsList
itemValueArgumentsList = CREATE OLEObject
itemValueArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
```

```
itemValueArgumentsList.Add(itemValueArguments1)
itemValueArgumentsList.Add(itemValueArguments2)
itemValueArgumentsList.Add(itemValueArguments3)

// Modify value of nodes

OLEObject operationResultList
operationResultList = client.WriteItemValueList(itemValueArgumentsList)

// Display results
Int i
FOR i = 0 TO operationResultList.Count - 1
    OLEObject operationResult
    operationResult = operationResultList.Item[i]
    IF operationResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": success" + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": " +
operationResult.Exception.GetBaseException().Message + "~r~n"
    END IF
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

VBScript

```
Rem This example shows how to write values into 3 items at once.

Option Explicit

Dim ItemValueArguments1: Set ItemValueArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemValueArguments1.Value = 23456

Dim ItemValueArguments2: Set ItemValueArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments2.ItemDescriptor.ItemID = "Simulation.Register_R8"
ItemValueArguments2.Value = 2.34567890

Dim ItemValueArguments3: Set ItemValueArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments3.ItemDescriptor.ItemID = "Simulation.Register_BSTR"
ItemValueArguments3.Value = "ABC"

Dim arguments(2)
Set arguments(0) = ItemValueArguments1
Set arguments(1) = ItemValueArguments2
Set arguments(2) = ItemValueArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.WriteMultipleItemValues(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim OperationResult: Set OperationResult = results(i)
    If OperationResult.Succeeded Then
        WScript.Echo "Result " & i & ": success"
    Else
        WScript.Echo "Result " & i & ": " & OperationResult.Exception.GetBaseException.Message
    End If
Next
```

VBScript

```
Rem This example shows how to write values into 3 items at once, test for success of each write and
display the exception
Rem message in case of failure.

Option Explicit

Dim ItemValueArguments1: Set ItemValueArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemValueArguments1.Value = 23456

Dim ItemValueArguments2: Set ItemValueArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments2.ItemDescriptor.ItemID = "Simulation.Register_R8"
ItemValueArguments2.Value = "This string cannot be converted to VT_R8"

Dim ItemValueArguments3: Set ItemValueArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments3.ItemDescriptor.ItemID = "UnknownItem"
ItemValueArguments3.Value = "ABC"

Dim arguments(2)
Set arguments(0) = ItemValueArguments1
Set arguments(1) = ItemValueArguments2
Set arguments(2) = ItemValueArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.WriteMultipleItemValues(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim OperationResult: Set OperationResult = results(i)
    If OperationResult.Succeeded Then
        WScript.Echo "Result " & i & ": success"
    Else
        WScript.Echo "Result " & i & ": " & OperationResult.Exception.GetBaseException.Message
    End If
Next
```

PHP

```
// This example shows how to write values into 3 items at once, test for success of each write and
display the exception
// message in case of failure.

$ItemValueArguments1 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments1->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemValueArguments1->Value = 23456;

$ItemValueArguments2 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments2->ItemDescriptor->ItemID = "Simulation.Register_R8";
$ItemValueArguments2->Value = "This string cannot be converted to VT_R8";

$ItemValueArguments3 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments3->ItemDescriptor->ItemID = "UnknownItem";
$ItemValueArguments3->Value = "ABC";

$arguments[0] = $ItemValueArguments1;
$arguments[1] = $ItemValueArguments2;
```

```
$arguments[2] = $ItemValueArguments3;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$results = $Client->WriteMultipleItemValues($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $OperationResult = $results[$i];
    if ($OperationResult->Succeeded)
        printf("Result %d: success\n", $i);
    else
        printf("Result %d: %s\n", $i, $OperationResult->ErrorMessageBrief);
}
```

Advanced topics

If you need to write a combination of value, timestamps, and status code be written, and your OPC server support this, see **Writing value, timestamps and status code (OPC UA) (Section 5.1.2.2.2.3)**.

5.1.2.2.1.1 Writing value, timestamp and quality (OPC Classic)

Some newer OPC servers allow a combination of value, timestamp, and quality (VTQ) be written into their items.

If you need to do this, call WriteItem or WriteMultipleItems method.

In This Topic

A single item

Object Pascal

```
// This example shows how to write a value, timestamp and quality into a single item.

class procedure WriteItem.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    Client.WriteItem('', 'OPCLabs.KitServer.2', 'Simulation.Register_I4', 12345,
EncodeDate(1980, 1, 1), DAQualities_GoodNonSpecific);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

VBScript

```
Rem This example shows how to write a value, timestamp and quality into a single item.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Client.WriteItem "", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345,
DateSerial(1980, 1, 1), 192
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

Multiple items

Object Pascal

```
// This example shows how to write values, timestamps and qualities into 3 items at
once.

class procedure WriteMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ItemVtqArguments1: _DAItemVtqArguments;
  ItemVtqArguments2: _DAItemVtqArguments;
  ItemVtqArguments3: _DAItemVtqArguments;
  Results: OleVariant;
  OperationResult: _OperationResult;

begin
  ItemVtqArguments1 := CoDAItemVtqArguments.Create;
  ItemVtqArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemVtqArguments1.ItemDescriptor.ItemID := 'Simulation.Register_I4';
  ItemVtqArguments1.Vtq.Value := 23456;
  ItemVtqArguments1.Vtq.TimestampLocal := Now();
  ItemVtqArguments1.Vtq.Quality.NumericalValue := DAQualities_GoodNonSpecific;

  ItemVtqArguments2 := CoDAItemVtqArguments.Create;
  ItemVtqArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemVtqArguments2.ItemDescriptor.ItemID := 'Simulation.Register_R8';
  ItemVtqArguments2.Vtq.Value := 2.34567890;
  ItemVtqArguments2.Vtq.TimestampLocal := Now();
  ItemVtqArguments2.Vtq.Quality.NumericalValue := DAQualities_GoodNonSpecific;

  ItemVtqArguments3 := CoDAItemVtqArguments.Create;
  ItemVtqArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemVtqArguments3.ItemDescriptor.ItemID := 'Simulation.Register_BSTR';
```

```
  ItemVtqArguments3.Vtq.Value := 'ABC';
  ItemVtqArguments3.Vtq.TimestampLocal := Now();
  ItemVtqArguments3.Vtq.Quality.NumericalValue := DAQualities_GoodNonSpecific;

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := ItemVtqArguments1;
  Arguments[1] := ItemVtqArguments2;
  Arguments[2] := ItemVtqArguments3;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.WriteMultipleItems(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    OperationResult := IInterface(Results[I]) as _OperationResult;
    if OperationResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ' *** Failure: ',
OperationResult.Exception.GetBaseException.Message);
  end;

end;
```

VBScript

```
Rem This example shows how to write values, timestamps and qualities into 3 items at
once.

Option Explicit

Dim ItemVtqArguments1: Set ItemVtqArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemVtqArguments")
ItemVtqArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemVtqArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemVtqArguments1.Vtq.Value = 23456
ItemVtqArguments1.Vtq.TimestampLocal = Now()
ItemVtqArguments1.Vtq.Quality.NumericalValue = 192

Dim ItemVtqArguments2: Set ItemVtqArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemVtqArguments")
ItemVtqArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemVtqArguments2.ItemDescriptor.ItemID = "Simulation.Register_R8"
ItemVtqArguments2.Vtq.Value = 2.34567890
ItemVtqArguments2.Vtq.TimestampLocal = Now()
ItemVtqArguments2.Vtq.Quality.NumericalValue = 192

Dim ItemVtqArguments3: Set ItemVtqArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemVtqArguments")
ItemVtqArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemVtqArguments3.ItemDescriptor.ItemID = "Simulation.Register_BSTR"
```

```
ItemVtqArguments3.Vtq.Value = "ABC"
ItemVtqArguments3.Vtq.TimestampLocal = Now()
ItemVtqArguments3.Vtq.Quality.NumericalValue = 192

Dim arguments(2)
Set arguments(0) = ItemVtqArguments1
Set arguments(1) = ItemVtqArguments2
Set arguments(2) = ItemVtqArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.WriteMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim OperationResult: Set OperationResult = results(i)
    If OperationResult.Succeeded Then
        WScript.Echo "Result " & i & ": success"
    Else
        WScript.Echo "Result " & i & ": " &
OperationResult.Exception.GetBaseException.Message
    End If
Next
```

More...

You can also specify the requested data type when writing the value, timestamp and quality. See **Data Type in OPC Classic Write (Section 5.1.2.2.1.2)**.

5.1.2.2.1.2 Data Type in OPC Classic Write

With some OPC servers or usage cases, it may be needed to specify the requested data type when writing into the item. Reasons for that may be e.g.:

- The OPC server uses some unusual communication method which prevents it from determining the proper canonical data type for the item.
- The OPC server is written in a special way that allows a different behavior based on the concrete data type requested.
- The OPC server may be capable of automatically creating new items when they are requested, and needs the requested data type in order to create the item properly.

In such cases you can pass the requested data type to the OPC server during writing.

In This Topic

A single item

There is an overload of the WriteItemValue extension method that allows you to pass in the requested data type.

C#

```
// This example shows how to write a value into a single item, specifying its requested
data type.

using System;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteItemValue
    {
        public static void RequestedDataType()
        {
            var client = new EasyDAClient();

            try
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345,
                    VarTypes.I4);   // <-- the requested data type
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single item, specifying its requested
data type.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteItemValue
        Public Shared Sub RequestedDataType()
            Dim client = New EasyDAClient()

            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345,
                                      VarTypes.I4) ' <-- the requested data type
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
```

```
        End Sub
    End Class
End Namespace
```

Multiple items

The DAItemValueArguments objects that you pass to the WriteMultipleItemValues method contain an ItemDescriptor, in which you can modify the RequestedDataType property as needed.

C#

```
// Shows how to write into multiple OPC items using a single method call, specifying
their requested data types.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteMultipleItemValues
    {
        public static void RequestedDataType()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Writing multiple item values...");
            OperationResult[] resultArray = client.WriteMultipleItemValues(new[] {
                    new DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I2", 12345)
                        { ItemDescriptor = { RequestedDataType = VarTypes.I2}}, // <--
the requested data type
                    new DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_R4", 234.56)
                        { ItemDescriptor = { RequestedDataType = VarTypes.R4}}  // <--
the requested data type
                });


            for (int i = 0; i < resultArray.Length; i++)
            {
                Debug.Assert(resultArray[i] != null);
                if (resultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                {
                    Debug.Assert(resultArray[i].Exception != null);
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray[i].ErrorMessageBrief);
```

```
            }
        }

        Console.WriteLine("Reading multiple item values...");
        ValueResult[] valueResultArray =
client.ReadMultipleItemValues("OPCLabs.KitServer.2",
            new DAItemDescriptor[] { "Simulation.Register_I2",
"Simulation.Register_R4" });

        for (int i = 0; i < valueResultArray.Length; i++)
        {
            Debug.Assert(valueResultArray[i] != null);
            Console.WriteLine("valueResultArray[{0}]: {1}", i,
valueResultArray[i]);
        }
    }
  }
}
```

VB.NET

```
' Shows how to write into multiple OPC items using a single method call, specifying
their requested data types.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteMultipleItemValues
        Public Shared Sub RequestedDataType()
            Dim client = New EasyDAClient()

            Console.WriteLine("Writing multiple item values...")
            Dim resultArray = client.WriteMultipleItemValues(New DAItemValueArguments()
{ _
                New DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I2", 12345) With _
                {.ItemDescriptor = New DAItemDescriptor() With {.RequestedDataType =
VarTypes.I2}}, _
                New DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_R4", 234.56) With _
                {.ItemDescriptor = New DAItemDescriptor() With {.RequestedDataType =
VarTypes.R4}} _
            })

            For i = 0 To resultArray.Length - 1
                Debug.Assert(resultArray(i) IsNot Nothing)
                If resultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Debug.Assert(resultArray(i).Exception IsNot Nothing)
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray(i).ErrorMessageBrief)
                End If
            Next i
```

```
            Console.WriteLine("Reading multiple item values...")
            Dim valueResultArray = client.ReadMultipleItemValues("OPCLabs.KitServer.2",
                    New DAItemDescriptor() {"Simulation.Register_I2",
"Simulation.Register_R4"})

            For i = 0 To valueResultArray.Length - 1
                Debug.Assert(valueResultArray(i) IsNot Nothing)
                Console.WriteLine("valueResultArray[{0}]: {1}", i, valueResultArray(i))
            Next i
        End Sub
    End Class
End Namespace
```

More...

Specifying the data type is also possible, in a similar manner, when **Writing value, timestamp and quality (OPC Classic) (Section 5.1.2.2.1.1)**.

5.1.2.2.2 Writing Attributes of OPC UA Nodes

A single node and attribute

If you want to write a data value into an attribute of a specific node in OPC UA, call the WriteValue method, passing it the data value you want to write, arguments for endpoint descriptor, node ID, and an optional data type.

In This Topic

C#

```
// This example shows how to write a value into a single node.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
```

```
            try
            {
                client.WriteValue(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10221",
12345);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single node.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10221",
12345)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to write a value into a single node.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "WriteValue.h"

namespace _EasyUAClient
{
    void WriteValue::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
```

```
            ClientPtr->WriteValue(
                L"http://opcua.demo-this.com:51211/UA/SampleServer", // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
                L"nsu=http://test.org/UA/Data/;i=10221",
                12345);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to write a value into a single node.

class procedure WriteValue.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  try
    Client.WriteValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer', // or 'opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer'
      'nsu=http://test.org/UA/Data/;i=10221',
      12345);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PHP

```
// This example shows how to write a value into a single node.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $Client->WriteValue(
        "http://opcua.demo-this.com:51211/UA/SampleServer", // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
        "nsu=http://test.org/UA/Data/;i=10221",
        12345);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}
```

PowerScript

```
// This example shows how to write a value into a single node.
```

```
mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Modify value of a node
TRY
    client.WriteValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", 12345)
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + oleRuntimeError.Description +
"~r~n"
    RETURN
END TRY

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to write a value into a single node.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Perform the operation
client.WriteValue(
    'http://opcua.demo-this.com:51211/UA/SampleServer', # or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
    'nsu=http://test.org/UA/Data/;i=10221',
    12345)
```

Visual Basic (VB 6.)

```
Rem This example shows how to write a value into a single node.

Private Sub WriteValue_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Call Client.WriteValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", 12345)
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description & vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

End Sub
```

VBScript

```
Rem This example shows how to write a value into a single node.
```

```
Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Client.WriteValue "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", 12345 ' or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

When you specify individual arguments to the simplest overload of the WriteValue method, the method will write to the **Value** attribute of the node, and if the value type is an array, it will modify the whole contents of that array. If you want to write into a different attribute, or write only to a subset of an array, use one of the more complex overloads of the WriteValue method, such as with an argument of UAWriteValueArguments type, and fill this argument will all necessary information.

Multiple nodes or attributes

For writing data values into multiple OPC attributes (in the same or different nodes) in an efficient manner, call the WriteMultipleValues method. You pass in an array of UAWriteValueArgument objects, each specifying the location of OPC node, attribute, and the value to be written.

C#

```
// This example shows how to write values into 3 nodes at once.

using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226", 2.34567890),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227", "ABC")
                });
```

```
            // Production code would check the success of the operation. See separate example for
that.
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226", 2.3456789), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227", "ABC") _
                } _
             )
            ' Production code would check the success of the operation. See separate example for
that.
        End Sub
    End Class
End Namespace
```

The example above can be extended by properly testing for success of each operation, like this:

C#

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void TestSuccess()
        {
```

```
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226",
                        "This string cannot be converted to Double"),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, test for success of each write and
display the exception
' message in case of failure.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub TestSuccess()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult = client.WriteMultipleValues(New
UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226", "This string cannot be converted to Double"), _
                    New UAWriteValueArguments(endpointDescriptor,
```

```
"nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                } _
            )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "WriteMultipleValues.h"

namespace _EasyUAClient
{
    void WriteMultipleValues::TestSuccess()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            _UAWriteValueArgumentsPtr WriteValueArguments1Ptr(_uuidof(UAWriteValueArguments));
            WriteValueArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            WriteValueArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10221";
            WriteValueArguments1Ptr->Value = 23456;

            _UAWriteValueArgumentsPtr WriteValueArguments2Ptr(_uuidof(UAWriteValueArguments));
            WriteValueArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            WriteValueArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10226";
            WriteValueArguments2Ptr->Value = L"This string cannot be converted to Double";

            _UAWriteValueArgumentsPtr WriteValueArguments3Ptr(_uuidof(UAWriteValueArguments));
            WriteValueArguments3Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            WriteValueArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;s=UnknownNode";
            WriteValueArguments3Ptr->Value = L"ABC";

            CComSafeArray<VARIANT> arguments(3);
            arguments.SetAt(0, _variant_t((IDispatch*)WriteValueArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)WriteValueArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)WriteValueArguments3Ptr));
            CComVariant vArguments(arguments);
```

```
            // Obtain values. By default, the Value attributes of the nodes will be Write.
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->WriteMultipleValues(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _UAWriteResultPtr ResultPtr = results[i];

                if (ResultPtr->Succeeded)
                    _tprintf(_T("Result %d success\n"), i);
                else
                    _tprintf(_T("Result %d: %s\n"), i, (LPCTSTR)CW2CT(ResultPtr->Exception-
>GetBaseException()->Message));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to write values into 3 nodes at once, test for
// success of each write and display the exception message in case of failure.

class procedure WriteMultipleValues.TestSuccess;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  WriteResult: _UAWriteResult;
  WriteValueArguments1, WriteValueArguments2, WriteValueArguments3: _UAWriteValueArguments;
  Results: OleVariant;
begin
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 23456;

  WriteValueArguments2 := CoUAWriteValueArguments.Create;
  WriteValueArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10226';
  WriteValueArguments2.Value := 'This string cannot be converted to Double';

  WriteValueArguments3 := CoUAWriteValueArguments.Create;
  WriteValueArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;s=UnknownNode';
  WriteValueArguments3.Value := 'ABC';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := WriteValueArguments1;
  Arguments[1] := WriteValueArguments2;
  Arguments[2] := WriteValueArguments3;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;
```

```
  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteResult := IInterface(Results[I]) as _UAWriteResult;
    if WriteResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ': ', WriteResult.Exception.GetBaseException.Message);
  end;
end;
```

PHP

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$WriteValueArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$WriteValueArguments1->Value = 23456;

$WriteValueArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226";
$WriteValueArguments2->Value = "This string cannot be converted to Double";

$WriteValueArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode";
$WriteValueArguments3->Value = "ABC";

$arguments[0] = $WriteValueArguments1;
$arguments[1] = $WriteValueArguments2;
$arguments[2] = $WriteValueArguments3;

// Modify values of nodes
$results = $Client->WriteMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $WriteResult = $results[$i];
    // The UA Test Server does not support this, and therefore a failure will occur.
    if ($WriteResult->Succeeded)
        printf("Result %d success\n", $i);
    else
        printf("Result %d: %s \n", $i, $WriteResult->Exception->GetBaseException()->Message);
}
```

PowerScript

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments.

OLEObject writeValueArguments1
writeValueArguments1 = CREATE OLEObject
writeValueArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAwriteValueArguments")

writeValueArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
writeValueArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10221"
writeValueArguments1.Value = 23456

OLEObject writeValueArguments2
writeValueArguments2 = CREATE OLEObject
writeValueArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAwriteValueArguments")

writeValueArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
writeValueArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10226"
writeValueArguments2.Value = "This string cannot be converted to Double"

OLEObject writeValueArguments3
writeValueArguments3 = CREATE OLEObject
writeValueArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAwriteValueArguments")

writeValueArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
writeValueArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode"
writeValueArguments3.Value = "ABC"

OLEObject writeValueArgumentsList
writeValueArgumentsList = CREATE OLEObject
writeValueArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
writeValueArgumentsList.Add(writeValueArguments1)
writeValueArgumentsList.Add(writeValueArguments2)
writeValueArgumentsList.Add(writeValueArguments3)

// Modify value of nodes

OLEObject operationResultList
operationResultList = client.WriteValueList(writeValueArgumentsList)

// Display results
Int i
FOR i = 0 TO operationResultList.Count - 1
    OLEObject operationResult
    operationResult = operationResultList.Item[i]
    IF operationResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": success" + "~r~n"
```

```
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": " +
operationResult.Exception.GetBaseException().Message + "~r~n"
    END IF
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Visual Basic (VB 6.)

```
Rem This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
Rem message in case of failure.

Private Sub WriteMultipleValues_TestSuccess_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    Dim WriteValueArguments1 As New UAWriteValueArguments
    WriteValueArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    WriteValueArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10221"
    WriteValueArguments1.SetValue 23456

    Dim WriteValueArguments2 As New UAWriteValueArguments
    WriteValueArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    WriteValueArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10226"
    WriteValueArguments2.SetValue "This string cannot be converted to Double"

    Dim WriteValueArguments3 As New UAWriteValueArguments
    WriteValueArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    WriteValueArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode"
    WriteValueArguments3.SetValue "ABC"

    Dim arguments(2) As Variant
    Set arguments(0) = WriteValueArguments1
    Set arguments(1) = WriteValueArguments2
    Set arguments(2) = WriteValueArguments3

    ' Modify values of nodes
    Dim results As Variant
    results = Client.WriteMultipleValues(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        Dim Result As UAWriteResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "Result " & i & " success" & vbCrLf
        Else
            OutputText = OutputText & "Result " & i & ": " &
Result.Exception.GetBaseException().Message & vbCrLf
        End If
    Next
End Sub
```

VBScript

```
Rem This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
Rem message in case of failure.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim WriteValueArguments1: Set WriteValueArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10221"
WriteValueArguments1.Value = 23456

Dim WriteValueArguments2: Set WriteValueArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10226"
WriteValueArguments2.Value = "This string cannot be converted to Double"

Dim WriteValueArguments3: Set WriteValueArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode"
WriteValueArguments3.Value = "ABC"

Dim arguments(2)
Set arguments(0) = WriteValueArguments1
Set arguments(1) = WriteValueArguments2
Set arguments(2) = WriteValueArguments3

' Modify values of nodes
Dim results: results = Client.WriteMultipleValues(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim WriteResult: Set WriteResult = results(i)
    If WriteResult.Succeeded Then
        WScript.Echo "Result " & i & " success"
    Else
        WScript.Echo "Result " & i & ": " & WriteResult.Exception.GetBaseException().Message
    End If
Next
```

Write results

The WriteMultiple and WriteMultipleValues methods return an array of UAWriteResult objects. Besides the inherited members of OperationResult, such as the Exception, the UAWriteResult contains additional information that describes the outcome of the Write operation in case of success:

- The Clamped flag: Determines whether the value written was accepted but was clamped.

- The CompletesAsynchronously flag: Determines whether the processing will complete asynchronously.

5.1.2.2.2.1 Data Type in OPC UA Write

The value written needs to be of correct type, and the data type needs to be passed to the OPC-UA server together with the value.

OPC Data Client-UA processes the value and data type in following steps:

1. Determine the (CLS-compliant) .NET data type. If you are writing to an attribute other than the **Value** attribute, the component uses the data type of that attribute as given by the OPC-UA specification, because of these attributes, the data types are fixed.

   Wherever the **Value** attribute is being written (which is the most common case), and your code has not provided a specific value type, the data type is resolved from the OPC-UA server by reading the **DataType** and **ValueRank** attributes of the same node. This is because the type of **Value** attribute can differ with each node.

   When the data type of the node is not among the standard data types defined in OPC UA, attempts to find its supertype, by following the appropriate reference in the server's address space. This process is repeated if needed, until a known data type is found, or the supertype cannot be determined.

2. The value that comes from your code is converted to the (CLS-compliant) .NET data type determined in previous step.
3. Determine the OPC-UA built-in type for the attribute being written.
4. If needed, the type of the value is changed to the OPC-UA built-in type determined in the previous step.

As you can see, not specifying the value type is convenient, but causes additional Read and possibly Browse operations on the OPC-UA server in Step 1. This can have negative performance implications. You can prevent the additional Reads with **Value** attribute by specifying the value type explicitly. When you do so, you must be careful to specify the correct type. For example, if you pass in a value of 199 and specify the value type as a signed integer (Int32 in .NET), OPC Data Client will send the value and corresponding type ID to the OPC-UA server as a signed integer. If the server expects a value that must be an unsigned integer, even though the value is otherwise correct (and also non-negative), an error can occur.

How do you specify the value type in your code? This depends on the type of method you are calling. Some methods accept individual arguments, and they have overloads with either valueType or valueTypeCode argument. Other methods accept UAWriteArguments or UAWriteValueArguments object; in this case, you specify the value type by properly setting the ValueType or ValueTypeCode property in this object. There is also a ValueTypeFullName property, which is a string form of the ValueType property. Its use is mainly from COM languages, where the .NET Type is not easily accessible.

The TypeCode can be used for basic, simple types. For more complex types, such as arrays, you need to specify the .NET Type.

When the value type is a null reference, or the value type code is TypeCode.Empty, it means that the value type is not specified, and the component will determine it by interrogating the OPC-UA server, as described at the beginning of the article.

Examples

The example below writes a single value, specifying the type to be written using a .NET Type.

Note that this example uses an overload of the WriteValue method that is not available in COM. If you are calling from a COM language or tool, use the WriteMultipleValues method instead. An example for that is given further down.

C#

```
// This example shows how to write a value into a single node, specifying a type
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the server reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// For a selected subset of most common types, you can also use a different overload of
the WriteValue method, and specify
// a value from the TypeCode enumeration instead.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void Type()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10221",
                    12345,
                    typeof(Int32));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single node, specifying a type
explicitly.
```

```
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the server reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' For a selected subset of most common types, you can also use a different overload of
the WriteValue method, and specify
' a value from the TypeCode enumeration instead.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
        Public Shared Sub Type()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue( _
                    endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;i=10221", _
                    12345, _
                    GetType(Int32))
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

The example below writes a single value, specifying the type to be written using a TypeCode.

Note that this example uses an overload of the WriteValue method that is not available in COM. If you are calling from a COM language or tool, use the WriteMultipleValues method instead. An example for that is given further down.

C#

```
// This example shows how to write a value into a single node, specifying a type code
explicitly.
//
// Reasons for specifying the type explicitly might be:
```

```
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
// a different overload of the WriteValue method.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void TypeCode()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10221",
                    12345,
                    System.TypeCode.Int32);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

Object Pascal

```
// This example shows how to write a value into a single node, specifying a type code
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
```

```
// TypeCode is easy to use, but it does not cover all possible types. It is also
possible
// to specify the .NET Type, using a different overload of the WriteValue method.


class procedure WriteValue.TypeCode;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  EndpointDescriptor: string;
  WriteResult: _UAWriteResult;
  WriteValueArguments1: _UAWriteValueArguments;
  Results: OleVariant;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Prepare the arguments
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 12345;
  WriteValueArguments1.ValueTypeCode := TypeCode_Int32;    // here is the type
explicitly specified

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := WriteValueArguments1;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify value of node
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  WriteResult := IUnknown(Results[0]) as _UAWriteResult;
  if not WriteResult.Succeeded then
    WriteLn('*** Failure: ', WriteResult.Exception.GetBaseException.Message);

end;
```

PHP

```
// This example shows how to write a value into a single node, specifying a type code
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// TypeCode is easy to use, but it does not cover all possible types. It is also
possible
// to specify the .NET Type, using a different overload of the WriteValue method.
```

```
const TypeCode_Int32 = 9;

$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";

// Prepare the arguments
$WriteValueArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$WriteValueArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$WriteValueArguments1->Value = 12345;
$WriteValueArguments1->ValueTypeCode = TypeCode_Int32;    // here is the type
explicitly specified

$arguments[0] = $WriteValueArguments1;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify value of node
$results = $Client->WriteMultipleValues($arguments);

$WriteResult = $results[0];
if (!$WriteResult->Succeeded)
    printf("*** Failure: %s\n", $WriteResult->Exception->GetBaseException()->Message);
```

VB.NET

```
' This example shows how to write a value into a single node, specifying a type code
explicitly.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
' a different overload of the WriteValue method.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
        Public Shared Sub TypeCode()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue( _
                    endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;i=10221", _
                    12345, _
                    System.TypeCode.Int32)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to write a value into a single node, specifying a type code
explicitly.
Rem
Rem Reasons for specifying the type explicitly might be:
Rem - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
Rem - The data type that the reports is incorrect.
Rem - Writing with an explicitly specified type is more efficient.
Rem
Rem TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
Rem a different overload of the WriteValue method.

Option Explicit

Const TypeCode_Int32 = 9

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Prepare the arguments
Dim WriteValueArguments1: Set WriteValueArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments1.EndpointDescriptor.UrlString = endpointDescriptor
WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221"
WriteValueArguments1.Value = 12345
WriteValueArguments1.ValueTypeCode = TypeCode_Int32

Dim arguments(0)
```

```
Set arguments(0) = WriteValueArguments1

' Modify value of a node
Dim results: results = Client.WriteMultipleValues(arguments)
Dim WriteResult: Set WriteResult = results(0)
If Not WriteResult.Succeeded Then
    WScript.Echo "*** Failure: " & WriteResult.Exception.GetBaseException().Message
End If
```

Visual Basic (VB 6.)

```
Rem This example shows how to write a value into a single node, specifying a type code
explicitly.
Rem
Rem Reasons for specifying the type explicitly might be:
Rem - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
Rem - The data type that the reports is incorrect.
Rem - Writing with an explicitly specified type is more efficient.
Rem
Rem TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
Rem a different overload of the WriteValue method.

Private Sub WriteValue_TypeCode_Command_Click()
    OutputText = ""

    Const TypeCode_Int32 = 9

    Dim endpointDescriptor As String
    endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
    'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
    'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Prepare the arguments
    Dim WriteValueArguments1 As New UAWriteValueArguments
    WriteValueArguments1.endpointDescriptor.UrlString = endpointDescriptor
    WriteValueArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10221"
    WriteValueArguments1.SetValue 12345
    WriteValueArguments1.ValueTypeCode = TypeCode_Int32

    Dim arguments(0) As Variant
    Set arguments(0) = WriteValueArguments1

    ' Modify value of node
    Dim results As Variant
    results = Client.WriteMultipleValues(arguments)

    ' Display results
    Dim Result As UAWriteResult: Set Result = results(0)
    If Not Result.Succeeded Then
        OutputText = OutputText & "*** Failure: " &
Result.Exception.GetBaseException().Message & vbCrLf
```

```
    End If
End Sub
```

The example below writes multiple values, specifying the types to be written in the ValueType property, using a .NET Type.

Note that this approach is, at least theoretically, also available in COM languages and tools, but due to difficulties with obtaining the .NET Type, you are much more likely to specify the type using the ValueTypeCode or ValueTypeFullName property, for which the examples are given further down.

C#

```
// This example shows how to write values into 3 nodes at once, specifying a type
explicitly. It tests for success of each
// write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueTypeCode or
ValueTypeFullName properties.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void ValueType()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10221", 23456)
                        {ValueType = typeof(Int32)},    // here is the type explicitly
specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double")
                        {ValueType = typeof(Double)},    // here is the type explicitly
specified
```

```
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                        {ValueType = typeof(string)}    // here is the type explicitly
specified
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, specifying a type
explicitly. It tests for success of each
' write and displays the exception message in case of failure.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' Alternative ways of specifying the type are using the ValueTypeCode or
ValueTypeFullName properties.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub ValueType()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult =
client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10221", 23456) _
                        With {.ValueType = GetType(Int32)}, _
```

```
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double") _
                        With {.ValueType = GetType(Double)}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                        With {.ValueType = GetType(String)} _
                } _
            )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

The example below writes multiple values, specifying the types to be written in the ValueTypeCode property, using a TypeCode.

This approach is suitable for a basic set of types covered by the TypeCode enumeration, and is also accessible to COM languages and tools. If you need to specify a type that is not covered by the TypeCode enumeration, use the ValueType or ValueTypeFullName property instead.

C#

```
// This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
// each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void ValueTypeCode()
        {
            UAEndpointDescriptor endpointDescriptor =
```

```
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10221", 23456)
                        {ValueTypeCode = TypeCode.Int32},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double")
                        {ValueTypeCode = TypeCode.Double},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                        {ValueTypeCode = TypeCode.String}    // here is the type
explicitly specified
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
' each write and displays the exception message in case of failure.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
```

```
    Partial Friend Class WriteMultipleValues
        Public Shared Sub ValueTypeCode()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult =
client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10221", 23456) _
                        With {.ValueTypeCode = TypeCode.Int32}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double") _
                        With {.ValueTypeCode = TypeCode.Double}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                        With {.ValueTypeCode = TypeCode.String} _
                } _
             )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
// each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.
```

```
class procedure WriteMultipleValues.ValueTypeCode;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  WriteResult: _UAWriteResult;
  WriteValueArguments1, WriteValueArguments2, WriteValueArguments3:
_UAWriteValueArguments;
  Results: OleVariant;
begin
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 23456;
  WriteValueArguments1.ValueTypeCode := TypeCode_Int32;    // here is the type
explicitly specified

  WriteValueArguments2 := CoUAWriteValueArguments.Create;
  WriteValueArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10226';
  WriteValueArguments2.Value := 'This string cannot be converted to Double';
  WriteValueArguments2.ValueTypeCode := TypeCode_Double;    // here is the type
explicitly specified

  WriteValueArguments3 := CoUAWriteValueArguments.Create;
  WriteValueArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;s=UnknownNode';
  WriteValueArguments3.Value := 'ABC';
  WriteValueArguments3.ValueTypeCode := TypeCode_String;    // here is the type
explicitly specified

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := WriteValueArguments1;
  Arguments[1] := WriteValueArguments2;
  Arguments[2] := WriteValueArguments3;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteResult := IInterface(Results[I]) as _UAWriteResult;
    if WriteResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ': ', WriteResult.Exception.GetBaseException.Message);
```

```
  end;
end;
```

PHP

```
// This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
// each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

const TypeCode_Int32 = 9;
const TypeCode_Double = 14;
const TypeCode_String = 18;

$WriteValueArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$WriteValueArguments1->Value = 23456;
$WriteValueArguments1->ValueTypeCode = TypeCode_Int32;    // here is the type
explicitly specified

$WriteValueArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226";
$WriteValueArguments2->Value = "This string cannot be converted to Double";
$WriteValueArguments2->ValueTypeCode = TypeCode_Double;    // here is the type
explicitly specified

$WriteValueArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode";
$WriteValueArguments3->Value = "ABC";
$WriteValueArguments3->ValueTypeCode = TypeCode_String;    // here is the type
explicitly specified

$arguments[0] = $WriteValueArguments1;
$arguments[1] = $WriteValueArguments2;
$arguments[2] = $WriteValueArguments3;

// Instantiate the client object
```

```
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify values of nodes
$results = $Client->WriteMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $WriteResult = $results[$i];
    if ($WriteResult->Succeeded)
        printf("Result %d success\n", $i);
    else
        printf("Result %d: %s \n", $i, $WriteResult->Exception->GetBaseException()-
>Message);
}
```

The example below writes multiple values, specifying the types to be written in the ValueTypeFullName property, using a string form of the .NET Type.

This approach is especially suitable in COM languages and tools, if you need to specify a type that is not covered by the TypeCode enumeration.

C#

```
// This example shows how to write values into 3 nodes at once, specifying a type's
full name explicitly. It tests for
// success of each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeCode
properties.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void ValueTypeFullName()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();
```

```
            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10221", 23456)
                        {ValueTypeFullName = "System.Int32"},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double")
                        {ValueTypeFullName = "System.Double"},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                        {ValueTypeFullName = "System.String"}    // here is the type
explicitly specified
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, specifying a type's full
name explicitly. It tests for
' success of each write and displays the exception message in case of failure.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' Alternative ways of specifying the type are using the ValueType or ValueTypeCode
properties.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub ValueTypeFullName()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
```

```
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult =
client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10221", 23456) _
                        With {.ValueTypeFullName = "System.Int32"}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double") _
                        With {.ValueTypeFullName = "System.Double"}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                        With {.ValueTypeFullName = "System.String"} _
                } _
             )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to write values into 3 nodes at once, specifying a type's
full name explicitly. It tests for
// success of each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeCode
properties.

class procedure WriteMultipleValues.ValueTypeFullName;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  WriteResult: _UAWriteResult;
  WriteValueArguments1, WriteValueArguments2, WriteValueArguments3:
```

```
_UAWriteValueArguments;
  Results: OleVariant;
begin
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 23456;
  WriteValueArguments1.ValueTypeFullName := 'System.Int32';    // here is the type
explicitly specified

  WriteValueArguments2 := CoUAWriteValueArguments.Create;
  WriteValueArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10226';
  WriteValueArguments2.Value := 'This string cannot be converted to Double';
  WriteValueArguments2.ValueTypeFullName := 'System.Double';    // here is the type
explicitly specified

  WriteValueArguments3 := CoUAWriteValueArguments.Create;
  WriteValueArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;s=UnknownNode';
  WriteValueArguments3.Value := 'ABC';
  WriteValueArguments3.ValueTypeFullName := 'System.String';    // here is the type
explicitly specified

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := WriteValueArguments1;
  Arguments[1] := WriteValueArguments2;
  Arguments[2] := WriteValueArguments3;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteResult := IInterface(Results[I]) as _UAWriteResult;
    if WriteResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ': ', WriteResult.Exception.GetBaseException.Message);
  end;
end;
```

5.1.2.2.2.2 Writing less common OPC UA data types

Writing a ByteString

C#

```
// This example shows how to write a value into a single node that is of type
ByteString.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void ByteString()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10230",
                    new byte[] {11, 22, 33, 44, 55});
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single node that is of type
ByteString.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
```

```
    Partial Friend Class WriteValue
        Public Shared Sub ByteString()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10230", New Byte() {11, 22, 33, 44, 55})
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to write a value into a single node that is of type
ByteString.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include <atlsafe.h>
#include "WriteValue.h"

namespace _EasyUAClient
{
    void WriteValue::ByteString()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Prepare the value to be written
            CComSafeArray<BYTE> array(5);
            array.SetAt(0, 11);
            array.SetAt(1, 22);
            array.SetAt(2, 33);
            array.SetAt(3, 44);
            array.SetAt(4, 55);

            const CComVariant value(array);

            // Perform the operation
```

```
            ClientPtr->WriteValue(
                L"http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
                L"nsu=http://test.org/UA/Data/;i=10230",
                value);
        }
        // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to write a value into a single node that is of type
ByteString.

class procedure WriteValue.ByteString;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Values: OleVariant;
begin
  Values := VarArrayCreate([0, 4], varByte);
  Values[0] := 11;
  Values[1] := 22;
  Values[2] := 33;
  Values[3] := 44;
  Values[4] := 55;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify value of a node
  try
    Client.WriteValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer', // or 'opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer'
      'nsu=http://test.org/UA/Data/;i=10230', Values);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PHP

```
// This example shows how to write a value into a single node that is of type
ByteString.

$Values[0] = 11;
$Values[1] = 22;
$Values[2] = 33;
$Values[3] = 44;
$Values[4] = 55;
```

```
// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify value of a node
try
{
    $Client->WriteValue(
        "http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        "nsu=http://test.org/UA/Data/;i=10230",
        $Values);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}
```

Visual Basic (VB 6.)

```
Rem This example shows how to write a value into a single node that is of type
ByteString.

Private Sub WriteValue_ByteString_Command_Click()
    OutputText = ""

    Dim Values(4) As Byte
    Values(0) = 11
    Values(1) = 22
    Values(2) = 33
    Values(3) = 44
    Values(4) = 55

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Call Client.WriteValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10230", Values)
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

End Sub
```

5.1.2.2.2.3 Writing value, timestamps and status code (OPC UA)

Some OPC servers allow a combination of value, timestamps, and status code be written into their nodes (or to some nodes only). If you need to do this, call Write or WriteMultiple method.

A single node and attribute

C#

```
// This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class Write
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            try
            {
                // Modify data of a node's attribute
                client.Write(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                    new UAAttributeData(12345, UASeverity.GoodOrSuccess,
DateTime.UtcNow));

                // The target server may not support this, and in such case a failure
will occur.
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.
```

```
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class Write
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Try
                ' Modify data of a node's attribute
                client.Write(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                                        New UAAttributeData(12345, UASeverity.GoodOrSuccess,
Date.UtcNow)) ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

                ' The target server may not support this, and in such case a failure
will occur.
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message)
            End Try
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.

$GoodOrSuccess = 0;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify data of a node's attribute
$StatusCode = new COM("OpcLabs.EasyOpc.UA.UAStatusCode");
$StatusCode->Severity = $GoodOrSuccess;
$AttributeData = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData->Value = 12345;
$AttributeData->StatusCode = $StatusCode;
$AttributeData->SourceTimestamp = (time() - 25569)/86400.0;

// Perform the operation
try
{
    $Client->Write(
```

```
        "http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        "nsu=http://test.org/UA/Data/;i=10221",
        $AttributeData);
    // The target server may not support this, and in such case a failure will occur.
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}
```

VBScript

```
Rem This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.

Option Explicit

Const GoodOrSuccess = 0

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Modify data of a node's attribute
Dim StatusCode: Set StatusCode = CreateObject("OpcLabs.EasyOpc.UA.UAStatusCode")
StatusCode.Severity = GoodOrSuccess
Dim AttributeData: Set AttributeData =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData.Value = 12345
AttributeData.StatusCode = StatusCode
AttributeData.SourceTimestamp = Now

' Perform the operation
On Error Resume Next
Client.Write "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", _
            AttributeData ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
' The target server may not support this, and in such case a failure will occur.
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

Multiple nodes or attributes

It is more efficient to use the WriteMultiple method, instead of several calls to the Write method.

C#

```
// This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
```

```
// write and display the exception message in case of failure.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class WriteMultiple
    {
        public static void TestSuccess()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify data of nodes' attributes
            OperationResult[] operationResultArray = client.WriteMultiple(new[]
                {
                    new UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                        new UAAttributeData(23456, UASeverity.GoodOrSuccess,
DateTime.UtcNow)),
                    new UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226",
                        new UAAttributeData(2.34567890, UASeverity.GoodOrSuccess,
DateTime.UtcNow)),
                    new UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227",
                        new UAAttributeData("ABC", UASeverity.GoodOrSuccess,
DateTime.UtcNow))
                });

            // The target server may not support this, and in such case failures will
occur.

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
' write and display the exception message in case of failure.
```

```
Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class WriteMultiple
        Public Shared Sub TestSuccess()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify data of nodes' attributes
            Dim operationResultArray() As OperationResult = client.WriteMultiple(New
UAWriteArguments() _
                {
                    New UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                                         New UAAttributeData(23456,
UASeverity.GoodOrSuccess, Date.UtcNow)),
                    New UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226",
                                         New UAAttributeData(2.3456789,
UASeverity.GoodOrSuccess, Date.UtcNow)),
                    New UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227",
                                         New UAAttributeData("ABC",
UASeverity.GoodOrSuccess, Date.UtcNow))
                }
            )

            ' The target server may not support this, and in such case failures will
occur.

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to write data (a value, timestamps and status code) into 3
```

```
nodes at once, test for success of each
// write and display the exception message in case of failure.

$GoodOrSuccess = 0;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$StatusCode = new COM("OpcLabs.EasyOpc.UA.UAStatusCode");
$StatusCode->Severity = $GoodOrSuccess;

$WriteArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments");
$WriteArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$AttributeData1 = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData1->Value = 23456;
$AttributeData1->StatusCode = $StatusCode;
$AttributeData1->SourceTimestamp = (time() - 25569)/86400.0;    // works in PHP v5.6,
does not work in PHP 7.3.10
$WriteArguments1->AttributeData = $AttributeData1;

$WriteArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments");
$WriteArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226";
$AttributeData2 = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData2->Value = 2.3456789;
$AttributeData2->StatusCode = $StatusCode;
$AttributeData2->SourceTimestamp = (time() - 25569)/86400.0;    // works in PHP v5.6,
does not work in PHP 7.3.10
$WriteArguments2->AttributeData = $AttributeData2;

$WriteArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments");
$WriteArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10227";
$AttributeData3 = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData3->Value = "ABC";
$AttributeData3->StatusCode = $StatusCode;
$AttributeData3->SourceTimestamp = (time() - 25569)/86400.0;    // works in PHP v5.6,
does not work in PHP 7.3.10
$WriteArguments3->AttributeData = $AttributeData3;

$arguments[0] = $WriteArguments1;
$arguments[1] = $WriteArguments2;
$arguments[2] = $WriteArguments3;

// Modify data of nodes' attributes
$results = $Client->WriteMultiple($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
```

```
    $WriteResult = $results[$i];
    // The target server may not support this, and in such case failures will occur.
    if ($WriteResult->Succeeded)
        printf("Result %d success\n", $i);
    else
        printf("Result %d: %s \n", $i, $WriteResult->Exception->GetBaseException()-
>Message);
}
```

VBScript

```
Rem This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
Rem write and display the exception message in case of failure.

Option Explicit

Const GoodOrSuccess = 0

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim StatusCode: Set StatusCode = CreateObject("OpcLabs.EasyOpc.UA.UAStatusCode")
StatusCode.Severity = GoodOrSuccess

Dim WriteArguments1: Set WriteArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments")
WriteArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221"
Dim AttributeData1: Set AttributeData1 =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData1.Value = 23456
AttributeData1.StatusCode = StatusCode
AttributeData1.SourceTimestamp = Now
WriteArguments1.AttributeData = AttributeData1

Dim WriteArguments2: Set WriteArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments")
WriteArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226"
Dim AttributeData2: Set AttributeData2 =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData2.Value = 2.3456789
AttributeData2.StatusCode = StatusCode
AttributeData2.SourceTimestamp = Now
WriteArguments2.AttributeData = AttributeData2

Dim WriteArguments3: Set WriteArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments")
WriteArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10227"
```

```
Dim AttributeData3: Set AttributeData3 =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData3.Value = "ABC"
AttributeData3.StatusCode = StatusCode
AttributeData3.SourceTimestamp = Now
WriteArguments3.AttributeData = AttributeData3

Dim arguments(2)
Set arguments(0) = WriteArguments1
Set arguments(1) = WriteArguments2
Set arguments(2) = WriteArguments3

' Modify data of nodes' attributes
Dim results: results = Client.WriteMultiple(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim WriteResult: Set WriteResult = results(i)
    ' The target server may not support this, and in such case failures will occur.
    If WriteResult.Succeeded Then
        WScript.Echo "Result " & i & " success"
    Else
        WScript.Echo "Result " & i & ": " &
WriteResult.Exception.GetBaseException().Message
    End If
Next
```

5.1.2.3 Browsing for Information (OPC Data)

OPC Data Client contains methods that allow your application to retrieve and enumerate information about OPC servers that exist on the network, and data available within these servers. Your code can then make use of the information obtained, e.g. to accommodate to configuration changes dynamically.

Note that if you just want to allow your user to browse interactively for various OPC elements, you can simply code your application to invoke the common dialogs that are already implemented in OPC Data Client (they are described further down in this text).

The methods we are describing in this chapter are for programmatic browsing, with no user interface (or when you provide the user interface by your own code).

5.1.2.3.1 Browsing for OPC Classic Servers

If you want to retrieve a list of OPC Data Access servers registered on a local or remote computer, call the BrowseServers method, passing it the name or address of the remote machine (use empty string for local computer).

In OPC Data Client.NET, you will receive back a ServerElementCollection object. If you want to connect to this OPC server later in your code by calling other methods, use the built-in conversion of ServerElement to a String or ServerDescriptor, and pass the resulting string as a serverClass or a serverUrl argument either directly to the method call, or to a constructor of ServerDescriptor object.

In OPC Data Client-COM, if you want to connect to some OPC server later in your code by calling other methods, obtain the value of ServerElement.ServerClass property, and pass the resulting string as a serverClass argument to the method call that accepts it.

Each ServerElement contains information gathered about one OPC server found on the specified machine, including things like the server's CLSID, ProgID, vendor name, and readable description. For an OPC XML server, it contains its URL.

JScript

```
// This example shows how to obtain all ProgIDs of all OPC Data Access servers on the
local machine.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
var ServerElements = Client.BrowseServers("")

for (var objEnum = new Enumerator(ServerElements) ; !objEnum.atEnd() ;
objEnum.moveNext()) {
    var ServerElement = objEnum.item();
    WScript.Echo("ServerElements(\"" + ServerElement.UrlString + "\").ProgId: " +
ServerElement.ProgId);
}
```

PHP

```
// This example shows how to obtain all ProgIDs of all OPC Data Access servers on the
local machine.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $ServerElements = $Client->BrowseServers("");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

foreach ($ServerElements as $ServerElement)
{
    printf("ServerElements(\"%s\").ProgId: %s\n", $ServerElement->ClsidString,
$ServerElement->ProgId);
}
```

VBScript

```
Rem This example shows how to obtain all ProgIDs of all OPC Data Access servers on the
local machine.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim ServerElements: Set ServerElements = Client.BrowseServers("")
If Err.Number <> 0 Then
```

```
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "ServerElements(""" & ServerElement.ClsidString & """).ProgId: " &
ServerElement.ProgId
Next
```

VBScript

```
Rem This example shows all information available about categories that particular OPC
servers do support.

Option Explicit

Sub DumpServerElements(ByVal ServerElements)
Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "Categories of """ & ServerElement.ProgID & """:"
    With ServerElement.ServerCategories
        WScript.Echo Space(4) & ".OpcAlarmsAndEvents10: " & .OpcAlarmsAndEvents10
        WScript.Echo Space(4) & ".OpcDataAccess10: " & .OpcDataAccess10
        WScript.Echo Space(4) & ".OpcDataAccess20: " & .OpcDataAccess20
        WScript.Echo Space(4) & ".OpcDataAccess30: " & .OpcDataAccess30
        WScript.Echo Space(4) & ".ToString(): " & .ToString()
    End With
Next
End Sub



Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.Echo
WScript.Echo "OPC DATA ACCESS"
On Error Resume Next
Dim DAServerElements: Set DAServerElements = DAClient.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
DumpServerElements DAServerElements

Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.Echo
WScript.Echo "OPC ALARMS AND EVENTS"
On Error Resume Next
Dim AEServerElements: Set AEServerElements = AEClient.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
DumpServerElements AEServerElements
```

VBScript

```
Rem This example shows all information available about OPC servers.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim ServerElements: Set ServerElements = Client.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "Information about server """ & ServerElement & """:"
    With ServerElement
        WScript.Echo Space(4) & ".ServerClass: " & .ServerClass
        WScript.Echo Space(4) & ".ClsidString: " & .ClsidString
        WScript.Echo Space(4) & ".ProgId: " & .ProgId
        WScript.Echo Space(4) & ".Description: " & .Description
        WScript.Echo Space(4) & ".Vendor: " & .Vendor
        WScript.Echo Space(4) & ".ServerCategories.ToString(): " &
.ServerCategories.ToString()
        WScript.Echo Space(4) & ".VersionIndependentProgId: " &
.VersionIndependentProgId
    End With
Next
```

5.1.2.3.2 Browsing for OPC Classic Nodes (Branches and Leaves)

OPC Address Space Organization

Items in an OPC server are typically organized in a tree hierarchy (address space), where the branch nodes serve organizational purposes (similar to folders in a file system), while the leaf nodes correspond to actual pieces of data that can be accessed (similar to files in a file system) – the OPC items. Each node has a “short” name that is unique among other branches or leaves under the same parent branch (or a root). Leaf nodes can be fully identified using a “long” ItemID, which determines the OPC item without a need to further qualify it with its position in the tree. ItemIDs may look like “Device1.Block101.Setpoint”, however their syntax and meaning is fully determined by the particular OPC server they are coming from.

In This Topic

Browse Methods

OPC Data Client gives you methods to traverse through the address space information and obtain the information available there. It is also possible to filter the returned nodes by various criteria, such as node name matching certain pattern, or a particular data type only, or writeable items only, etc.

If you want to retrieve a list of all sub-branches under a given branch (or under a root) of the OPC server, call the BrowseBranches method. You will receive back a DANodeElementCollection object. Each DANodeElement contains information gathered about one sub-branch node, such as its name, or indication whether it has children.

VBScript

```
Rem This example shows how to obtain all branches at the root of the address space. For
each branch, it displays whether
Rem it may have child nodes.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim BranchElements: Set BranchElements = Client.BrowseBranches("",
"OPCLabs.KitServer.2", "")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim BranchElement: For Each BranchElement In BranchElements
    WScript.Echo "BranchElements(""" & BranchElement.Name & """).HasChildren: " &
BranchElement.HasChildren
Next
```

Similarly, if you want to retrieve a list of leaves under a given branch (or under a root) of the OPC server, call the BrowseLeaves method. You will also receive back a DANodeElementCollection object, this time containing the leaves only. You can find information such as the Item ID from the DANodeElement of any leaf, and pass it further to methods like ReadItem or SubscribeItem.

VBScript

```
Rem This example shows how to obtain all leaves under the "Simulation" branch of the
address space. For each leaf, it displays
Rem the ItemID of the node.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim LeafElements: Set LeafElements = Client.BrowseLeaves("", "OPCLabs.KitServer.2",
"Simulation")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
```

```
    WScript.Quit
End If
On Error Goto 0

Dim LeafElement: For Each LeafElement In LeafElements
    WScript.Echo "LeafElements(""" & LeafElement.Name & """).ItemId: " &
LeafElement.ItemId
Next
```

The most generic address space browsing method is BrowseNodes. It combines the functionality of BrowseBranches and BrowseLeaves, and it also allows the widest range of filtering options by passing in an argument of type DABrowseParameters, or individual arguments for data type filter and access rights filter.

Object Pascal

```
// This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
// whether the node is a branch or a leaf.

class procedure BrowseNodes.Main;
var
  BrowseParameters: _DABrowseParameters;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Count: Cardinal;
  Element: OleVariant;
  ServerDescriptor: _ServerDescriptor;
  NodeDescriptor: _DANodeDescriptor;
  NodeElement: _DANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _DANodeElementCollection;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';

  NodeDescriptor := CoDANodeDescriptor.Create;
  NodeDescriptor.ItemId := 'Simulation';

  BrowseParameters := CoDABrowseParameters.Create;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    NodeElements := Client.BrowseNodes(
      ServerDescriptor,
      NodeDescriptor,
      BrowseParameters);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
```

```
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _DANodeElement;
//    WriteLn(NodeElement.Name, ': ', NodeElement.ItemId);
    WriteLn('BrowseElements("', NodeElement.Name, '"):');
    WriteLn('    .IsBranch: ', NodeElement.IsBranch);
    WriteLn('    .IsLeaf: ', NodeElement.IsLeaf);
  end;
end;
```

PHP

```
// This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
// whether the node is a branch or a leaf.

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";

$NodeDescriptor = new COM("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor");
$NodeDescriptor->ItemID = "Simulation";

$BrowseParameters = new COM("OpcLabs.EasyOpc.DataAccess.DABrowseParameters");

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $NodeElements = $Client->BrowseNodes($ServerDescriptor, $NodeDescriptor,
$BrowseParameters);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

foreach ($NodeElements as $NodeElement)
{
    printf("NodeElements(\"%s\"):\n", $NodeElement->Name);
    printf("    .IsBranch: %s\n", $NodeElement->IsBranch ? 'true' : 'false');
    printf("    .IsLeaf: %s\n", $NodeElement->IsLeaf ? 'true' : 'false');
}
```

PowerScript

```
// This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
// whether the node is a branch or a leaf.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
```

```
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare the arguments

OLEObject serverDescriptor
serverDescriptor = CREATE OLEObject
serverDescriptor.ConnectToNewObject("OpcLabs.EasyOpc.ServerDescriptor")
serverDescriptor.ServerClass = "OPCLabs.KitServer.2"

OLEObject nodeDescriptor
nodeDescriptor = CREATE OLEObject
nodeDescriptor.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
nodeDescriptor.ItemID = "Simulation"

OLEObject browseParameters
browseParameters = CREATE OLEObject
browseParameters.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.DABrowseParameters")

// Perform the operation
OLEObject nodeElements
TRY
    nodeElements = client.BrowseNodes(serverDescriptor, nodeDescriptor,
browseParameters)
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

OLEObject nodeElementList
nodeElementList = nodeElements.ToList()

// Display results
Int i
FOR i = 0 TO nodeElementList.Count - 1
    OLEObject nodeElement
    nodeElement = nodeElementList.Item[i]
    mle_outputtext.Text = mle_outputtext.Text + 'NodeElements("' + nodeElement.Name +
'"):' + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "    nodeElement.IsBranch: " +
String(nodeElement.IsBranch) + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "    nodeElement.IsLeaf: " +
String(nodeElement.IsLeaf) + "~r~n"
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

VBScript

```
Rem This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
Rem whether the node is a branch or a leaf.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
```

```
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation"

Dim BrowseParameters: Set BrowseParameters =
CreateObject("OpcLabs.EasyOpc.DataAccess.DABrowseParameters")

Dim Client: Set Client= CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseNodes(ServerDescriptor,
NodeDescriptor, BrowseParameters)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo "NodeElements(""" & NodeElement.Name & """):"
    With NodeElement
        WScript.Echo Space(4) & ".IsBranch: " & .IsBranch
        WScript.Echo Space(4) & ".IsLeaf: " & .IsLeaf
    End With
Next
```

More Examples

C#

```
// This example shows how to recursively browse the nodes in the OPC address space.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class BrowseNodes
    {
        public static void Recursive()
        {
            var stopwatch = new Stopwatch();
            stopwatch.Start();

            var client = new EasyDAClient();
            _branchCount = 0;
            _leafCount = 0;
```

```
            try
            {
                BrowseFromNode(client, "OPCLabs.KitServer.2", "");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            stopwatch.Stop();
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds);
            Console.WriteLine("Branch count: {0}", _branchCount);
            Console.WriteLine("Leaf count: {0}", _leafCount);
        }

        private static void BrowseFromNode(
            EasyDAClient client,
            ServerDescriptor serverDescriptor,
            DANodeDescriptor parentNodeDescriptor)
        {
            Debug.Assert(client != null);
            Debug.Assert(serverDescriptor != null);
            Debug.Assert(parentNodeDescriptor != null);

            // Obtain all node elements under parentNodeDescriptor
            var browseParameters = new DABrowseParameters();    // no filtering
whatsoever
            DANodeElementCollection nodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters);
            // Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            // it, here omitted for brevity.

            foreach (DANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine(nodeElement);

                // If the node is a branch, browse recursively into it.
                if (nodeElement.IsBranch)
                {
                    _branchCount++;
                    BrowseFromNode(client, serverDescriptor, nodeElement);
                }
                else
                {
                    _leafCount++;
                }
            }
        }

        private static int _branchCount;
```

```
        private static int _leafCount;
    }
}
```

VB.NET

```
' This example shows how to recursively browse the nodes in the OPC address space.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class BrowseNodes

        Shared Sub Recursive()
            Dim stopwatch = New Stopwatch()
            stopwatch.Start()

            Dim client = New EasyDAClient()
            _branchCount = 0
            _leafCount = 0

            Try
                BrowseFromNode(client, "OPCLabs.KitServer.2", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            stopwatch.Stop()
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds)
            Console.WriteLine("Branch count: {0}", _branchCount)
            Console.WriteLine("Leaf count: {0}", _leafCount)
        End Sub

        Private Shared Sub BrowseFromNode( _
            client As EasyDAClient,
            serverDescriptor As ServerDescriptor,
            parentNodeDescriptor As DANodeDescriptor)

            Debug.Assert(client IsNot Nothing)
            Debug.Assert(serverDescriptor IsNot Nothing)
            Debug.Assert(parentNodeDescriptor IsNot Nothing)

            ' Obtain all node elements under parentNodeDescriptor
            Dim browseParameters = New DABrowseParameters() ' no filtering whatsoever
            Dim nodeElementCollection As DANodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters)
            ' Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            ' it, here omitted for brevity.
```

```
            For Each nodeElement As DANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine(nodeElement)

                ' If the node is a branch, browse recursively into it.
                If nodeElement.IsBranch Then
                    _branchCount += 1
                    BrowseFromNode(client, serverDescriptor, nodeElement)
                Else
                    _leafCount += 1
                End If
            Next nodeElement
        End Sub

        Private Shared _branchCount As Integer
        Private Shared _leafCount As Integer
    End Class
End Namespace
```

C#

```
// This example shows how to recursively browse the nodes in the OPC XML-DA address
space.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class BrowseNodes
    {
        public static void RecursiveXml()
        {
            var stopwatch = new Stopwatch();
            stopwatch.Start();

            var client = new EasyDAClient();
            _branchCount = 0;
            _leafCount = 0;

            try
            {
                BrowseFromNode(client, "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx", "");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
```

```
            stopwatch.Stop();
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds);
            Console.WriteLine("Branch count: {0}", _branchCount);
            Console.WriteLine("Leaf count: {0}", _leafCount);
        }

        private static void BrowseFromNode(
            EasyDAClient client,
            ServerDescriptor serverDescriptor,
            DANodeDescriptor parentNodeDescriptor)
        {
            Debug.Assert(client != null);
            Debug.Assert(serverDescriptor != null);
            Debug.Assert(parentNodeDescriptor != null);

            // Obtain all node elements under parentNodeDescriptor
            var browseParameters = new DABrowseParameters();    // no filtering
whatsoever
            DANodeElementCollection nodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters);
            // Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            // it, here omitted for brevity.

            foreach (DANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine(nodeElement);

                // If the node is a branch, browse recursively into it.
                if (nodeElement.IsBranch)
                {
                    _branchCount++;
                    BrowseFromNode(client, serverDescriptor, nodeElement);
                }
                else
                {
                    _leafCount++;
                }
            }
        }

        private static int _branchCount;
        private static int _leafCount;
    }
}
```

VB.NET

```
' This example shows how to recursively browse the nodes in the OPC XML-DA address
space.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
```

```
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class BrowseNodes

        Shared Sub RecursiveXml()
            Dim stopwatch = New Stopwatch()
            stopwatch.Start()

            Dim client = New EasyDAClient()
            _branchCount = 0
            _leafCount = 0

            Try
                BrowseFromNode(client, "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            stopwatch.Stop()
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds)
            Console.WriteLine("Branch count: {0}", _branchCount)
            Console.WriteLine("Leaf count: {0}", _leafCount)
        End Sub

        Private Shared Sub BrowseFromNode( _
            client As EasyDAClient,
            serverDescriptor As ServerDescriptor,
            parentNodeDescriptor As DANodeDescriptor)

            Debug.Assert(client IsNot Nothing)
            Debug.Assert(serverDescriptor IsNot Nothing)
            Debug.Assert(parentNodeDescriptor IsNot Nothing)

            ' Obtain all node elements under parentNodeDescriptor
            Dim browseParameters = New DABrowseParameters() ' no filtering whatsoever
            Dim nodeElementCollection As DANodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters)
            ' Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            ' it, here omitted for brevity.

            For Each nodeElement As DANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine(nodeElement)

                ' If the node is a branch, browse recursively into it.
                If nodeElement.IsBranch Then
                    _branchCount += 1
                    BrowseFromNode(client, serverDescriptor, nodeElement)
```

```
                Else
                    _leafCount += 1
                End If
            Next nodeElement
        End Sub

        Private Shared _branchCount As Integer
        Private Shared _leafCount As Integer
    End Class
End Namespace
```

C#

```
// Recursively browses and displays the nodes in the OPC address space, and attempts to
read and display values of all OPC
// items it finds.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class BrowseNodes
    {
        const string ServerClass = "OPCLabs.KitServer.2";

        static readonly EasyDAClient Client = new EasyDAClient();

        static void BrowseAndReadFromNode(string parentItemId)
        {
            // Obtain all node elements under parentItemId
            var browseParameters = new DABrowseParameters(); // no filtering whatsoever
            DANodeElementCollection nodeElementCollection = Client.BrowseNodes("",
ServerClass, parentItemId,
                browseParameters);
            // Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for it, here
            // omitted for brevity.

            foreach (DANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);

                // If the node is a leaf, it might be possible to read from it
                if (nodeElement.IsLeaf)
                {
                    // Determine what the display - either the value read, or exception
message in case of failure.
                    string display;
                    try
                    {
                        object value = Client.ReadItemValue("", ServerClass,
nodeElement);
                        display = String.Format("{0}", value);
```

```
                    }
                    catch (OpcException exception)
                    {
                        display = String.Format("** {0} **",
exception.GetBaseException().Message);
                    }

                    Console.WriteLine("{0} -> {1}", nodeElement.ItemId, display);
                }
                // If the node is not a leaf, just display its itemId
                else
                    Console.WriteLine("{0}", nodeElement.ItemId);

                // If the node is a branch, browse recursively into it.
                if (nodeElement.IsBranch &&
                    (nodeElement.ItemId != "SimulateEvents")
                    /* this branch is too big for the purpose of this example */)
                    BrowseAndReadFromNode(nodeElement);
            }
        }

        public static void RecursiveWithRead()
        {
            Console.WriteLine("Browsing and reading values...");
            // Set timeout to only wait 1 second - default would be 1 minute to wait
for good quality that may never come.
            Client.InstanceParameters.Timeouts.ReadItem = 1000;

            // Do the actual browsing and reading, starting from root of OPC address
space (denoted by empty string for itemId)
            try
            {
                BrowseAndReadFromNode("");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' Recursively browses and displays the nodes in the OPC address space, and attempts to
read and display values of all OPC
' items it finds.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class BrowseNodes
        Private Const ServerClass As String = "OPCLabs.KitServer.2"
```

```
        Private Shared ReadOnly Client As New EasyDAClient()

        Private Shared Sub BrowseAndReadFromNode(parentItemId As String)
            ' Obtain all node elements under parentItemId
            Dim browseParameters = New DABrowseParameters() ' no filtering whatsoever
            Dim nodeElementCollection As DANodeElementCollection =
Client.BrowseNodes("", ServerClass, parentItemId, browseParameters)
            ' Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for it, here
            ' omitted for brevity.

            For Each nodeElement As DANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)

                ' If the node is a leaf, it might be possible to read from it
                If nodeElement.IsLeaf Then
                    ' Determine what the display - either the value read, or exception
message in case of failure.
                    Dim display As String
                    Try
                        Dim value As Object = Client.ReadItemValue("", ServerClass,
nodeElement.ItemId)
                        display = String.Format("{0}", value)
                    Catch exception As OpcException
                        display = String.Format("** {0} **",
exception.GetBaseException().Message)
                    End Try

                    Console.WriteLine("{0} -> {1}", nodeElement.ItemId, display)
                    ' If the node is not a leaf, just display its itemId
                Else
                    Console.WriteLine("{0}", nodeElement.ItemId)
                End If

                ' If the node is a branch, browse recursively into it.
                If nodeElement.IsBranch AndAlso (nodeElement.ItemId <>
"SimulateEvents") Then ' this branch is too big for the purpose of this example
                    BrowseAndReadFromNode(nodeElement.ItemId)
                End If
            Next nodeElement
        End Sub

        Shared Sub RecursiveWithRead()
            Console.WriteLine("Browsing and reading values...")
            ' Set timeout to only wait 1 second - default would be 1 minute to wait for
good quality that may never come.
            Client.InstanceParameters.Timeouts.ReadItem = 1000

            ' Do the actual browsing and reading, starting from root of OPC address
space (denoted by empty string for itemId)
            Try
                BrowseAndReadFromNode("")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
```

```
        End Sub
    End Class
End Namespace
```

5.1.2.3.3 Browsing for OPC Classic Access Paths

> Access paths are somewhat obsolete feature of OPC Data Access specification, and few OPC servers actually use it; but if a particular OPC server does use access paths, specifying the proper access path together with ItemID may be the only way to retrieve the data you want.

If you want to retrieve a list of possible access paths available for a specific OPC item, call the BrowseAccessPaths method, passing it the information about the OPC server, and the ItemID. You will receive back an array of strings; each element of this array is an access path that you can use with methods such as ReadItem or SubscribeItem.

In OPC Data Client.NET, you can also pass the access path to a constructor of DAItemDescriptor object and later use that descriptor with various methods.

VBScript

```
Rem This example shows how to obtain all access paths available for an item.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.Random"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim accessPaths: accessPaths = Client.BrowseAccessPaths(ServerDescriptor,
NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim i: For i = LBound(accessPaths) To UBound(accessPaths)
    WScript.Echo "accessPaths(" & i & "): " & accessPaths(i)
Next
```

5.1.2.3.4 Browsing for OPC Classic Properties

Each OPC item has typically associated a set of OPC properties with it. OPC properties contain additional information related to the item. The OPC specifications define a set of common properties; however, each OPC server is free to implement some more, vendor-specific properties as well.

If you want to retrieve a list of all properties available on a given OPC item, call the BrowseProperties method, passing it the ItemID you are interested in. You will receive back a DAPropertyElementCollection object. Each DAPropertyElement contains information about one OPC property, such as its (numeric) PropertyId, data type, or a readable description. The PropertyId can be later used as an argument in calling methods such as GetPropertyValue.

You can use DAPropertyId.GetName and GetPropertyType methods to obtain the string identifier of the property, or its type. With OPC XML, properties are identified by a XML qualified name, and you will find it in the DAPropertyElement.QualifiedName property.

Constants for specific (well-known) OPC properties are contained in the DAPropertyIds enumeration.

Object Pascal

```
// This example shows how to enumerate all properties of an OPC item. For each
property, it displays its Id and description.

class procedure BrowseProperties.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Count: Cardinal;
  Element: OleVariant;
  ServerDescriptor: _ServerDescriptor;
  NodeDescriptor: _DANodeDescriptor;
  PropertyElement: _DAPropertyElement;
  PropertyElementEnumerator: IEnumVariant;
  PropertyElements: _DAPropertyElementCollection;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';

  NodeDescriptor := CoDANodeDescriptor.Create;
  NodeDescriptor.ItemId := 'Simulation.Random';

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    PropertyElements := Client.BrowseProperties(
      ServerDescriptor,
      NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  PropertyElementEnumerator := PropertyElements.GetEnumerator;
  while (PropertyElementEnumerator.Next(1, Element, Count) = S_OK) do
```

```
  begin
    PropertyElement := IUnknown(Element) as _DAPropertyElement;
    WriteLn('PropertyElements("', PropertyElement.PropertyId.NumericalValue,
'").Description: ', PropertyElement.Description);
  end;
end;
```

PHP

```
// This example shows how to enumerate all properties of an OPC item. For each
property, it displays its Id and description.

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";

$NodeDescriptor = new COM("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor");
$NodeDescriptor->ItemID = "Simulation.Random";

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $PropertyElements = $Client->BrowseProperties($ServerDescriptor, $NodeDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

foreach ($PropertyElements as $PropertyElement)
{
    printf("PropertyElements(\"%s\").Description: %s\n", $PropertyElement->PropertyID-
>NumericalValue, $PropertyElement->Description);
}
```

VBScript

```
Rem This example shows how to enumerate all properties of an OPC item. For each
property, it displays its Id and description.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.Random"

Dim EasyDAClient: Set EasyDAClient =
CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim PropertyElements: Set PropertyElements =
EasyDAClient.BrowseProperties(ServerDescriptor, NodeDescriptor)
If Err.Number <> 0 Then
```

```
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim PropertyElement: For Each PropertyElement In PropertyElements
    WScript.Echo "PropertyElements(""" & PropertyElement.PropertyID.NumericalValue &
""").Description: " & PropertyElement.Description
Next
```

5.1.2.3.5 Discovering OPC UA Servers

OPC UA Discovery allows OPC Data Client to find available OPC UA servers, and determine how to make connections to them. There are several approaches to discovery, and they can be combined together to achieve the desired functionality. Clients and servers can be on the same host, on different hosts in the same subnet, or even on completely different locations in an administrative domain.

OPC Data Client provides following kinds of discovery:

In This Topic

- **OPC UA Local Discovery (Section 5.1.2.3.5.1)**. Obtains information about the servers on a specified host using OPC UA Local Discovery Server (LDS) running on that host.
- **OPC UA Network Discovery (Section 5.1.2.3.5.2)**. Also known as MulticastSubnet Discovery. The information is obtained from a Local Discovery Server with Multicast Extensions (LDS-ME), which uses the mDNS protocol to exchange the information about OPC UA servers with other LDS-ME-s on the network (or multiple networks).
- **OPC UA Global Discovery (Section 5.1.2.3.5.3)**. Obtains information about applications available in an administrative domain by interrogating an OPC UA Global Discovery Server (GDS).

It is also possible to access the discovery functionality in a common way, using **Generalized OPC UA Discovery (Section 5.1.2.3.5.4)**.

Flattened and unflattened results

By default, the results (the application element collection) of many of the methods described in this chapter are returned *flattened*. This means that a single OPC UA application may appear multiple times in the resulting collection, and each time it will have precisely one DiscoveryUriString associated with it.

Alternatively, you can use method overloads with the additional 'flat' argument, and set it true, in order to receive *unflattened* result. In this case, each OPC UA application will be contained just once in the resulting collection, but it may have one or more discovery URL in its DiscoveryUriStrings property. In this case, do not use the DiscoveryUriString property, because it will contain an empty string if the application has more than one discovery URL.

5.1.2.3.5.1 OPC UA Local Discovery

OPC Data Client-UA allows you to work with so called OPC-UA Local Discovery Server(s). OPC-UA Local Discovery Server is a special OPC-UA service that provides information about other OPC-UA servers available.

In This Topic

> ⚠ The term "local discovery" might be misleading sometimes. OPC UA Local Discovery is not just for discovering OPC UA servers that reside on the same computer as where your OPC UA client is. OPC UA Local Discover allows you to discover servers on a remote computer as well, provided that you know the host name of the computer, and the OPC UA Local Discovery Server is running on that remote computer.

If you want to retrieve a list of OPC Unified Architecture servers registered on a local or remote computer, call the DiscoverLocalServers method, passing it the name or address of the remote machine.

You will receive back a UADiscoveryElementCollection object, which is collection of UADiscoveryElement-s. Each element contains a server's discovery URL. The discovery URL is the main piece of information that you can further use if you want to connect to that OPC-UA server. Each UADiscoveryElement contains information gathered about one OPC server found by the discovery process, including things like the application name, application type, its Product URI etc.

C#

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on a given machine.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class DiscoverLocalServers
    {
        public static void Overload1()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of server elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverLocalServers("opcua.demo-
this.com");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
                Console.WriteLine("discoveryElementCollection[\"
{0}\"].ApplicationUriString: {1}",
                    discoveryElement.DiscoveryUriString,
```

```
discoveryElement.ApplicationUriString);

            // Example output:
            // discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
            // discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        }
    }
}
```

VB.NET

```
' This example shows how to obtain application URLs of all OPC Unified Architecture
servers on a given machine.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class DiscoverLocalServers
        Public Shared Sub Overload1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of server elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverLocalServers("opcua.demo-
this.com")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine("discoveryElementCollection[""
{0}""].ApplicationUriString: {1}", _
                                  discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString)
            Next discoveryElement

            ' Example output:
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
            'discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "DiscoverLocalServers.h"

namespace _EasyUAClient
{
    void DiscoverLocalServers::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Obtain collection of server elements
            _UADiscoveryElementCollectionPtr DiscoveryElementsPtr = ClientPtr-
>DiscoverLocalServers(L"opcua.demo-this.com");

            // Display results
            IEnumVARIANTPtr EnumDiscoveryElementPtr = DiscoveryElementsPtr-
>GetEnumerator();
            _variant_t vDiscoveryElement;
            while (EnumDiscoveryElementPtr->Next(1, &vDiscoveryElement, NULL) == S_OK)
            {
                _UADiscoveryElementPtr DiscoveryElementPtr(vDiscoveryElement);
                _tprintf(_T("DiscoveryElementCollection[\"%s\"]: "),
(LPCTSTR)CW2CT(DiscoveryElementPtr->DiscoveryUriString));
                _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(DiscoveryElementPtr-
>ApplicationUriString));
                vDiscoveryElement.Clear();
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

class procedure DiscoverLocalServers.Main;
var
  Client: EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of server elements
```

```
  DiscoveryElements := Client.DiscoverLocalServers('opcua.demo-this.com');

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn(
      'DiscoveryElements["',
      DiscoveryElement.DiscoveryUriString,
      '".ApplicationUriString: ',
      DiscoveryElement.ApplicationUriString);
  end;
end;
```

Object Pascal

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

class procedure DiscoverLocalServers.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of server elements
  try
    DiscoveryElements := Client.DiscoverLocalServers('opcua.demo-this.com');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn(
      'DiscoveryElements["',
      DiscoveryElement.DiscoveryUriString,
      '".ApplicationUriString: ',
      DiscoveryElement.ApplicationUriString);
  end;
end;
```

PHP

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain collection of server elements
try
{
    $DiscoveryElementCollection = $Client->DiscoverLocalServers("opcua.demo-this.com");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}

// Display results
foreach ($DiscoveryElementCollection as $DiscoveryElement)
{
    printf("DiscoveryElementCollection[\"%s\"].ApplicationUriString: %s\n",
        $DiscoveryElement->DiscoveryUriString, $DiscoveryElement-
>ApplicationUriString);
}
```

Python

```
# This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Obtain collection of server elements
discoveryElementCollection = client.DiscoverLocalServers('opcua.demo-this.com')

# Display results
for discoveryElement in discoveryElementCollection:
    print('DiscoveryElementCollection["', discoveryElement.DiscoveryUriString,
'"].ApplicationUriString: ',
          discoveryElement.ApplicationUriString)
```

Visual Basic (VB 6.)

```
Rem This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

Private Sub DiscoverLocalServers_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient
```

```
    ' Obtain collection of server elements
    On Error Resume Next
    Dim DiscoveryElementCollection As UADiscoveryElementCollection
    Set DiscoveryElementCollection = Client.DiscoverLocalServers("opcua.demo-this.com")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    Dim DiscoveryElement As UADiscoveryElement: For Each DiscoveryElement In
DiscoveryElementCollection
        OutputText = OutputText & "DiscoveryElementCollection[""" &
DiscoveryElement.DiscoveryUriString & """].ApplicationUriString: " & _
            DiscoveryElement.applicationUriString & vbCrLf
    Next
End Sub
```

VBScript

```
Rem This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain collection of server elements
On Error Resume Next
Dim DiscoveryElementCollection: Set DiscoveryElementCollection =
Client.DiscoverLocalServers("opcua.demo-this.com")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim DiscoveryElement: For Each DiscoveryElement In DiscoveryElementCollection
    WScript.Echo "DiscoveryElementCollection[""" & DiscoveryElement.DiscoveryUriString
& """].ApplicationUriString: " & _
        DiscoveryElement.ApplicationUriString
Next
```

In .NET, if you want to connect to the discovered OPC server later in your code by calling other methods, use the built-in conversion of UADiscoveryElement to UAEndpointDescriptor, and pass the resulting object as an endpointDescriptor argument either directly to some other method call.

When performing a local discovery of OPC-UA servers, the component attempts to connect to multiple possible discovery endpoints in parallel, speeding up the discovery. This behavior is controlled by the ParallelDiscovery property in the UADiscoveryParameters object. The endpoints that the components attempts to use for discovery on a given host (either sequentially, or in parallel), are given by the DiscoveryUriTemplateStrings property of the UAHostParameters.

You can also bypass the use of UAHostParameters, and call the DiscoverLocalServers overload that takes an input array or URL strings to attempt discovery on.

There are also DiscoverLocalApplications methods, which allow you to specify the application types you are interested in, using the combination of flags from the UAApplicationTypes enumeration. This can be useful e.g. if you want to do hierarchical discovery and return also the discovery servers (which are not returned by default).

The FindLocalApplications method can be used, if you want to interrogate specific discovery endpoints.

C#

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class FindLocalApplications
    {
        public static void Main1()
        {

            string[] discoveryUriStrings =
            {
                "opc.tcp://opcua.demo-this.com:4840/UADiscovery",
                "http://opcua.demo-this.com/UADiscovery/Default.svc",
                "http://opcua.demo-this.com:52601/UADiscovery"
            };

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection =
client.FindLocalApplications(discoveryUriStrings, UAApplicationTypes.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
                Console.WriteLine("discoveryElementCollection[\"
{0}\"].ApplicationUriString: {1}",
                    discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString);
```

```
            // Example output:
            //discoveryElementCollection["http://opcua.demo-
this.com:62543/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            //discoveryElementCollection["opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            //discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            //discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            //discoveryElementCollection["https://opcua.demo-
this.com:51212/UA/SampleServer/"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            //discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

class procedure FindLocalApplications.Main;
const
  UAApplicationTypes_Server = 1;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
  DiscoveryUriStrings: OleVariant;
begin
  DiscoveryUriStrings := VarArrayCreate([0, 2], varVariant);
  DiscoveryUriStrings[0] := 'opc.tcp://opcua.demo-this.com:4840/UADiscovery';
  DiscoveryUriStrings[1] := 'http://opcua.demo-this.com/UADiscovery/Default.svc';
  DiscoveryUriStrings[2] := 'http://opcua.demo-this.com:52601/UADiscovery';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of application elements
  try
    DiscoveryElements := Client.FindLocalApplications(DiscoveryUriStrings,
UAApplicationTypes_Server);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
```

```
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn(
      'DiscoveryElements["',
      DiscoveryElement.DiscoveryUriString,
      '".ApplicationUriString: ',
      DiscoveryElement.ApplicationUriString);
  end;
end;
```

VB.NET

```
' This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class FindLocalApplications
        Public Shared Sub Main1()
            Dim discoveryUriStrings() As String =
            { _
                "opc.tcp://opcua.demo-this.com:4840/UADiscovery", _
                "http://opcua.demo-this.com/UADiscovery/Default.svc", _
                "http://opcua.demo-this.com:52601/UADiscovery" _
            }

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection =
client.FindLocalApplications(discoveryUriStrings, UAApplicationTypes.Server)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine("discoveryElementCollection[""
{0}""].ApplicationUriString: {1}", _
                    discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString)
            Next discoveryElement
```

```
            ' Example output:
            'discoveryElementCollection["http://opcua.demo-
this.com:62543/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            'discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            'discoveryElementCollection["https://opcua.demo-
this.com:51212/UA/SampleServer/"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

Option Explicit

Const UAApplicationTypes_Server = 1

Dim discoveryUriStrings(2)
discoveryUriStrings(0) = "opc.tcp://opcua.demo-this.com:4840/UADiscovery"
discoveryUriStrings(1) = "http://opcua.demo-this.com/UADiscovery/Default.svc"
discoveryUriStrings(2) = "http://opcua.demo-this.com:52601/UADiscovery"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain collection of application elements
On Error Resume Next
Dim DiscoveryElementCollection: Set DiscoveryElementCollection =
Client.FindLocalApplications(discoveryUriStrings, UAApplicationTypes_Server)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim DiscoveryElement: For Each DiscoveryElement In DiscoveryElementCollection
    WScript.Echo "DiscoveryElementCollection[""" & DiscoveryElement.DiscoveryUriString
& """].ApplicationUriString: " & _
        DiscoveryElement.ApplicationUriString
Next
```

Note that QuickOPC returns UADiscoveryElementCollection from all OPC UA discovery methods, but the amount of information contained in each returned UADiscoveryElement depends on the discovery approach used. For OPC UA Local Discovery, following primary properties of each UADiscoveryElement are filled in:

- ApplicationName
- ApplicationType
- ApplicationUriString
- DiscoveryProfileUriString
- DiscoveryUriString (see further down for more information)
- DiscoveryUriStrings (see further down for more information)
- GatewayServerUriString
- ProductUri

> ⚠ OPC UA Local Discovery Server (LDS) is not part of OPC Data Client installation.

5.1.2.3.5.2 OPC UA Network Discovery

OPC Data Client-UA allows you to obtain information about OPC UA servers from a Local Discovery Server with Multicast Extensions (LDS-ME), which uses the mDNS protocol to exchange the information about OPC UA servers with other LDS-ME-s on the network (or multiple networks).

In .NET, if you want to retrieve a list of OPC Unified Architecture servers available on your network, call the DiscoverNetworkServers method without further parameters. This will obtain the information about servers using the LDS-ME installed on the machine where the component is running, and without further restricting (filtering) them.

Further in .NET, if you want to retrieve a list of OPC Unified Architecture servers available on your network, call the DiscoverNetworkServers method with one or two arguments, passing it

- a StringCollection which specifies the server capabilities you are looking for,
- the name of the machine where the LDS-ME is running,
- or both.

In COM, the DiscoverNetworkServers method has two arguments, and therefore if you want no filtering, pass a null reference (or an empty collection) as the first argument, and pass "localhost" as the second argument if you want to use the LDS-ME installed on your machine.

In all cases described above, you will receive back a UADiscoveryElementCollection object, which is collection of UADiscoveryElement-s. Each element contains a server's discovery URL. The discovery URL is the main piece of information that you can further use if you want to connect to that OPC-UA server. Each UADiscoveryElement contains information gathered about one OPC server found by the discovery process, including things like the server name, application type, and - mainly - its DiscoveryUriString.

C#

```
// This example shows how to obtain information about OPC UA servers available on the
network.
// The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
```

```
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class DiscoverNetworkServers
    {
        public static void Main1()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString);
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString);
            }
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain information about OPC UA servers available on the
network.

class procedure DiscoverNetworkServers.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
```

```
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of application elements
  try
    DiscoveryElements := Client.DiscoverNetworkServers(Unassigned, 'opcua.demo-
this.com');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn;
    WriteLn('Server name: ', DiscoveryElement.ServerName);
    WriteLn('Discovery URI string: ', DiscoveryElement.DiscoveryUriString);
    WriteLn('Server capabilities: ', DiscoveryElement.ServerCapabilities.ToString);
  end;
end;
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers available on the
network.
' The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverNetworkServers
        Public Shared Sub Main1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
```

```
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString)
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString)
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to obtain information about OPC UA servers available on the
network.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "DiscoverNetworkServers.h"

namespace _EasyUAClient
{
    void DiscoverNetworkServers::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            _StringCollectionPtr ServerElementFilterPtr(__uuidof(StringCollection));
            _variant_t vServerElementFilter(ServerElementFilterPtr.GetInterfacePtr());

            // Obtain collection of application elements
            _UADiscoveryElementCollectionPtr DiscoveryElementsPtr =
                ClientPtr->DiscoverNetworkServers(vServerElementFilter, L"opcua.demo-
this.com");

            // Display results
            IEnumVARIANTPtr EnumDiscoveryElementPtr = DiscoveryElementsPtr-
>GetEnumerator();
            _variant_t vDiscoveryElement;
            while (EnumDiscoveryElementPtr->Next(1, &vDiscoveryElement, NULL) == S_OK)
            {
                _UADiscoveryElementPtr DiscoveryElementPtr(vDiscoveryElement);
                _tprintf(_T("\n"));
                _tprintf(_T("Application type: %d\n"), DiscoveryElementPtr-
>ApplicationType);
                _tprintf(_T("Server name: %s\n"), (LPCTSTR)CW2CT(DiscoveryElementPtr-
>ServerName));
                _tprintf(_T("Discovery URI string: %s\n"),
(LPCTSTR)CW2CT(DiscoveryElementPtr->DiscoveryUriString));
                _tprintf(_T("Server capabilities: "));
```

```
                IEnumVARIANTPtr EnumServerCapabilityPtr = DiscoveryElementPtr-
>ServerCapabilities->GetEnumerator();
                _variant_t vServerCapability = NULL;
                while (EnumServerCapabilityPtr->Next(1, &vServerCapability, NULL) ==
S_OK)
                    _tprintf(_T("%s "), (LPCTSTR)CW2CT(_bstr_t(vServerCapability)));
                _tprintf(_T("\n"));
                vDiscoveryElement.Clear();
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

VBScript

```
Rem This example shows how to obtain information about OPC UA servers available on the
network.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain collection of application elements
On Error Resume Next
Dim DiscoveryElementCollection: Set DiscoveryElementCollection = _
    Client.DiscoverNetworkServers(Nothing, "opcua.demo-this.com")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim DiscoveryElement: For Each DiscoveryElement In DiscoveryElementCollection
    WScript.Echo
    WScript.Echo "Server name: " & DiscoveryElement.ServerName
    WScript.Echo "Discovery URI string: " & DiscoveryElement.DiscoveryUriString
    Dim s: s = ""
    Dim serverCapability: For Each serverCapability In
DiscoveryElement.ServerCapabilities
        s = s & serverCapability & " "
    Next
    WScript.Echo "Server capabilities: " & s
Next
```

You can also obtain the results unflattened:

C#

```
// This example shows how to obtain information about OPC UA servers available on the
network.
// The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.
```

```
using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class DiscoverNetworkServers
    {
        public static void Hierarchical()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com", flat:false);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI strings:");
                foreach (string discoveryUriString in
discoveryElement.DiscoveryUriStrings)
                    Console.WriteLine("  {0}", discoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers available on the
network.
' The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
```

```
    Partial Friend Class DiscoverNetworkServers
        Public Shared Sub Hierarchical()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com", flat:=False)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI strings:")
                For Each discoveryUriString As String In
discoveryElement.DiscoveryUriStrings
                    Console.WriteLine("  {0}", discoveryUriString)
                Next discoveryUriString
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

Note that QuickOPC returns UADiscoveryElementCollection from all OPC UA discovery methods, but the amount of information contained in each returned UADiscoveryElement depends on the discovery approach used. For OPC UA Global Discovery and OPC UA Network Discovery, only following primary properties of each UADiscoveryElement are filled in:

- ApplicationType
- DiscoveryUriString
- ServerCapabilities
- ServerName

⚠ OPC UA Local Discovery Server with Multicast Extensions (LDS-ME) is not part of OPC Data Client installation.

5.1.2.3.5.3 OPC UA Global Discovery

OPC Data Client-UA allows you to query information contained in so called OPC-UA Global Discovery Server(s). A Global Discovery Server (GDS) maintains discovery information for OPC UA applications available in an administrative domain.

If you want to retrieve a list of OPC Unified Architecture servers in an administrative domain, call the DiscoverGlobalServers method, passing it the endpoint of the GDS.

In addition, you can call an overload of the DiscoverGlobalServers method that also takes a filter parameter of

type UAQueryServerFilter. Using this filter, you can limit the search only to servers that meet the specified criteria. The criteria include:

- Application name.
- Application URI string.
- Product URI string.
- Server capabilities.

You can also call the EasyUAClient.DiscoverGlobalServers method that takes no arguments, in which case the endpoint descriptor of the GDS will be taken from the GdsEndpointDescriptor property of the InstanceParameters.

In all cases described above, you will receive back a UADiscoveryElementCollection object, which is collection of UADiscoveryElement-s. Each element contains a server's discovery URL. The discovery URL is the main piece of information that you can further use if you want to connect to that OPC-UA server. Each UADiscoveryElement contains information gathered about one OPC server found by the discovery process, including things like the application name, server capabilities, etc.

C#

```
// This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
// The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class DiscoverGlobalServers
    {
        public static void Main1()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection =
client.DiscoverGlobalServers("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
```

```
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
            }
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
// The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

class procedure DiscoverGlobalServers.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of application elements
  try
    DiscoveryElements := Client.DiscoverGlobalServers('opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn;
    WriteLn('Server name: ', DiscoveryElement.ServerName);
    WriteLn('Discovery URI string: ', DiscoveryElement.DiscoveryUriString);
    WriteLn('Server capabilities: ', DiscoveryElement.ServerCapabilities.ToString);
  end;
end;
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers from the Global
```

```
Discovery Server (GDS).
' The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverGlobalServers
        Public Shared Sub Main1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection =
client.DiscoverGlobalServers("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString)
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

You can also obtain the results unflattened:

C#

```
// This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
// The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
```

```
    partial class DiscoverGlobalServers
    {
        public static void Hierarchical()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverGlobalServers(
                    "opc.tcp://opcua.demo-this.com:58810/GlobalDiscoveryServer",
                    flat:false);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI strings:");
                foreach (string discoveryUriString in
discoveryElement.DiscoveryUriStrings)
                    Console.WriteLine("  {0}", discoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString);
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
' The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverGlobalServers
        Public Shared Sub Hierarchical()
            ' Instantiate the client object
```

```
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverGlobalServers( _
                    "opc.tcp://opcua.demo-this.com:58810/GlobalDiscoveryServer",
                    flat:=False)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI strings:")
                For Each discoveryUriString As String In
discoveryElement.DiscoveryUriStrings
                    Console.WriteLine("  {0}", discoveryUriString)
                Next discoveryUriString
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString)
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

In .NET, if you want to connect to the discovered OPC server later in your code by calling other methods, use the built-in conversion of UADiscoveryElement to UAEndpointDescriptor, and pass the resulting object as an endpointDescriptor argument either directly to some other method call.

Note that QuickOPC returns UADiscoveryElementCollection from all OPC UA discovery methods, but the amount of information contained in each returned UADiscoveryElement depends on the discovery approach used. For OPC UA Global Discovery and OPC UA Network Discovery, only following primary properties of each UADiscoveryElement are filled in:

- ApplicationType
- DiscoveryUriString
- ServerCapabilities
- ServerName

> ⚠ OPC UA Global Discovery Server (GDS) is not part of OPC Data Client installation.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

5.1.2.3.5.4 Generalized OPC UA Discovery

The various kinds of OPC UA discovery can be generalized into just one method, Discover. This method is given a discovery query argument (of UADiscoveryQuery type) that tells it the requirements for the discovery. The specific kinds of OPC UA discovery described earlier are, in fact, implemented by constructing the discovery query appropriately, and then calling the Discover method.

The UADiscoveryQuery type is an abstract class which is at the root of a structured class hierarchy, with several concrete subclasses:

- UAGlobalApplicationDiscoveryQuery is for searching the applications globally in the administrative domain. With this query, you need to specify the GDS endpoint, and optionally a filter (UAQueryServerFilter), used to find servers that meet the specified criteria.
- UALocalApplicationDiscoveryQuery is used when you know the LDS endpoints, and want to perform discovery on these endpoints (usually in parallel). This is a specialized kind of local discovery.
- UALocalApplicationDiscoveryQuery is for finding applications local to a specified host.
- UANetworkServerDiscoveryQuery is for finding servers on a multicast subnet (or more connected subnets) using LDS-ME (Local Discovery Server with Multicast Extensions).

This generalized discovery mechanism allows for flexibility both on the component and developer sides, and possible future expansion with other kinds of discovery, without a need to introduce new methods into the IEasyUAClient interface.

5.1.2.3.6 Browsing for OPC UA Nodes

Principles

In This Topic

In OPC Unified Architecture, the address space is composed of nodes. The nodes are interconnected by means of references. The references do not necessarily have to be hierarchical; this means that the address space does not form a tree, but it is generally an interconnected “mesh” of nodes.

Nodes are of different node classes, and each node class has a fixed set of attributes, defined in the OPC specification. References can also be of various types, and the set of reference types is extensible.

OPC Data Client gives you methods to traverse through the address space information and obtain the information available there. It is also possible to filter the returned nodes by criteria such as the node classes of interest, or reference types to follow.

The address space may contain all kinds of information and various node classes and types of references. In this chapter, we will focus on browsing for node classes and reference types that are relevant for data access tasks.

There are following browse methods specialized for data access:

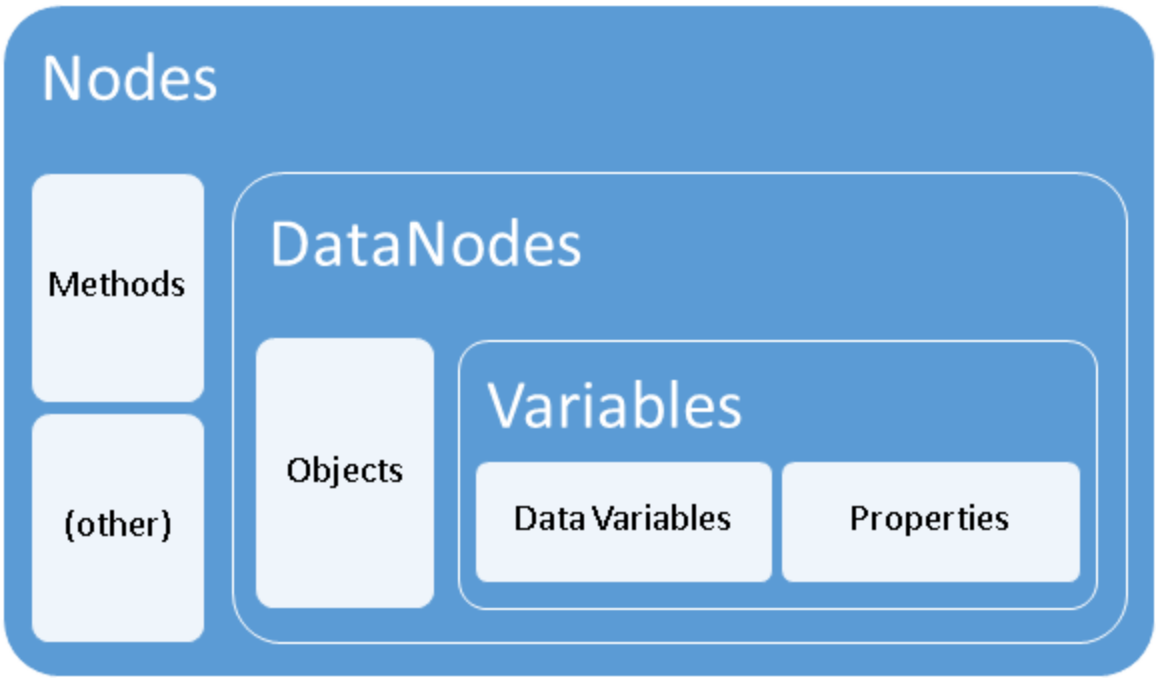
- BrowseObjects
- BrowseDataVariables
- BrowseProperties
- BrowseVariables
- BrowseDataNodes

Other browsing methods are:

- BrowseMethods
- Browse

Objects (returned by the BrowseObjects method) are equivalent of folders in a file system – they provide means of organizing other nodes logically. Data variables (returned by the BrowseDataVariables method) represent a value, and clients can read the value, write the value, or subscribe to changes of the value. Properties (returned by the BrowseProperties method) represent additional characteristics of a node which is not described by its attributes. Variables (either data variables, or properties), are returned by the BrowseVariables method. The BrowseDataNodes method returns all kinds of data nodes – and we use the term data nodes for any of objects, variables, or properties.

The following picture shows the relationship between various browsing options.



If you want to retrieve a list of all nodes that can be followed from a given node (or from an Objects folder that is a default starting point for data access browsing) of the OPC server, call one of the methods listed above. You will receive back a UANodeElementCollection, which is a collection of UANodeElement objects. Each UANodeElement contains information gathered about one node that can be followed from a given node, such as its browse name, its node ID, its display name, or a type definition. The UANodeElement is convertible to UANodeDescriptor, and you can therefore pass it to methods for further browsing from that node, or to methods like Read, ReadValue, SubscribeMonitoredItem, or SubscribeDataChange.

Note: In some cases, defined by the OPC UA specification, the elements returned in the UANodeElementCollection will have unique browse names. In general, however, OPC UA allows multiple returned elements share the same browse name.

The OPC UA address space may also contain Methods. They can be used to call (invoke) a code in the OPC UA server, passing input and output arguments to/from the method. Methods available on a specified object are returned by the BrowseMethods call.

The most generic address space browsing method is the Browse Method. It allows the widest range of filtering options by passing in an argument of type UABrowseParameters. You can specify any set of node classes, and reference type IDs, with this object.

Using the ReferenceTypeIds property in the UABrowseParameters, you can specify which references in the address space will be followed in browsing. The IncludeSubtypes flag determines whether subtypes of the specified reference type should be returned by the browsing.

Using the BrowseDirections property in the UABrowseParameters, you can specify which directions of the references the Browse method should return. The available choices are given by the UABrowseDirections enumeration, and are Forward, Inverse, and Both.

Examples

C#

```
// This example shows how to obtain "data nodes" (objects, variables and properties)
under the "Objects" node in the address
// space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseDataNodes
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain data nodes under "Objects" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseDataNodes(endpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
```

```
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: Server
        //nodeElement.NodeId: Server
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2253
        //
        //nodeElement.DisplayName: Data
        //nodeElement.NodeId: nsu = http://test.org/UA/Data/ ;ns=2;i=10157
        //nodeElement.NodeId.ExpandedText: nsu = http://test.org/UA/Data/ ;ns=2;i=10157
        //
        //nodeElement.DisplayName: Boilers
        //nodeElement.NodeId: nsu = http://opcfoundation.org/UA/Boiler/ ;ns=4;i=1240
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/Boiler/
;ns=4;i=1240
        //
        //nodeElement.DisplayName: MemoryBuffers
        //nodeElement.NodeId: nsu = http://samples.org/UA/memorybuffer ;ns=7;i=1025
        //nodeElement.NodeId.ExpandedText: nsu = http://samples.org/UA/memorybuffer
;ns=7;i=1025
    }
}
```

VB.NET

```
' This example shows how to obtain "data nodes" (objects, variables and properties)
under the "Objects" node in the address
' space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseDataNodes
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain data nodes under "Objects" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseDataNodes(endpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
```

```
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2253
            'nodeElement.DisplayName: Server
            '
            'nodeElement.NodeId: nsu=http://test.org/UA/Data/;i=10157
            'nodeElement.DisplayName: Data
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/Boiler/;i=1240
            'nodeElement.DisplayName: Boilers
            '
            'nodeElement.NodeId: nsu=http://samples.org/UA/memorybuffer;i=1025
            'nodeElement.DisplayName: MemoryBuffers
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
// For each node, it displays its browse name and node ID.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "BrowseDataNodes.h"

namespace _EasyUAClient
{
    void BrowseDataNodes::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            _UANodeElementCollectionPtr NodeElementsPtr = ClientPtr->BrowseDataNodes(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10791");

            // Display results
            IEnumVARIANTPtr EnumNodeElementPtr = NodeElementsPtr->GetEnumerator();
            _variant_t vNodeElement;
            while (EnumNodeElementPtr->Next(1, &vNodeElement, NULL) == S_OK)
            {
                _UANodeElementPtr NodeElementPtr(vNodeElement);
```

```
                _tprintf(_T("%s: "), (LPCTSTR)CW2CT(NodeElementPtr->BrowseName-
>ToString));
                _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(NodeElementPtr->NodeId->ToString));
                vNodeElement.Clear();
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to obtain all data nodes (objects and variables)
// under a given node of the OPC-UA address space. For each node, it displays
// its browse name and node ID.

class procedure BrowseDataNodes.Main;
var
  Client: EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  NodeElements := Client.BrowseDataNodes(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10791');

  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn(NodeElement.BrowseName.ToString, ': ', NodeElement.NodeId.ToString);
  end;
end;
```

Object Pascal

```
// This example shows how to obtain all data nodes (objects and variables)
// under a given node of the OPC-UA address space. For each node, it displays
// its browse name and node ID.

class procedure BrowseDataNodes.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
begin
  // Instantiate the client object
```

```
  Client := CoEasyUAClient.Create;

  try
    NodeElements := Client.BrowseDataNodes(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10791');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn(NodeElement.BrowseName.ToString, ': ', NodeElement.NodeId.ToString);
  end;
end;
```

PHP

```
// This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
// For each node, it displays its browse name and node ID.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $NodeElements = $Client->BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("%s: %s\n", $NodeElement->BrowseName, $NodeElement->NodeId);
}
```

PowerScript

```
// This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
// For each node, it displays its browse name and node ID.

mle_outputtext.Text = ""
```

```
// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Perform the operation
OLEObject nodeElements
TRY
    nodeElements = client.BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
Int i
FOR i = 0 TO nodeElements.Count - 1
    OLEObject nodeElement
    nodeElement = nodeElements.Item[i]
    mle_outputtext.Text = mle_outputtext.Text + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "nodeElement.NodeId: " +
nodeElement.NodeId.DisplayString + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "nodeElement.DisplayName: " +
nodeElement.DisplayName + "~r~n"
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to obtain all data nodes (objects and variables) under a given
node of the OPC-UA address space.
# For each node, it displays its browse name and node ID.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Perform the operation
nodeElements = client.BrowseDataNodes('http://opcua.demo-
this.com:51211/UA/SampleServer', 'nsu=http://test.org/UA/Data/;i=10791')

# Display results
for nodeElement in nodeElements:
    print(nodeElement.BrowseName, ': ', nodeElement.NodeId)
```

Visual Basic (VB 6.)

```
Rem This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
Rem For each node, it displays its browse name and node ID.

Private Sub BrowseDataNodes_Main_Command_Click()
```

```
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Dim NodeElements As UANodeElementCollection
    Set NodeElements = Client.BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    Dim NodeElement: For Each NodeElement In NodeElements
        OutputText = OutputText & NodeElement.BrowseName & ": " & NodeElement.NodeId &
vbCrLf
    Next
End Sub
```

VBScript

```
Rem This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
Rem For each node, it displays its browse name and node ID.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo NodeElement.BrowseName & ": " & NodeElement.NodeId
Next
```

C#

```
// Shows how to obtain references of all kinds to nodes of all classes, from the
"Server" node in the address space.
```

```
using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class Browse
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain nodes under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.Browse(
                    endpointDescriptor,
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerArray
        //nodeElement.NodeId: Server_ServerArray
```

```
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
        //
        //nodeElement.DisplayName: NamespaceArray
        //nodeElement.NodeId: Server_NamespaceArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
        //
        //nodeElement.DisplayName: ServerStatus
        //nodeElement.NodeId: Server_ServerStatus
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
        //
        //nodeElement.DisplayName: ServiceLevel
        //nodeElement.NodeId: Server_ServiceLevel
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
        //
        //nodeElement.DisplayName: Auditing
        //nodeElement.NodeId: Server_Auditing
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
        //
        //nodeElement.DisplayName: ServerCapabilities
        //nodeElement.NodeId: Server_ServerCapabilities
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
        //
        //nodeElement.DisplayName: ServerDiagnostics
        //nodeElement.NodeId: Server_ServerDiagnostics
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
        //
        //nodeElement.DisplayName: VendorServerInfo
        //nodeElement.NodeId: Server_VendorServerInfo
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
        //
        //nodeElement.DisplayName: ServerRedundancy
        //nodeElement.NodeId: Server_ServerRedundancy
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
        //
        //nodeElement.DisplayName: Namespaces
        //nodeElement.NodeId: Server_Namespaces
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
        //
        //nodeElement.DisplayName: GetMonitoredItems
        //nodeElement.NodeId: Server_GetMonitoredItems
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11492
        //
        //nodeElement.DisplayName: Data
        //nodeElement.NodeId: nsu=http://test.org/UA/Data/ ;ns=2;i=10157
        //nodeElement.NodeId.ExpandedText: nsu=http://test.org/UA/Data/ ;ns=2;i=10157
        //
        //nodeElement.DisplayName: Boilers
        //nodeElement.NodeId: nsu=http://opcfoundation.org/UA/Boiler/ ;ns=4;i=1240
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/Boiler/
;ns=4;i=1240
        //
        //nodeElement.DisplayName: ServerType
        //nodeElement.NodeId: ServerType
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2004
    }
}
```

VB.NET

```
' Shows how to obtain references of all kinds to nodes of all classes, from the
"Server" node in the address space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class Browse
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain nodes under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.Browse( _
                    endpointDescriptor, _
                    UAObjectIds.Server, _
                    New UABrowseParameters(UANodeClass.All, New UANodeId()
{UAReferenceTypeIds.References}))
                ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain nodes under a given node of the OPC-UA address
space.
Rem For each node, it displays its browse name and node ID.
```

```
Option Explicit

Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UANodeDescriptor")
Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
BrowsePathParser.DefaultNamespaceUriString = "http://test.org/UA/Data/"
NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar")

Dim BrowseParameters: Set BrowseParameters =
CreateObject("OpcLabs.EasyOpc.UA.UABrowseParameters")
BrowseParameters.StandardName = "AllForwardReferences"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.Browse(EndpointDescriptor, NodeDescriptor,
BrowseParameters)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo NodeElement.BrowseName & ": " & NodeElement.NodeId
Next
```

C#

```
// This example shows how to obtain data variables under the "Server" node in the
address space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseDataVariables
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
```

```
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain variables under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
UAObjectIds.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerStatus
        //nodeElement.NodeId: Server_ServerStatus
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
    }
}
```

Object Pascal

```
// This example shows how to obtain data variables under the "Server" node
// in the address space.

class procedure BrowseDataVariables.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
  ServerNodeId: _UANodeId;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
```

```
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain variables under "Server" node
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  try
    NodeElements := Client.BrowseDataVariables(EndpointDescriptor,
ServerNodeId.ExpandedText);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn;
    WriteLn('nodeElement.NodeId: ', NodeElement.NodeId.ToString);
    WriteLn('nodeElement.NodeId.ExpandedText: ', NodeElement.NodeId.ExpandedText);
    WriteLn('nodeElement.DisplayName: ', NodeElement.DisplayName);
  end;

  // Example output:
  //
  //nodeElement.NodeId: Server_ServerStatus
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
  //nodeElement.DisplayName: ServerStatus
end;
```

PHP

```
// This example shows how to obtain data variables under the "Server" node
// in the address space.

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain variables under "Server" node
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";
```

```
try
{
    $NodeElements = $Client->BrowseDataVariables($EndpointDescriptor, $ServerNodeId-
>ExpandedText);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("\n");
    printf("nodeElement.NodeId: %s\n", $NodeElement->NodeId);
    printf("nodeElement.NodeId.ExpandedText: %s\n", $NodeElement->NodeId-
>ExpandedText);
    printf("nodeElement.DisplayName: %s\n", $NodeElement->DisplayName);
}

// Example output:
//
//nodeElement.NodeId: Server_ServerStatus
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
//nodeElement.DisplayName: ServerStatus
```

VB.NET

```
' This example shows how to obtain data variables under the "Server" node in the
address space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseDataVariables
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain variables under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
UAObjectIds.Server)
```

```
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2256
            'nodeElement.DisplayName: ServerStatus
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain data variables under the "Server" node in the
address space.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain variables under "Server" node
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
On Error Resume Next
Dim NodeElementCollection: Set NodeElementCollection =
Client.BrowseDataVariables(endpointDescriptor, ServerNodeId.ExpandedText)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElementCollection
    WScript.Echo
    WScript.Echo "nodeElement.NodeId: " & NodeElement.NodeId
    WScript.Echo "nodeElement.NodeId.ExpandedText: " & NodeElement.NodeId.ExpandedText
    WScript.Echo "nodeElement.DisplayName: " & NodeElement.DisplayName
Next
```

```
' Example output:
'
'nodeElement.NodeId: Server_ServerStatus
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
'nodeElement.DisplayName: ServerStatus
```

C#

```
// This example shows how to obtain objects under the "Server" node in the address
space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseObjects
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain objects under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseObjects(endpointDescriptor,
UAObjectIds.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
```

```
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerCapabilities
        //nodeElement.NodeId: Server_ServerCapabilities
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2268
        //
        //nodeElement.DisplayName: ServerDiagnostics
        //nodeElement.NodeId: Server_ServerDiagnostics
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2274
        //
        //nodeElement.DisplayName: VendorServerInfo
        //nodeElement.NodeId: Server_VendorServerInfo
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2295
        //
        //nodeElement.DisplayName: ServerRedundancy
        //nodeElement.NodeId: Server_ServerRedundancy
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2296
        //
        //nodeElement.DisplayName: Namespaces
        //nodeElement.NodeId: Server_Namespaces
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=11715
    }
}
```

Object Pascal

```
// This example shows how to obtain objects under the "Server" node
// in the address space.

class procedure BrowseObjects.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
  ServerNodeId: _UANodeId;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain objects under "Server" node
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  try
    NodeElements := Client.BrowseObjects(EndpointDescriptor,
ServerNodeId.ExpandedText);
  except
```

```
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn;
    WriteLn('nodeElement.NodeId: ', NodeElement.NodeId.ToString);
    WriteLn('nodeElement.NodeId.ExpandedText: ', NodeElement.NodeId.ExpandedText);
    WriteLn('nodeElement.DisplayName: ', NodeElement.DisplayName);
  end;

  // Example output:
  //
  //nodeElement.NodeId: Server_ServerCapabilities
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
  //nodeElement.DisplayName: ServerCapabilities
  //
  //nodeElement.NodeId: Server_ServerDiagnostics
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
  //nodeElement.DisplayName: ServerDiagnostics
  //
  //nodeElement.NodeId: Server_VendorServerInfo
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
  //nodeElement.DisplayName: VendorServerInfo
  //
  //nodeElement.NodeId: Server_ServerRedundancy
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
  //nodeElement.DisplayName: ServerRedundancy
  //
  //nodeElement.NodeId: Server_Namespaces
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
  //nodeElement.DisplayName: Namespaces

end;
```

PHP

```
// This example shows how to obtain objects under the "Server" node
// in the address space.

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
```

```
// Obtain variables under "Server" node
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";

try
{
    $NodeElements = $Client->BrowseObjects($EndpointDescriptor, $ServerNodeId-
>ExpandedText);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("\n");
    printf("nodeElement.NodeId: %s\n", $NodeElement->NodeId);
    printf("nodeElement.NodeId.ExpandedText: %s\n", $NodeElement->NodeId-
>ExpandedText);
    printf("nodeElement.DisplayName: %s\n", $NodeElement->DisplayName);
}

// Example output:
//
//nodeElement.NodeId: Server_ServerCapabilities
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
//nodeElement.DisplayName: ServerCapabilities
//
//nodeElement.NodeId: Server_ServerDiagnostics
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
//nodeElement.DisplayName: ServerDiagnostics
//
//nodeElement.NodeId: Server_VendorServerInfo
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
//nodeElement.DisplayName: VendorServerInfo
//
//nodeElement.NodeId: Server_ServerRedundancy
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
//nodeElement.DisplayName: ServerRedundancy
//
//nodeElement.NodeId: Server_Namespaces
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
//nodeElement.DisplayName: Namespaces
```

VB.NET

```
' This example shows how to obtain objects under the "Server" node in the address
space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel
```

```
Namespace UADocExamples._EasyUAClient
    Friend Class BrowseObjects
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain objects under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseObjects(endpointDescriptor,
UAObjectIds.Server)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2268
            'nodeElement.DisplayName: ServerCapabilities
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2274
            'nodeElement.DisplayName: ServerDiagnostics
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2295
            'nodeElement.DisplayName: VendorServerInfo
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2296
            'nodeElement.DisplayName: ServerRedundancy
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain objects under the "Server" node in the address
space.

Option Explicit
```

```
Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain objects under "Server" node
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
On Error Resume Next
Dim NodeElementCollection: Set NodeElementCollection =
Client.BrowseObjects(endpointDescriptor, ServerNodeId.ExpandedText)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElementCollection
    WScript.Echo
    WScript.Echo "nodeElement.NodeId: " & NodeElement.NodeId
    WScript.Echo "nodeElement.NodeId.ExpandedText: " & NodeElement.NodeId.ExpandedText
    WScript.Echo "nodeElement.DisplayName: " & NodeElement.DisplayName
Next

' Example output:
'
'nodeElement.NodeId: Server_ServerCapabilities
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
'nodeElement.DisplayName: ServerCapabilities
'
'nodeElement.NodeId: Server_ServerDiagnostics
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
'nodeElement.DisplayName: ServerDiagnostics
'
'nodeElement.NodeId: Server_VendorServerInfo
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
'nodeElement.DisplayName: VendorServerInfo
'
'nodeElement.NodeId: Server_ServerRedundancy
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
'nodeElement.DisplayName: ServerRedundancy
'
'nodeElement.NodeId: Server_Namespaces
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
'nodeElement.DisplayName: Namespaces
```

C#

```
// This example shows how to obtain properties under the "Server" node in the address
space.
```

```
using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseProperties
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain properties under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseProperties(endpointDescriptor,
UAObjectIds.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerArray
        //nodeElement.NodeId: Server_ServerArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
        //
        //nodeElement.DisplayName: NamespaceArray
        //nodeElement.NodeId: Server_NamespaceArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
```

```
        //
        //nodeElement.DisplayName: ServiceLevel
        //nodeElement.NodeId: Server_ServiceLevel
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
        //
        //nodeElement.DisplayName: Auditing
        //nodeElement.NodeId: Server_Auditing
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
    }
}
```

Object Pascal

```
// This example shows how to obtain properties under the "Server" node
// in the address space.

class procedure BrowseProperties.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
  ServerNodeId: _UANodeId;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain properties under "Server" node
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  try
    NodeElements := Client.BrowseProperties(EndpointDescriptor,
ServerNodeId.ExpandedText);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn;
    WriteLn('nodeElement.NodeId: ', NodeElement.NodeId.ToString);
    WriteLn('nodeElement.NodeId.ExpandedText: ', NodeElement.NodeId.ExpandedText);
    WriteLn('nodeElement.DisplayName: ', NodeElement.DisplayName);
```

```
  end;

  // Example output:
  //
  //nodeElement.NodeId: Server_ServerArray
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
  //nodeElement.DisplayName: ServerArray
  //
  //nodeElement.NodeId: Server_NamespaceArray
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
  //nodeElement.DisplayName: NamespaceArray
  //
  //nodeElement.NodeId: Server_ServiceLevel
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
  //nodeElement.DisplayName: ServiceLevel
  //
  //nodeElement.NodeId: Server_Auditing
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
  //nodeElement.DisplayName: Auditing

end;
```

PHP

```
// This example shows how to obtain properties under the "Server" node
// in the address space.

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain variables under "Server" node
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";

try
{
    $NodeElements = $Client->BrowseProperties($EndpointDescriptor, $ServerNodeId-
>ExpandedText);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("\n");
    printf("nodeElement.NodeId: %s\n", $NodeElement->NodeId);
```

```
    printf("nodeElement.NodeId.ExpandedText: %s\n", $NodeElement->NodeId-
>ExpandedText);
    printf("nodeElement.DisplayName: %s\n", $NodeElement->DisplayName);
}

// Example output:
//
//nodeElement.NodeId: Server_ServerArray
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
//nodeElement.DisplayName: ServerArray
//
//nodeElement.NodeId: Server_NamespaceArray
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
//nodeElement.DisplayName: NamespaceArray
//
//nodeElement.NodeId: Server_ServiceLevel
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
//nodeElement.DisplayName: ServiceLevel
//
//nodeElement.NodeId: Server_Auditing
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
//nodeElement.DisplayName: Auditing
```

VB.NET

```
' This example shows how to obtain properties under the "Server" node in the address
space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseProperties
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain properties under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseProperties(endpointDescriptor,
UAObjectIds.Server)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
```

```
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2254
            'nodeElement.DisplayName: ServerArray
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2255
            'nodeElement.DisplayName: NamespaceArray
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2267
            'nodeElement.DisplayName: ServiceLevel
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2994
            'nodeElement.DisplayName: Auditing            }
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain properties under the "Server" node in the address
space.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain properties under "Server" node
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
On Error Resume Next
Dim NodeElementCollection: Set NodeElementCollection =
Client.BrowseProperties(endpointDescriptor, ServerNodeId.ExpandedText)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElementCollection
```

```
    WScript.Echo
    WScript.Echo "nodeElement.NodeId: " & NodeElement.NodeId
    WScript.Echo "nodeElement.NodeId.ExpandedText: " & NodeElement.NodeId.ExpandedText
    WScript.Echo "nodeElement.DisplayName: " & NodeElement.DisplayName
Next

' Example output:
'
'nodeElement.NodeId: Server_ServerArray
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
'nodeElement.DisplayName: ServerArray
'
'nodeElement.NodeId: Server_NamespaceArray
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
'nodeElement.DisplayName: NamespaceArray
'
'nodeElement.NodeId: Server_ServiceLevel
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
'nodeElement.DisplayName: ServiceLevel
'
'nodeElement.NodeId: Server_Auditing
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
'nodeElement.DisplayName: Auditing
```

C#

```
// This example shows how to obtain all method nodes under a given node of the OPC-UA
address space.
// For each node, it displays its browse name and node ID.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseMethods
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain methods under the specified node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseMethods(endpointDescriptor,
"nsu=http://test.org/UA/Data/ ;i=10755");
```

```
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
                Console.WriteLine($"{nodeElement.BrowseName}: {nodeElement.NodeId}");
        }

        // Example output:
        //ScalarMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10756
        //ScalarMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10759
        //ScalarMethod3: nsu = http://test.org/UA/Data/ ;ns=2;i=10762
        //ArrayMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10765
        //ArrayMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10768
        //ArrayMethod3: nsu = http://test.org/UA/Data/ ;ns=2;i=10771
        //UserScalarMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10774
        //UserScalarMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10777
        //UserArrayMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10780
        //UserArrayMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10783
    }
}
```

PHP

```
// This example shows how to obtain all method nodes under a given node of the OPC-UA
address space.
// For each node, it displays its browse name and node ID.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $NodeElements = $Client->BrowseMethods("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10755");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}


// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("%s: %s\n", $NodeElement->BrowseName, $NodeElement->NodeId);
}
```

VBScript

```
Rem This example shows how to obtain all method nodes under a given node of the OPC-UA
address space.
Rem For each node, it displays its browse name and node ID.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseMethods("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/ ;i=10755")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo NodeElement.BrowseName & ": " & NodeElement.NodeId
Next


' Example output:
'ScalarMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10756
'ScalarMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10759
'ScalarMethod3: nsu=http://test.org/UA/Data/ ;ns=2;i=10762
'ArrayMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10765
'ArrayMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10768
'ArrayMethod3: nsu=http://test.org/UA/Data/ ;ns=2;i=10771
'UserScalarMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10774
'UserScalarMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10777
'UserArrayMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10780
'UserArrayMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10783
```

5.1.2.4 Subscribing to Information (OPC Data)

If your application needs to monitor changes of certain process value (OPC Classic item, OPC UA monitored item), it can subscribe to it, and receive notifications when the value changes. For performance reasons, this approach is preferred over repeatedly reading the item's value (polling). Note that OPC Data Client has internal optimizations which greatly reduce the negative effects of polling, however subscription is still preferred.

OPC Data Client contains methods that allow you to subscribe to OPC Classic items or OPC UA monitored items, change the subscription parameters, and unsubscribe.

5.1.2.4.1 Subscribing to OPC Classic Items

Subscription is initiated by calling either SubscribeItem or SubscribeMultipleItems method. For any change in the subscribed item's value, your application will receive the ItemChanged event notification, described further below. Obviously, you first need to hook up event handler for that event, and in order to prevent event loss, you should do it before subscribing. Alternatively, you can pass a callback method into the SubscribeItem or SubscribeMultipleItems call.

Values of some items may be changing quite frequently, and receiving all changes that are generated is not desirable for performance reasons; there are also physical limitations to the event throughput in the system. Your application needs to specify the requested update rate, which effectively tells the OPC server that you do not need to receive event notifications any faster than that. For OPC items that support it, you can optionally specify a percent deadband; only changes that exceed the deadband will generate an event notification.

In OPC Data Client.NET, the requested update rate, percent deadband, and data type are all contained in a DAGroupParameters object.

A single item

If you want to subscribe to a specific OPC item, call the SubscribeItem method. You can pass in individual arguments for machine name, server class, ItemID, data type, requested update rate, and an optional percent deadband. Usually, you also pass in a State argument of type Object (in OPC Data Client.NET) or VARIANT (in OPC Data Client-COM). When the item's value changes, the State argument is then passed to the ItemChanged event handler in the EasyDAItemChangedEventArgs.Arguments.State property. The SubscribeItem method returns a subscription handle that you can later use to change the subscription parameters, or unsubscribe.

C#

```
// Hooking up events and receiving OPC item changes.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeItem
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                var eventHandler = new
EasyDAItemChangedEventHandler(client_ItemChanged);
                client.ItemChanged += eventHandler;

                Console.WriteLine("Subscribing item...");
                client.SubscribeItem("", "OPCLabs.KitServer.2", "Demo.Ramp", 200);
                Thread.Sleep(30 * 1000);
                client.UnsubscribeAllItems();
                client.ItemChanged -= eventHandler;
            }
        }
```

```
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' Hooking up events and receiving OPC item changes.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeItem
        Shared Sub Main1()
            Using client = New EasyDAClient()
                Dim eventHandler = New EasyDAItemChangedEventHandler(AddressOf
client_ItemChanged)
                AddHandler client.ItemChanged, eventHandler

                Console.WriteLine("Subscribing item...")
                client.SubscribeItem("", "OPCLabs.KitServer.2", "Demo.Ramp", 200)
                Thread.Sleep(30 * 1000)
                client.UnsubscribeAllItems()
                RemoveHandler client.ItemChanged, eventHandler
            End Using
        End Sub

        Private Shared Sub client_ItemChanged(sender As Object, e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a single item and display the value
of the item with each change.

type
  TSubscribeItem_ClientEventHandlers = class
    // Item changed event handler
    procedure OnItemChanged(
      ASender: TObject;
      sender: OleVariant;
```

```
      const eventArgs: _EasyDAItemChangedEventArgs);
  end;

procedure TSubscribeItem_ClientEventHandlers.OnItemChanged(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyDAItemChangedEventArgs);
begin
  if eventArgs.Succeeded then
    WriteLn(eventArgs.Vtq.ToString)
  else
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
end;

class procedure SubscribeItem.Main;
var
  Client: TEasyDAClient;
  ClientEventHandlers: TSubscribeItem_ClientEventHandlers;
begin
  // Instantiate the client object and hook events
  Client := TEasyDAClient.Create(nil);
  ClientEventHandlers := TSubscribeItem_ClientEventHandlers.Create;
  Client.OnItemChanged := ClientEventHandlers.OnItemChanged;

  Client.SubscribeItem('', 'OPCLabs.KitServer.2', 'Simulation.Random', 1000);

  WriteLn('Processing item changed events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a single item and display the value
of the item with each change.
//
// Some related documentation: http://php.net/manual/en/function.com-event-sink.php .
Pay attention to the comment that says
// "Be careful how you use this feature; if you are doing something similar to the
example below, then it doesn't really make
// sense to run it in a web server context.". What they are trying to say is that
processing a web request should be
// a short-lived code, which does not fit well with the idea of being subscribed to
events and received them over longer time.
// It is possible to write such code, but it is only useful when processing the request
is allowed to take relatively long. Or,
// when you are using PHP from command-line, or otherwise - not to serve a web page
directly.
```

```
//
// Subscribing to QuickOPC-COM events in the context of PHP Web application, while not
imposing the limitations to the request
// processing time, has to be "worked around", e.g. using the "event pull" mechanism.

class DEasyDAClientEvents {
    function ItemChanged($varSender, $varE)
    {
        if ($varE->Succeeded)
        {
        print $varE->Vtq->ToString();
            print "\n";
        }
        else
        {
            printf("*** Failure: %s\n", $varE->ErrorMessageBrief);
        }
    }
}

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$Events = new DEasyDAClientEvents();
com_event_sink($Client, $Events, "DEasyDAClientEvents");

$Client->SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

print "Processing item changed events for 1 minute...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

VBScript

```
Rem This example shows how to subscribe to changes of a single item and display the
value of the item with each change.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

Client.SubscribeItem "", "OPCLabs.KitServer.2", "Simulation.Random", 1000

WScript.Echo "Processing item changed events for 1 minute..."
WScript.Sleep 60*1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq.ToString
End Sub
```

VBScript

```
Rem This example subscribes to changes of 2 items separately, and displays rich
information available with each item changed
Rem event notification.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

Client.SubscribeItem "", "OPCLabs.KitServer.2", "Simulation.Random", 5*1000
Client.SubscribeItem "", "OPCLabs.KitServer.2", "Trends.Ramp (1 min)", 5*1000

WScript.Echo "Processing item changed events for 1 minute..."
WScript.Sleep 60*1000



Sub Client_ItemChanged(Sender, e)
    On Error Resume Next
    WScript.Echo
    WScript.Echo "e.Arguments.State: " & e.Arguments.State
    WScript.Echo "e.Arguments.ServerDescriptor.MachineName: " &
e.Arguments.ServerDescriptor.MachineName
    WScript.Echo "e.Arguments.ServerDescriptor.ServerClass: " &
e.Arguments.ServerDescriptor.ServerClass
    WScript.Echo "e.Arguments.ItemDescriptor.ItemId: " &
e.Arguments.ItemDescriptor.ItemId
    WScript.Echo "e.Arguments.ItemDescriptor.AccessPath: " &
e.Arguments.ItemDescriptor.AccessPath
    WScript.Echo "e.Arguments.ItemDescriptor.RequestedDataType: " &
e.Arguments.ItemDescriptor.RequestedDataType
    WScript.Echo "e.Arguments.GroupParameters.Locale: " &
e.Arguments.GroupParameters.Locale
    WScript.Echo "e.Arguments.GroupParameters.RequestedUpdateRate: " &
e.Arguments.GroupParameters.RequestedUpdateRate
    WScript.Echo "e.Arguments.GroupParameters.PercentDeadband: " &
e.Arguments.GroupParameters.PercentDeadband
    WScript.Echo "e.Exception.Message: " & e.Exception.Message
    WScript.Echo "e.Exception.Source: " & e.Exception.Source
    WScript.Echo "e.Exception.ErrorCode: " & e.Exception.ErrorCode
    WScript.Echo "e.Vtq.Value: " & e.Vtq.Value
    WScript.Echo "e.Vtq.Timestamp: " & e.Vtq.Timestamp
    WScript.Echo "e.Vtq.TimestampLocal: " & e.Vtq.TimestampLocal
    WScript.Echo "e.Vtq.Quality: " & e.Vtq.Quality
End Sub
```

In OPC Data Client.NET, you can also pass in a combination of ServerDescriptor, DAItemDescriptor and DAGroupParameters objects, in place of individual arguments.

The State argument is typically used to provide some sort of correlation between objects in your application, and the event notifications. For example, if you are programming an HMI application and you want the event handler to update the control that displays the item's value, you may want to set the State argument to the control object itself. When the event notification arrives, you simply update the control indicated by the State property of EasyDAItemChangedEventArgs, without having to look it up by ItemId or so.

Multiple items

To subscribe to multiple items simultaneously in an efficient manner, call the SubscribeMultipleItems method (instead of multiple SubscribeItem calls in a loop). You receive back an array of integers, which are the subscription handles.

You pass in an array of DAItemGroupArguments objects (each containing information for a single subscription to be made), to the SubscribeMultipleItems method.

C#

```
// This example shows how subscribe to changes of multiple items and display the value
of the item with each change.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeMultipleItems
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_Main1_ItemChanged;

                client.SubscribeMultipleItems(
                    new[] {
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Sine (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, null)
                        });

                Console.WriteLine("Processing item changed events for 1 minute...");
                Thread.Sleep(60 * 1000);
            }
        }

        // Item changed event handler
        static void client_Main1_ItemChanged(object sender, EasyDAItemChangedEventArgs
e)
        {
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq);
            else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief);
        }
    }
```

```
}
```

VB.NET

```
' This example shows how subscribe to changes of multiple items and display the value of
the item with each change.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeMultipleItems
        Public Shared Sub Main1()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged_Main1

                client.SubscribeMultipleItems(New DAItemGroupArguments() { _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, Nothing), _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Ramp (1
min)", 1000, Nothing), _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Sine (1
min)", 1000, Nothing), _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, Nothing) _
                })

                Console.WriteLine("Processing item changed events for 1 minute...")
                Thread.Sleep(60 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged_Main1(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs)
            ' Display the data
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq)
            Else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to subscribe to changes of multiple items and display the
value of the item with each change.

class DEasyDAClientEvents {
    function ItemChanged($varSender, $varE)
    {
        if ($varE->Succeeded)
        {
```

```
            printf("%s: %s\n", $varE->Arguments->ItemDescriptor->ItemId, $varE->Vtq-
>ToString());
        }
        else
        {
            printf("*** Failure: %s\n", $varE->ErrorMessageBrief);
        }
    }
}

$ItemSubscriptionArguments1 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments1->ItemDescriptor->ItemID = "Simulation.Random";
$ItemSubscriptionArguments1->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments2 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments2->ItemDescriptor->ItemID = "Trends.Ramp (1 min)";
$ItemSubscriptionArguments2->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments3 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments3->ItemDescriptor->ItemID = "Trends.Sine (1 min)";
$ItemSubscriptionArguments3->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments4 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments4->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemSubscriptionArguments4->GroupParameters->RequestedUpdateRate = 1000;

$arguments[0] = $ItemSubscriptionArguments1;
$arguments[1] = $ItemSubscriptionArguments2;
$arguments[2] = $ItemSubscriptionArguments3;
$arguments[3] = $ItemSubscriptionArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$Events = new DEasyDAClientEvents();
com_event_sink($Client, $Events, "DEasyDAClientEvents");

$Client->SubscribeMultipleItems($arguments);

print "Processing item changed events for 1 minute...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

Object Pascal

```
// This example shows how to subscribe to changes of multiple items and display the
value of the item with each change.

type
  TSubscribeMultipleItems_ClientEventHandlers = class
    // Item changed event handler
    procedure OnItemChanged(
```

```
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyDAItemChangedEventArgs);
  end;

procedure TSubscribeMultipleItems_ClientEventHandlers.OnItemChanged(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyDAItemChangedEventArgs);
begin
  if eventArgs.Succeeded then
    WriteLn(eventArgs.Arguments.ItemDescriptor.ItemId, ': ', eventArgs.Vtq.ToString)
  else
    WriteLn(eventArgs.Arguments.ItemDescriptor.ItemId, ' *** Failure: ',
eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyDAClient;
  ClientEventHandlers: TSubscribeMultipleItems_ClientEventHandlers;
  HandleArray: OleVariant;
  ItemSubscriptionArguments1: _EasyDAItemSubscriptionArguments;
  ItemSubscriptionArguments2: _EasyDAItemSubscriptionArguments;
  ItemSubscriptionArguments3: _EasyDAItemSubscriptionArguments;
  ItemSubscriptionArguments4: _EasyDAItemSubscriptionArguments;
begin
  ItemSubscriptionArguments1 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments1.ItemDescriptor.ItemID := 'Simulation.Random';
  ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate := 1000;

  ItemSubscriptionArguments2 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments2.ItemDescriptor.ItemID := 'Trends.Ramp (1 min)';
  ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate := 1000;

  ItemSubscriptionArguments3 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments3.ItemDescriptor.ItemID := 'Trends.Sine (1 min)';
  ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate := 1000;

  ItemSubscriptionArguments4 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments4.ItemDescriptor.ItemID := 'Simulation.Register_I4';
  ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate := 1000;

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ItemSubscriptionArguments1;
  Arguments[1] := ItemSubscriptionArguments2;
  Arguments[2] := ItemSubscriptionArguments3;
  Arguments[3] := ItemSubscriptionArguments4;

  // Instantiate the client object and hook events
  Client := TEasyDAClient.Create(nil);
  ClientEventHandlers := TSubscribeMultipleItems_ClientEventHandlers.Create;
```

```
  Client.OnItemChanged := ClientEventHandlers.OnItemChanged;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleItems(Arguments));

  WriteLn('Processing item changed events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VBScript

```
Rem This example shows how to subscribe to changes of multiple items and display the
value of the item with each change.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
```

```
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

Client.SubscribeMultipleItems arguments

WScript.Echo "Processing item changed events for 1 minute..."
WScript.Sleep 60*1000

Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Arguments.ItemDescriptor.ItemId & ": " & e.Vtq
End Sub
```

Common considerations

Note: It is NOT an error to subscribe to the same item twice (or more times), even with precisely the same parameters. You will receive separate subscription handles, and with regard to your application, this situation will look no different from subscribing to different items. Internally, however, the subscription made to the OPC server will be optimized (merged together) if possible.

5.1.2.4.2 Subscribing to OPC UA Monitored Items

Subscription is initiated by calling one of the SubscribeMonitoredItem, SubscribeDataChange, or SubscribeMultipleMonitoredItems method. For any significant change in the subscribed item, your application will receive the DataChangeNotification event notification, described further below. Obviously, you first need to hook up event handler for that event, and in order to prevent event loss, you should do it before subscribing. Alternatively, you can pass a callback method into the SubscribeMonitoredItem, SubscribeDataChange or SubscribeMultipleMonitoredItems call.

In This Topic

Some items may be changing quite frequently, and receiving all changes that are generated is not desirable for performance reasons; there are also physical limitations to the event throughput in the system. You may also want to influence the amount of received information, or control how and how often the information is delivered. OPC Data Client-UA has two sets of parameters for this: subscription parameters and monitoring parameters.

Subscription parameters (can be fully contained in UASubscriptionParameters object) influence the information delivery. Main subscription parameters are the relative priority, and publishing interval (in milliseconds). The publishing interval defines the cyclic rate that the subscription is requested to return notifications to the client.

Monitoring parameters (can be fully contained in UAMonitoringParameters object) influence which information is delivered to the client. Main monitoring parameters are the data change filter (optional), size of monitored item queue, and sampling interval (in milliseconds; it is the fastest rate at which the monitored items should be accessed and evaluated).

The data change filter, if present, is contained in the UADataChangeFilter object, and defines the conditions under which a data change notification should be reported and, optionally, a deadband for value changes where no notification is generated. The Trigger property selects whether a notification is generated only when the status code associated with the value changes, or either the status code or the value change, or if any of the status code, value, or a source timestamp change. The DeadbandType property determines the type of deadband used, and can either select no deadband, absolute deadband, or percent deadband.

There are implicit conversions available from System.Double and UADataChangeTrigger to UADataChangeFilter. This allows the developer to simply use the absolute deadband, or the trigger selection, in place of any data change filter.

In order to make the settings of various parameters easier, OPC Data Client-UA allows you to omit the specification of publishing interval in subscription parameters by leaving it at zero (the default). In this case, the component will calculate the publishing interval by dividing the sampling interval by the so-called automatic publishing factor (EasyUAClient.SharedParameters.EngineParameters.AutomaticPublishingFactor) with a default pre-set by the component. The resulting value is limited by the value of EasyUAClient.SharedParameters.EngineParameters.FastestAutomaticPublishingInterval property. This algorithm should determine a suitable publishing interval in the most common cases.

A single node and attribute

If you want to subscribe to a specific value in OPC address space, call the SubscribeDataChange or SubscribeMonitoredItem method. These methods have number of overloads, but in its simplest form, only three arguments are necessary: an endpoint descriptor, node ID, and a sampling interval. Other callbacks allow you to specify e.g. the callback method, a user-defined state object, absolute deadband value, or pass in a full set of parameters using EasyUAMonitoredItemArguments object. The SubscribeDataChange and SubscribeMonitoredItem methods return a subscription handle that you can later use to change the subscription parameters, or unsubscribe.

It is common to pass in a State argument of type Object. When the monitored item's value changes, the State argument is then passed to the DataChangeNotification event handler (or your callback method) in the EasyUADataChangeNotificationEventArgs.Arguments object.

The State argument is typically used to provide some sort of correlation between objects in your application, and the event notifications. For example, if you are programming an HMI application and you want the event handler to update the control that displays the item's value, you may want to set the State argument to the control object itself. When the event notification arrives, you simply update the control indicated by the State property of EasyUADataChangeNotificationEventArgs.Arguments, without having to look it up by Node Id or so.

C#

```
// This example shows how to subscribe to changes of a single monitored item and
display the value of the item with each change.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
```

```
    partial class SubscribeDataChange
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000);

            Console.WriteLine("Processing data change events for 20 seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("Value: {0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item and display
the value of the item with each change.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class SubscribeDataChange
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
```

```
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to subscribe to changes of a single monitored item and
display each change.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include "SubscribeDataChange.h"

namespace _EasyUAClient
{
    // CEasyUAClientEvents

    class CEasyUAClientEvents : public IDispEventImpl<1, CEasyUAClientEvents>
    {
    public:
    BEGIN_SINK_MAP(CEasyUAClientEvents)
        // Event handlers must have the __stdcall calling convention
        SINK_ENTRY(1, DISPID_EASYUACLIENTEVENTS_DATACHANGENOTIFICATION,
&CEasyUAClientEvents::DataChangeNotification)
    END_SINK_MAP()

    public:
```

```
        // The handler for EasyUAClient.DataChangeNotification event
        STDMETHOD(DataChangeNotification)(VARIANT varSender,
_EasyUADataChangeNotificationEventArgs* pEventArgs)
        {
            // Display the data
            // Remark: Production code would check EventArgsPtr->Exception before
accessing EventArgsPtr->AttributeData.
            _UAAttributeDataPtr AttributeDataPtr(pEventArgs->AttributeData);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(AttributeDataPtr->ToString));

            return S_OK;
        }
    };



    void SubscribeDataChange::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Hook events
            CEasyUAClientEvents* pClientEvents = new CEasyUAClientEvents();
            AtlGetObjectSourceInterface(ClientPtr, &pClientEvents->m_libid,
&pClientEvents->m_iid,
                &pClientEvents->m_wMajorVerNum, &pClientEvents->m_wMinorVerNum);
            pClientEvents->m_iid = _uuidof(DEasyUAClientEvents);
            pClientEvents->DispEventAdvise(ClientPtr, &pClientEvents->m_iid);

            //
            _tprintf(_T("Subscribing...\n"));
            ClientPtr->SubscribeDataChange(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10853",
                1000);

            _tprintf(_T("Processing monitored item changed events for 1 minute...\n"));
            Sleep(60*1000);

            // Unhook events
            pClientEvents->DispEventUnadvise(ClientPtr, &pClientEvents->m_iid);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change.

type
  TClientEventHandlers121 = class
```

```
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers121.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeDataChange.Main;
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers121;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers121.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing data change events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a single monitored item and
display each change.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
```

```
    {
        // Display the data
        if ($E->Succeeded)
        printf("%s\n", $E->AttributeData);
        else
        printf("*** Failure : %s\n", $E->ErrorMessageBrief);
    }
}



// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeDataChange(
   "http://opcua.demo-this.com:51211/UA/SampleServer",
   "nsu=http://test.org/UA/Data/;i=10853",
    1000);

printf("Processing monitored item changed events for 1 minute...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of a single monitored item and
display each change.

' The client object, with events
'Public WithEvents Client1 As EasyUAClient

Private Sub SubscribeDataChange_Main_Command_Click()
    OutputText = ""

    Set Client1 = New EasyUAClient

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Call Client1.SubscribeDataChange("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10853", 1000)

    OutputText = OutputText & "Processing data changed notification events for 1
minute..." & vbCrLf
    Pause 60000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Client1.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client1 = Nothing
End Sub

Private Sub Client1_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
```

```
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to changes of a single monitored item and
display each change.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Client.SubscribeDataChange "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000

WScript.Echo "Processing monitored item changed events for 1 minute..."
WScript.Sleep 60*1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

Multiple nodes or attributes

To subscribe to multiple items simultaneously in an efficient manner, call the SubscribeMultipleMonitoredItems method (instead of multiple SubscribeDataChange or SubscribeMonitoredItem calls in a loop). You receive back an array of subscription handles. You pass in an array of EasyUAMonitoredItemArguments objects (each containing information for a single subscription to be made), to the SubscribeMultipleMonitoredItems method.

C#

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
' each change.
```

```
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
             )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include <atlsafe.h>
#include "SubscribeMultipleMonitoredItems.h"

namespace _EasyUAClient
{
    // CEasyUAClientEvents

    class CEasyUAClientEvents : public IDispEventImpl<1, CEasyUAClientEvents>
    {
    public:
    BEGIN_SINK_MAP(CEasyUAClientEvents)
        // Event handlers must have the __stdcall calling convention
        SINK_ENTRY(1, DISPID_EASYUACLIENTEVENTS_DATACHANGENOTIFICATION,
&CEasyUAClientEvents::DataChangeNotification)
    END_SINK_MAP()

    public:
        // The handler for EasyUAClient.DataChangeNotification event
        STDMETHOD(DataChangeNotification)(VARIANT varSender,
_EasyUADataChangeNotificationEventArgs* pEventArgs)
        {
            // Display the data
            _tprintf(_T("%s: "), (LPCTSTR)CW2CT(pEventArgs->Arguments->NodeDescriptor-
>ToString));
            // Remark: Production code would check EventArgsPtr->Exception before
accessing EventArgsPtr->AttributeData.
            _UAAttributeDataPtr AttributeDataPtr(pEventArgs->AttributeData);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(AttributeDataPtr->ToString));

            return S_OK;
        }
    };



    void SubscribeMultipleMonitoredItems::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Hook events
            CEasyUAClientEvents* pClientEvents = new CEasyUAClientEvents();
            AtlGetObjectSourceInterface(ClientPtr, &pClientEvents->m_libid,
&pClientEvents->m_iid,
                &pClientEvents->m_wMajorVerNum, &pClientEvents->m_wMinorVerNum);
```

```
            pClientEvents->m_iid = _uuidof(DEasyUAClientEvents);
            pClientEvents->DispEventAdvise(ClientPtr, &pClientEvents->m_iid);

            //
            _tprintf(_T("Subscribing...\n"));

            _UAMonitoringParametersPtr
MonitoringParametersPtr(_uuidof(UAMonitoringParameters));
            MonitoringParametersPtr->SamplingInterval = 1000;

            _UAMonitoredItemArgumentsPtr
MonitoringArguments1Ptr(_uuidof(EasyUAMonitoredItemArguments));
            MonitoringArguments1Ptr->EndpointDescriptor->UrlString =
L"http://opcua.demo-this.com:51211/UA/SampleServer";
            MonitoringArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10845";
            MonitoringArguments1Ptr->MonitoringParameters = MonitoringParametersPtr;

            _UAMonitoredItemArgumentsPtr
MonitoringArguments2Ptr(_uuidof(EasyUAMonitoredItemArguments));
            MonitoringArguments2Ptr->EndpointDescriptor->UrlString =
L"http://opcua.demo-this.com:51211/UA/SampleServer";
            MonitoringArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10853";
            MonitoringArguments2Ptr->MonitoringParameters = MonitoringParametersPtr;

            _UAMonitoredItemArgumentsPtr
MonitoringArguments3Ptr(_uuidof(EasyUAMonitoredItemArguments));
            MonitoringArguments3Ptr->EndpointDescriptor->UrlString =
L"http://opcua.demo-this.com:51211/UA/SampleServer";
            MonitoringArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10855";
            MonitoringArguments3Ptr->MonitoringParameters = MonitoringParametersPtr;

            CComSafeArray<VARIANT> arguments(3);
            arguments.SetAt(0, _variant_t((IDispatch*)MonitoringArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)MonitoringArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)MonitoringArguments3Ptr));
            CComVariant vArguments(arguments);

            CComSafeArray<VARIANT> handles;
            handles.Attach(ClientPtr->SubscribeMultipleMonitoredItems(&vArguments));

            for (int i = handles.GetLowerBound(); i <= handles.GetUpperBound(); i++)
            {
                _variant_t vString;
                vString.ChangeType(VT_BSTR, &_variant_t(handles[i]));
                _tprintf(_T("handleArray(%d): %s\n"), i,
(LPCTSTR)CW2CT((_bstr_t)vString));
            }

            _tprintf(_T("Processing monitored item changed events for 10
seconds...\n"));
            Sleep(10*1000);

            _tprintf(_T("Unsubscribing...\n"));
            ClientPtr->UnsubscribeAllMonitoredItems();
```

```
            _tprintf(_T("Waiting for 5 seconds...\n"));
            Sleep(5*1000);

            // Unhook events
            pClientEvents->DispEventUnadvise(ClientPtr, &pClientEvents->m_iid);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

type
  TClientEventHandlers123 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers123.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleMonitoredItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers123;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers123.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;
```

```
  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

class ClientEvents {
```

```
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}



// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;
```

```
printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
Rem each change.

' The client object, with events
'Public WithEvents Client2 As EasyUAClient

Private Sub SubscribeMultipleMonitoredItems_Main_Command_Click()
    OutputText = ""

    Set Client2 = New EasyUAClient

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Dim MonitoringParameters As New UAMonitoringParameters
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments1 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"
    Set MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments1.SetState ("Item1")

    Dim MonitoredItemArguments2 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"
    Set MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments2.SetState ("Item2")

    Dim MonitoredItemArguments3 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10855"
    Set MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments3.SetState ("Item3")

    Dim arguments(2) As Variant
    Set arguments(0) = MonitoredItemArguments1
    Set arguments(1) = MonitoredItemArguments2
    Set arguments(2) = MonitoredItemArguments3
    Dim handleArray As Variant
    handleArray = Client2.SubscribeMultipleMonitoredItems(arguments)

    Dim i As Long: For i = LBound(handleArray) To UBound(handleArray)
        OutputText = OutputText & "handleArray(" & i & "): " & handleArray(i) & vbCrLf
    Next
```

```
    OutputText = OutputText & "Processing monitored item changed events for 10
seconds..." & vbCrLf
    Pause 10000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Call Client2.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client2 = Nothing
    OutputText = OutputText & "Done." & vbCrLf
End Sub

Private Sub Client2_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
Rem each change.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
MonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
```

```
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

C#

```
// This example shows how to subscribe to changes of all data variables under a
specified object in OPC UA address space.

using System;
using System.Linq;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void AllInObject()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
```

```
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification_AllInObject;

            // Obtain variables under "Scalar" node
            Console.WriteLine("Browsing...");
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
"nsu=http://test.org/UA/Data/;ns=2;i=10787");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Create array with monitored item arguments
            EasyUAMonitoredItemArguments[] monitoredItemArgumentsArray =
nodeElementCollection
                .Select(element => new EasyUAMonitoredItemArguments(null,
endpointDescriptor, element))
                .ToArray();

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(monitoredItemArgumentsArray);

            Console.WriteLine("Processing monitored item changed events for 20
seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification_AllInObject(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of all data variables under a
specified object in OPC UA address space.

Imports System.Linq
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub AllInObject()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification_AllInObject

            ' Obtain variables under "Scalar" node
            Console.WriteLine("Subscribing...")
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
"nsu=http://test.org/UA/Data/;ns=2;i=10787")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Create array with monitored item arguments

            Dim monitoredItemArgumentsArray() As EasyUAMonitoredItemArguments =
nodeElementCollection.Select(Function(element) New
EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, element)).ToArray()

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(monitoredItemArgumentsArray)

            Console.WriteLine("Processing monitored item changed events for 20
seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
```

```
        End Sub

        Private Shared Sub client_DataChangeNotification_AllInObject(ByVal sender As
Object, ByVal e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Common considerations

If you are only interested in certain kinds of changes in the monitored item, or need to reduce the amount of incoming notifications by specifying a deadband, use **OPC UA Data Change Filter (Section 5.1.2.4.2.1)**.

Note: It is NOT an error to subscribe to the same item twice (or more times), even with precisely same parameters. You will receive separate subscription handles, and with regard to your application, this situation will look no different from subscribing to different items. Internally, however, the subscription made to the OPC server might be optimized (merged together), depending on other parameters and circumstances.

5.1.2.4.2.1 OPC UA Data Change Filter

OPC UA Data Change Filter is used when you are only interested in certain kinds of changes in the monitored item, or need to reduce the amount of incoming notifications by specifying a deadband. The Data Change Filter defines the conditions under which a data change notification should be reported and, optionally, a range or band for value changes where no data change notification is generated.

The data change filter, in its entirety, is represented by the UADataChangeFilter Class, and is located in the DataChangeFilter Property of the UAMonitoringParameters Class. There are also useful overloads of various IEasyUAClient Interface extension methods, and implicit conversions, that allow you to shorten the code, if you do not want to create an instance of the data change filter class explicitly.

One of typical use cases for the data change filter is when the client application is only interested in receiving new data when the status code or the value has changed, but not when new data has been collected with the same value (and the new data only differ in the timestamp or timestamps). In such case, a data change filter with the trigger set to UADataChangeTrigger.StatusValue is used.

Following examples show this, for a single node and for multiple nodes.

C#

```
// This example shows how to subscribe to changes of a monitored item with data change
filter.
```

```
using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeDataChange
    {
        public static void Filter()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification_Filter;

            Console.WriteLine("Subscribing...");
            // Report a notification if either the StatusCode or the value change.
            // The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
and can thus be used in its place.
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000,
                UADataChangeTrigger.StatusValue);

            Console.WriteLine("Processing data change events for 20 seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification_Filter(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("Value: {0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a monitored item with data change
filter.

Imports OpcLabs.EasyOpc.UA
```

```
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeDataChange
        Public Shared Sub Filter()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification_Filter

            Console.WriteLine("Subscribing...")
            ' Report a notification if either the StatusCode Or the value change.
            ' The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
And can thus be used in its place.
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000, UADataChangeTrigger.StatusValue)

            Console.WriteLine("Processing data change events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification_Filter(ByVal sender As Object,
ByVal e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a monitored item
// with data change filter.

type
  TClientEventHandlers120 = class
    procedure Client_DataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
```

```
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers120.Client_DataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
  begin
      WriteLn(eventArgs.AttributeData.ToString);
  end
  else
      WriteLn(' *** Failure: ', eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeDataChange.Filter;
const
  UADataChangeFilter_StatusValue = 1;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers120;
  DataChangeFilter: _UADataChangeFilter;
  EndpointDescriptor: string;
  MonitoringItemArguments1: _EasyUAMonitoredItemArguments;
  MonitoringParameters:  _UAMonitoringParameters;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers120.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.Client_DataChangeNotification;

  // Prepare the arguments.
  // Report a notification if either the StatusCode or the value change.
  DataChangeFilter := CoUADataChangeFilter.Create;
  DataChangeFilter.Trigger := UADataChangeFilter_StatusValue;
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.DataChangeFilter := DataChangeFilter;
  MonitoringParameters.SamplingInterval := 100;
  MonitoringItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoringItemArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoringItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoringItemArguments1.MonitoringParameters := MonitoringParameters;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoringItemArguments1;

  WriteLn('Subscribing...');
  Client.SubscribeMultipleMonitoredItems(arguments);
```

```
  WriteLn('Processing monitored item changed events for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a monitored item
// with data change filter.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s\n", $E->AttributeData);
        else
            printf(" *** Failure: %s\n", $E->ErrorMessageBrief);
    }
}

const UADataChangeFilter_StatusValue = 1;

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";


// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

// Prepare the arguments.
// Report a notification if either the StatusCode or the value change.
$DataChangeFilter = new COM("OpcLabs.EasyOpc.UA.UADataChangeFilter");
$DataChangeFilter->Trigger = UADataChangeFilter_StatusValue;
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->DataChangeFilter = $DataChangeFilter;
$MonitoringParameters->SamplingInterval = 100;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
```

```
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Processing monitored item changed events for 20 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 20);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of a monitored item with data change
filter.

' The client object, with events
'Public WithEvents Client3 As EasyUAClient

Private Sub SubscribeDataChange_Filter_Command_Click()
    OutputText = ""

    Dim endpointDescriptor As String
    endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
    'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
    'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

    ' Instantiate the client object and hook events
    Set Client3 = New EasyUAClient

    ' Prepare the arguments.
    ' Report a notification if either the StatusCode or the value change.
    Dim DataChangeFilter As New UADataChangeFilter
    DataChangeFilter.Trigger = UADataChangeTrigger_StatusValue

    Dim MonitoringParameters As New UAMonitoringParameters
    Set MonitoringParameters.DataChangeFilter = DataChangeFilter
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments1 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments1.endpointDescriptor.UrlString = endpointDescriptor
    MonitoredItemArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"
    Set MonitoredItemArguments1.MonitoringParameters = MonitoringParameters

    Dim arguments(0) As Variant
    Set arguments(0) = MonitoredItemArguments1

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Call Client3.SubscribeMultipleMonitoredItems(arguments)

    OutputText = OutputText & "Processing monitored item changed events for 20
seconds..." & vbCrLf
```

```
    Pause 20000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Client3.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client3 = Nothing
End Sub

Private Sub Client3_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to changes of a monitored item with data change
filter.

Option Explicit

Const UADataChangeTrigger_StatusValue = 1

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

' Prepare the arguments.
' Report a notification if either the StatusCode or the value change.
Dim DataChangeFilter: Set DataChangeFilter =
CreateObject("OpcLabs.EasyOpc.UA.UADataChangeFilter")
DataChangeFilter.Trigger = UADataChangeTrigger_StatusValue
'
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
Set MonitoringParameters.DataChangeFilter = DataChangeFilter
MonitoringParameters.SamplingInterval = 1000
'
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = endpointDescriptor
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
'
```

```
Dim arguments(0)
Set arguments(0) = MonitoredItemArguments1

WScript.Echo "Subscribing..."
Client.SubscribeMultipleMonitoredItems arguments

WScript.Echo "Processing monitored item changed events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display value
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= "*** Failure: " & e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

C#

```
// This example shows how to subscribe to changes of multiple monitored items and use a
data change filter.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void Filter()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification_Filter;

            Console.WriteLine("Subscribing...");
            // Report a notification if either the StatusCode or the value change.
            // The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
and can thus be used in its place.
            client.SubscribeMultipleMonitoredItems(new[]
```

```
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845",
                        new UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853",
                        new UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855",
                        new UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue))
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification_Filter(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of multiple monitored items and use a
data change filter.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub Filter()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
```

```
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification_Filter

            Console.WriteLine("Subscribing...")
            ' Report a notification if either the StatusCode Or the value change.
            ' The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
And can thus be used in its place.
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                {
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", New UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", New UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", New UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue))
                }
             )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification_Filter(ByVal sender As Object,
ByVal e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of multiple monitored items
// and use a data change filter.

type
  TClientEventHandlers122 = class
```

```
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers122.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleMonitoredItems.Filter;
const
  UADataChangeFilter_StatusValue = 1;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers122;
  DataChangeFilter: _UADataChangeFilter;
  HandleArray: OleVariant;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers122.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  // Prepare the arguments.
  // Report a notification if either the StatusCode or the value change.
  DataChangeFilter := CoUADataChangeFilter.Create;
  DataChangeFilter.Trigger := UADataChangeFilter_StatusValue;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.DataChangeFilter := DataChangeFilter;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
```

```
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  WriteLn('Subscribing...');
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of multiple monitored items
// and use a data change filter.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}

const UADataChangeFilter_StatusValue = 1;

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

// Prepare the arguments.
// Report a notification if either the StatusCode or the value change.
```

```
$DataChangeFilter = new COM("OpcLabs.EasyOpc.UA.UADataChangeFilter");
$DataChangeFilter->Trigger = UADataChangeFilter_StatusValue;
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->DataChangeFilter = $DataChangeFilter;
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;

printf("Subscribing...\n");
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of multiple monitored items
Rem and use a data change filter.

' The client object, with events
'Public WithEvents Client4 As EasyUAClient

Private Sub SubscribeMultipleMonitoredItems_Filter_Command_Click()
    OutputText = ""
```

```
    ' Instantiate the client object and hook events
    Set Client4 = New EasyUAClient

    ' Prepare the arguments.
    ' Report a notification if either the StatusCode or the value change.
    Dim DataChangeFilter As New UADataChangeFilter
    DataChangeFilter.Trigger = UADataChangeTrigger_StatusValue
    '
    Dim MonitoringParameters As New UAMonitoringParameters
    Set MonitoringParameters.DataChangeFilter = DataChangeFilter
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments1 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"
    Set MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments1.SetState ("Item1")

    Dim MonitoredItemArguments2 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"
    Set MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments2.SetState ("Item2")

    Dim MonitoredItemArguments3 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10855"
    Set MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments3.SetState ("Item3")

    Dim arguments(2) As Variant
    Set arguments(0) = MonitoredItemArguments1
    Set arguments(1) = MonitoredItemArguments2
    Set arguments(2) = MonitoredItemArguments3

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Dim handleArray As Variant
    handleArray = Client4.SubscribeMultipleMonitoredItems(arguments)

    Dim i As Long: For i = LBound(handleArray) To UBound(handleArray)
        OutputText = OutputText & "handleArray(" & i & "): " & handleArray(i) & vbCrLf
    Next

    OutputText = OutputText & "Processing monitored item changed events for 10
seconds..." & vbCrLf
    Pause 10000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Call Client4.UnsubscribeAllMonitoredItems
```

```
    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client4 = Nothing
    OutputText = OutputText & "Done." & vbCrLf
End Sub

Private Sub Client4_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

5.1.2.4.3 Changing Existing Subscription (OPC Data)

It is not necessary to unsubscribe and then subscribe again if you want to change parameters of existing subscription, such as its update rate (OPC Classic) or sampling interval (OPC UA).

In This Topic

A single subscription

Instead of unsubscribing and subscribing again, change the parameters by calling the ChangeItemSubscription (OPC Classic) or ChangeDataChangeSubscription or ChangeMonitoredItemSubscription (OPC UA) method, passing it the subscription handle, and the new parameters in form of DAGroupParameters object (in OPC Data Client.NET), EasyUASubscriptionChangeArguments (in OPC Data Client-UA) or individually a requested update rate or sampling interval, and optionally a deadband value (this is the only way in OPC Data Client-COM).

Examples for OPC Classic:

C#

```
// This example shows how change the update rate of an existing subscription.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ChangeItemSubscription
    {
        public static void Main1()
        {
```

```
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Changing subscription...");
                client.ChangeItemSubscription(handle, new DAGroupParameters(100));

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllItems();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Item changed event handler
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how change the update rate of an existing subscription.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient

    Partial Friend Class ChangeItemSubscription
        Public Shared Sub Main1()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
```

```
                Console.WriteLine("Changing subscription...")
                client.ChangeItemSubscription(handle, New DAGroupParameters(100))

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeAllItems()

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how change the update rate of an existing subscription.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Changing subscription..."
Client.ChangeItemSubscription handle, 100

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.DisconnectObject Client
Set Client = Nothing
```

```
Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq
End Sub
```

Examples for OPC XML-DA:

C#

```
// This example shows how change the update rate of an existing OPC XML-DA subscription.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class ChangeItemSubscription
    {
        public static void Main1Xml()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeItem(
                    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                    "Dynamic/Analog Types/Int",
                    2000,
                    state: null);

                Console.WriteLine("Waiting for 20 seconds...");
                Thread.Sleep(20 * 1000);

                Console.WriteLine("Changing subscription...");
                client.ChangeItemSubscription(handle, new DAGroupParameters(500));

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllItems();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }
```

```
        // Item changed event handler
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how change the update rate of an existing OPC XML-DA subscription.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess.Xml

    Partial Friend Class ChangeItemSubscription
        Public Shared Sub Main1Xml()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeItem(
                        "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                        "Dynamic/Analog Types/Int",
                        2000,
                        state:=Nothing)

                Console.WriteLine("Waiting for 20 seconds...")
                Thread.Sleep(20 * 1000)

                Console.WriteLine("Changing subscription...")
                client.ChangeItemSubscription(handle, New DAGroupParameters(500))

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeAllItems()

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
```

```
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Examples for OPC UA:

C#

```
// This example shows how to change the sampling rate of an existing monitored item
subscription.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class ChangeMonitoredItemSubscription
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int handle = client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Changing subscription...");
            client.ChangeMonitoredItemSubscription(handle, 100);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
```

```
        {
            if (e.Succeeded)
                Console.WriteLine(e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

Object Pascal

```
// This example shows how to change the sampling rate of an existing monitored
// item subscription.

type
  TClientEventHandlers110 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers110.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.AttributeData.ToString)
  else
      WriteLn('*** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure ChangeMonitoredItemSubscription.Main;
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers110;
  Handle: Cardinal;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers110.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  Handle := Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  Client.ChangeMonitoredItemSubscription(Handle, 100);
```

```
  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how to change the sampling rate of an existing monitored item
subscription.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class ChangeMonitoredItemSubscription
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handle As Integer = _
                client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Changing subscription...")
            client.ChangeMonitoredItemSubscription(handle, 100)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
```

```
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal e
As EasyUADataChangeNotificationEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to change the sampling rate of an existing monitored item
subscription.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeDataChange( _
    "http://opcua.demo-this.com:51211/UA/SampleServer", _
    "nsu=http://test.org/UA/Data/;i=10853", _
    1000)

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Changing subscription..."
Client.ChangeMonitoredItemSubscription handle, 100

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display =
e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

Multiple subscriptions

For changing parameters of multiple subscriptions in an efficient manner, call the ChangeMultipleItemSubscriptions (OPC Classic), or ChangeMultipleDataChangeSubscriptions or ChangeMultipleMonitoredItemSubscriptions (OPC UA) method.

Examples for OPC Classic:

VBScript

```
Rem This example shows how change the update rate of multiple existing subscriptions.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handleArray: handleArray = Client.SubscribeMultipleItems(arguments)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Changing subscriptions..."
```

```
Dim HandleGroupArguments1: Set HandleGroupArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAHandleGroupArguments")
HandleGroupArguments1.Handle = handleArray(0)
HandleGroupArguments1.GroupParameters.RequestedUpdateRate = 100

Dim HandleGroupArguments2: Set HandleGroupArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAHandleGroupArguments")
HandleGroupArguments2.Handle = handleArray(1)
HandleGroupArguments2.GroupParameters.RequestedUpdateRate = 100

Dim subscriptionChangeArguments(1)
Set subscriptionChangeArguments(0) = HandleGroupArguments1
Set subscriptionChangeArguments(1) = HandleGroupArguments2

Client.ChangeMultipleItemSubscriptions subscriptionChangeArguments

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.DisconnectObject Client
Set Client = Nothing



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq
End Sub
```

Examples for OPC UA:

C#

```
// This example shows how change the sampling rate of multiple existing monitored item
subscriptions.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class ChangeMultipleMonitoredItemSubscriptions
    {
```

```
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Changing subscriptions...");
            client.ChangeMultipleMonitoredItemSubscriptions(handleArray, 100);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

Object Pascal

```
// This example shows how change the sampling rate of multiple existing monitored item
subscriptions.


type
  TClientEventHandlers111 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers111.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure ChangeMultipleMonitoredItemSubscriptions.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers111;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  OldMonitoringParameters, NewMonitoringParameters: _UAMonitoringParameters;
  SubscriptionChangeArguments: OleVariant;
  SubscriptionChangeArguments1, SubscriptionChangeArguments2,
SubscriptionChangeArguments3:
    _EasyUASubscriptionChangeArguments;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers111.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  OldMonitoringParameters := CoUAMonitoringParameters.Create;
  OldMonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := OldMonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
```

```
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := OldMonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := OldMonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Changing subscriptions...');
  NewMonitoringParameters := CoUAMonitoringParameters.Create;
  NewMonitoringParameters.SamplingInterval := 100;
  SubscriptionChangeArguments1 := CoEasyUASubscriptionChangeArguments.Create;
  SubscriptionChangeArguments1.Handle := Cardinal(HandleArray[0]);
  SubscriptionChangeArguments1.MonitoringParameters := NewMonitoringParameters;
  SubscriptionChangeArguments2 := CoEasyUASubscriptionChangeArguments.Create;
  SubscriptionChangeArguments2.Handle := Cardinal(HandleArray[1]);
  SubscriptionChangeArguments2.MonitoringParameters := NewMonitoringParameters;
  SubscriptionChangeArguments3 := CoEasyUASubscriptionChangeArguments.Create;
  SubscriptionChangeArguments3.Handle := Cardinal(HandleArray[2]);
  SubscriptionChangeArguments3.MonitoringParameters := NewMonitoringParameters;
  SubscriptionChangeArguments := VarArrayCreate([0, 2], varVariant);
  SubscriptionChangeArguments[0] := SubscriptionChangeArguments1;
  SubscriptionChangeArguments[1] := SubscriptionChangeArguments2;
  SubscriptionChangeArguments[2] := SubscriptionChangeArguments3;

  Client.ChangeMultipleMonitoredItemSubscriptions(SubscriptionChangeArguments);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);
```

```
  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how change the sampling rate of multiple existing monitored item
subscriptions.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class ChangeMultipleMonitoredItemSubscriptions
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                })

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Changing subscriptions...")
            client.ChangeMultipleMonitoredItemSubscriptions(handleArray, 100)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
```

```
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal e
As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how change the sampling rate of multiple existing monitored item
subscriptions.

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim OldMonitoringParameters: Set OldMonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
OldMonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = OldMonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = OldMonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = OldMonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
```

```
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Changing subscriptions..."
Dim NewMonitoringParameters: Set NewMonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
NewMonitoringParameters.SamplingInterval = 100
Dim SubscriptionChangeArguments1: Set SubscriptionChangeArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUASubscriptionChangeArguments")
SubscriptionChangeArguments1.Handle = handleArray(0)
Set SubscriptionChangeArguments1.MonitoringParameters = NewMonitoringParameters
Dim SubscriptionChangeArguments2: Set SubscriptionChangeArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUASubscriptionChangeArguments")
SubscriptionChangeArguments2.Handle = handleArray(1)
Set SubscriptionChangeArguments2.MonitoringParameters = NewMonitoringParameters
Dim SubscriptionChangeArguments3: Set SubscriptionChangeArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUASubscriptionChangeArguments")
SubscriptionChangeArguments3.Handle = handleArray(2)
Set SubscriptionChangeArguments3.MonitoringParameters = NewMonitoringParameters
Dim subscriptionChangeArguments(2)
Set subscriptionChangeArguments(0) = SubscriptionChangeArguments1
Set subscriptionChangeArguments(1) = SubscriptionChangeArguments2
Set subscriptionChangeArguments(2) = SubscriptionChangeArguments3
Client.ChangeMultipleMonitoredItemSubscriptions subscriptionChangeArguments

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display =
e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

5.1.2.4.4 Unsubscribing (OPC Data)

From a single item

In This Topic

If you no longer want to receive item change notifications, you need to unsubscribe from them. In OPC Data Client.NET (OPC Classic), to unsubscribe from a single OPC item, call the UnsubscribeItem method, passing it the subscription handle. In OPC Data Client-UA, to unsubscribe from a single monitored item, call the UnsubscribeMonitoredItem method, passing it the subscription handle as well.

Examples for OPC Classic:

C#

```
// This example shows how subscribe to changes of multiple items, and unsubscribe from
one of them.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    class UnsubscribeItem
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged;

                int[] handleArray = client.SubscribeMultipleItems(
                    new[] {
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Sine (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, null)
                        });

                Console.WriteLine("Processing item changed events for 30 seconds...");
                Thread.Sleep(30 * 1000);

                Console.WriteLine("Unsubscribing from the first item...");
                client.UnsubscribeItem(handleArray[0]);

                Console.WriteLine();

                Console.WriteLine("Processing item changed events for 30 seconds...");
                Thread.Sleep(30 * 1000);
            }
        }

        // Item changed event handler
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
```

```
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq);
            else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of multiple items, and unsubscribe from
one of them.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class UnsubscribeItem
        Shared Sub Main1()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged

                Dim handleArray = client.SubscribeMultipleItems(
                        New DAItemGroupArguments() _
                        { _
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, Nothing),
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, Nothing),
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Sine (1 min)", 1000, Nothing),
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, Nothing)
                        })

                Console.WriteLine("Processing item changed events for 30 seconds...")
                Thread.Sleep(30 * 1000)

                Console.WriteLine("Unsubscribing from the first item...")
                client.UnsubscribeItem(handleArray(0))

                Console.WriteLine()

                Console.WriteLine("Processing item changed events for 30 seconds...")
                Thread.Sleep(30 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged(sender As Object, e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq)
            Else
                Console.WriteLine("{0} *** Failure: {1}",
```

```
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how unsubscribe from changes of a single item.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeItem handle

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq
End Sub
```

Examples for OPC UA:

C#

```
// This example shows how unsubscribe from changes of a single monitored item.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class UnsubscribeMonitoredItem
    {
        public static void Main1()
        {
```

```
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int handle = client.SubscribeDataChange(
                endpointDescriptor,
                "nsu=http://test.org/UA/Data/;i=10853",
                1000);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeMonitoredItem(handle);

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how unsubscribe from changes of a single monitored item.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class UnsubscribeMonitoredItem
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handle As Integer =
                client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeMonitoredItem(handle)

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal e
As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how unsubscribe from changes of a single monitored item.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client= CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeDataChange("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10853", 1000)

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeMonitoredItem handle

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000
```

```
Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display =
e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

From multiple items

In OPC Data Client.NET (OPC Classic), to unsubscribe from multiple OPC items in an efficient manner, call the UnsubscribeMultipleItems method (instead of calling UnsubscribeItem in a loop), passing it an array of subscription handles. In OPC Data Client-UA, to unsubscribe from multiple monitored items in an efficient manner, call the UnsubscribeMultipleMonitoredItems method (instead of calling UnsubscribeMonitoredItem in a loop), passing it an array of subscription handles.

Examples for OPC Classic:

VBScript

```
Rem This example shows how to unsubscribe from changes of multiple items.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000
```

```
Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handleArray: handleArray = Client.SubscribeMultipleItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from two items..."
Dim handles1(1)
handles1(0) = handleArray(1)
handles1(1) = handleArray(2)
Client.UnsubscribeMultipleItems handles1

WScript.Echo "Processing item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from all remaining items..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Arguments.ItemDescriptor.ItemId & ": " & e.Vtq
End Sub
```

PHP

```
// This example shows how to unsubscribe from changes of multiple items.

class DEasyDAClientEvents {
    function ItemChanged($varSender, $varE)
    {
        if ($varE->Succeeded)
        {
            printf("%s: %s\n", $varE->Arguments->ItemDescriptor->ItemId, $varE->Vtq-
>ToString());
        }
```

```
        else
        {
            printf("*** Failure: %s\n", $varE->ErrorMessageBrief);
        }
    }
}

$ItemSubscriptionArguments1 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments1->ItemDescriptor->ItemID = "Simulation.Random";
$ItemSubscriptionArguments1->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments2 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments2->ItemDescriptor->ItemID = "Trends.Ramp (1 min)";
$ItemSubscriptionArguments2->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments3 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments3->ItemDescriptor->ItemID = "Trends.Sine (1 min)";
$ItemSubscriptionArguments3->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments4 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments4->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemSubscriptionArguments4->GroupParameters->RequestedUpdateRate = 1000;

$arguments[0] = $ItemSubscriptionArguments1;
$arguments[1] = $ItemSubscriptionArguments2;
$arguments[2] = $ItemSubscriptionArguments3;
$arguments[3] = $ItemSubscriptionArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$Events = new DEasyDAClientEvents();
com_event_sink($Client, $Events, "DEasyDAClientEvents");

print "Subscribing...\n";
$handleArray = $Client->SubscribeMultipleItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %s\n", $i, $handleArray[$i]);
}

print "Processing item changed events for 10 seconds...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

print "Unsubscribing from two items...\n";
$handles1[0] = $handleArray[1];
$handles1[1] = $handleArray[2];
$Client->UnsubscribeMultipleItems($handles1);

print "Processing item changed events for 10 seconds...\n";
```

```
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

print "Unsubscribing from all remaining items...\n";
$Client->UnsubscribeAllItems;

print "Waiting for 5 seconds...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Examples for OPC UA:

C#

```
// This example shows how to unsubscribe from changes of multiple items.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class UnsubscribeMultipleMonitoredItems
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeMultipleMonitoredItems(handleArray);

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }
```

```
        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to unsubscribe from changes of multiple items.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class UnsubscribeMultipleMonitoredItems
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
            )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeMultipleMonitoredItems(handleArray)
```

```
            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal e
As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to unsubscribe from changes of multiple items.

type
  TClientEventHandlers127 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers127.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure UnsubscribeMultipleMonitoredItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers127;
  Handle: Cardinal;
  HandleArray: OleVariant;
  HandleSafeArray: PSafeArray;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
```

```
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers127.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  HandleSafeArray := Client.SubscribeMultipleMonitoredItems(Arguments);
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(HandleSafeArray);

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeMultipleMonitoredItems(HandleArray);

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VBScript

```
Rem This example shows how to unsubscribe from changes of multiple items.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client= CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
MonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from two monitored items..."
Dim handles1(1)
handles1(0) = handleArray(1)
handles1(1) = handleArray(2)
Client.UnsubscribeMultipleMonitoredItems handles1

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from all remaining items..."
Client.UnsubscribeAllMonitoredItems
```

```
WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display =
e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

From just some items

Obviously, when you are subscribing from multiple items, you can either choose to subscribe from the same set of items you used when subscribing (in which case you can simply pass the array of handles received from the SubscribeMultipleXXXX method to the UnsubscribeMultipleXXXX method), or, you can choose which items you unsubscribe from, picking just the handles you want.

Examples for OPC UA:

Object Pascal

```
// This example shows how to unsubscribe from changes of just some monitored
// items.

type
  TClientEventHandlers128 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers128.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure UnsubscribeMultipleMonitoredItems.Some;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
```

```
  ClientEventHandlers: TClientEventHandlers128;
  Handle: Cardinal;
  HandleArray: OleVariant;
  HandleSafeArray: PSafeArray;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  SelectedHandles: OleVariant;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers128.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn;
  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  HandleSafeArray := Client.SubscribeMultipleMonitoredItems(Arguments);
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(HandleSafeArray);

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn;
  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn;
```

```
  WriteLn('Unsubscribing from 2 monitored items...');
  SelectedHandles := VarArrayCreate([0, 1], varVariant);
  // We will unsubscribe from the first and third monitored item we have
  // previously subscribed to.
  SelectedHandles[0] := HandleArray[0];
  SelectedHandles[1] := HandleArray[2];
  Client.UnsubscribeMultipleMonitoredItems(SelectedHandles);

  WriteLn;
  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn;
  WriteLn('Unsubscribing from all remaining monitored items...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn;
  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn;
  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to unsubscribe from changes of just some monitored
// items.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}



// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("\n");
printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
```

```
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("\n");
printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("\n");
printf("Unsubscribing from 2 monitored items...\n");
$SelectedHandles[0] = $handleArray[0];
$SelectedHandles[1] = $handleArray[2];
$Client->UnsubscribeMultipleMonitoredItems($SelectedHandles);

printf("\n");
printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("\n");
printf("Unsubscribing from all remaining monitored items...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("\n");
printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

From all items

In OPC Data Client Classic, you can also unsubscribe from all items you have previously subscribed to (on the same

instance of EasyDAClient object) by calling the UnsubscribeAllItems method. In OPC Data Client-UA, you can unsubscribe from all monitored items you have previously subscribed to (on the same instance of EasyUAClient object) by calling the UnsubscribeAllMonitoredItems method.

Examples for OPC Classic:

VBScript

```
Rem This example shows how to unsubscribe from changes of all items.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handleArray: handleArray = Client.SubscribeMultipleItems(arguments)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllItems
```

```
WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Arguments.ItemDescriptor.ItemId & ": " & e.Vtq
End Sub
```

Examples for OPC UA:

C#

```
// This example shows how to unsubscribe from changes of all items.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class UnsubscribeAllMonitoredItems
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);
```

```
            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

Object Pascal

```
// This example shows how to unsubscribe from changes of all items.

type
  TClientEventHandlers126 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers126.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure UnsubscribeAllMonitoredItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers126;
  EndpointDescriptor: string;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
```

```
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers126.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to unsubscribe from changes of all items.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
```

```
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}

$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

VB.NET

```
' This example shows how to unsubscribe from changes of all items.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class UnsubscribeAllMonitoredItems
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
             )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal e
As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
```

```
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to unsubscribe from changes of all items.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
MonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000
```

```
Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display =
e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

Implicit and explicit unsubscribe

If you are no longer using the parent EasyDAClient or EasyUAClient object, you do not necessarily have to unsubscribe from the items, but it is highly recommended that you do so; if you don't, be aware of the possible consequences described below.

In OPC Data Client.NET and OPC Data Client-UA, the subscriptions will otherwise be internally alive until the .NET CLR (garbage collector) decides to finalize and destroy the parent EasyDAClient or EasyUAClient object (if ever); you cannot, however, determine that moment. You can alternatively call the Dispose method of the EasyDAClient (or EasyUAClient) object's IDisposable interface, which will unsubscribe from all items for you.

In OPC Data Client-COM, the subscriptions will otherwise be internally alive. You can alternatively release all references to the EasyDAClient object, which will unsubscribe from all items for you.

5.1.2.4.5 Obtaining Subscription Information (OPC Data)

If you need to obtain information about a subscription made or modified earlier, you have two methods available in OPC Data Client-UA.

Calling GetMonitoredItemArguments method with a handle obtained when the subscription was made will return an EasyUAMonitoredItemArguments object with all current parameters.

C#

```
// This example shows how to obtain parameters of certain monitored item subscription.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class GetMonitoredItemArguments
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Getting monitored item arguments...");
            EasyUAMonitoredItemArguments monitoredItemArguments =
                client.GetMonitoredItemArguments(handleArray[2]);
            Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor);
            Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval);
            Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval);

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Your code would do the processing here
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain parameters of certain monitored item subscription.

type
  TClientEventHandlers115 = class
    procedure Client_DataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers115.Client_DataChangeNotification(
```

```
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Your code would do the processing here
end;

class procedure GetMonitoredItemArguments.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers115;
  HandleArray: OleVariant;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers115.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.Client_DataChangeNotification;

  WriteLn('Subscribing...');
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Getting monitored item arguments...');
  MonitoredItemArguments := Client.GetMonitoredItemArguments(HandleArray[2]);

  WriteLn('NodeDescriptor: ', MonitoredItemArguments.NodeDescriptor.ToString);
  WriteLn('SamplingInterval: ', MonitoredItemArguments.MonitoringParameters.ToString);
  WriteLn('PublishingInterval: ',
MonitoredItemArguments.SubscriptionParameters.ToString);

  WriteLn('Waiting for 5 seconds...');
```

```
  PumpSleep(5*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how to obtain parameters of certain monitored item subscription.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class GetMonitoredItemArguments
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
            )

            Console.WriteLine("Getting monitored item arguments...")
            Dim monitoredItemArguments As EasyUAMonitoredItemArguments =
client.GetMonitoredItemArguments(handleArray(2))
            Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor)
            Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval)
            Console.WriteLine("PublishingInterval: {0}",
```

```
monitoredItemArguments.SubscriptionParameters.PublishingInterval)

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Your code would do the processing here
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain parameters of certain monitored item subscription.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."

Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"

Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
```

```
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

WScript.Echo "Getting monitored item arguments..."
Dim MonitoredItemArguments: Set MonitoredItemArguments =
Client.GetMonitoredItemArguments(handleArray(2))

WScript.Echo "NodeDescriptor: " & MonitoredItemArguments.NodeDescriptor
WScript.Echo "SamplingInterval: " &
MonitoredItemArguments.MonitoringParameters.SamplingInterval
WScript.Echo "PublishingInterval: " &
MonitoredItemArguments.SubscriptionParameters.PublishingInterval

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Your code would do the processing here
End Sub
```

Calling GetMonitoredItemArgumentsDictionary obtains information about all monitored item subscriptions on the EasyUAClient object. It returns a dictionary of monitored item argument objects (as above). The dictionary is keyed by the monitored item subscription handles, and its values contain information describing each subscription.

C#

```
// This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class GetMonitoredItemArgumentsDictionary
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
```

```
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Getting monitored item arguments dictionary...");
            EasyUAMonitoredItemArgumentsDictionary monitoredItemArgumentsDictionary =
                client.GetMonitoredItemArgumentsDictionary();

            foreach (EasyUAMonitoredItemArguments monitoredItemArguments in
monitoredItemArgumentsDictionary.Values)
            {
                Console.WriteLine();
                Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor);
                Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval);
                Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval);
            }

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Your code would do the processing here
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

type
  TClientEventHandlers116 = class
    procedure Client_DataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
```

```
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers116.Client_DataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Your code would do the processing here
end;

class procedure GetMonitoredItemArgumentsDictionary.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers116;
  Count: Cardinal;
  Element: OleVariant;
  HandleArray: OleVariant;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoredItemArgumentsDictionary: _EasyUAMonitoredItemArgumentsDictionary;
  MonitoredItemArgumentsEnumerator: IEnumVARIANT;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers116.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.Client_DataChangeNotification;

  WriteLn('Subscribing...');
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Getting monitored item arguments dictionary...');
```

```
  MonitoredItemArgumentsDictionary := Client.GetMonitoredItemArgumentsDictionary;

  MonitoredItemArgumentsEnumerator := MonitoredItemArgumentsDictionary.GetEnumerator;
  while (MonitoredItemArgumentsEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    MonitoredItemArguments := IUnknown(Element.Value) as _EasyUAMonitoredItemArguments;
    WriteLn;
    WriteLn('NodeDescriptor: ', MonitoredItemArguments.NodeDescriptor.ToString);
    WriteLn('SamplingInterval: ',
MonitoredItemArguments.MonitoringParameters.ToString);
    WriteLn('PublishingInterval: ',
MonitoredItemArguments.SubscriptionParameters.ToString);
  end;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class GetMonitoredItemArgumentsDictionary
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
```

```
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
            )

            Console.WriteLine("Getting monitored item arguments dictionary...")
            Dim monitoredItemArgumentsDictionary As
EasyUAMonitoredItemArgumentsDictionary = _
                    client.GetMonitoredItemArgumentsDictionary()

            For Each monitoredItemArguments As EasyUAMonitoredItemArguments In
monitoredItemArgumentsDictionary.Values
                Console.WriteLine()
                Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor)
                Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval)
                Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval)
            Next monitoredItemArguments

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Your code would do the processing here
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."

Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
```

```
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"

Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3

Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

WScript.Echo "Getting monitored item arguments dictionary..."
Dim MonitoredItemArgumentsDictionary: Set MonitoredItemArgumentsDictionary =
Client.GetMonitoredItemArgumentsDictionary

Dim DictionaryEntry: For Each DictionaryEntry In MonitoredItemArgumentsDictionary
    Dim MonitoredItemArguments: Set MonitoredItemArguments = DictionaryEntry.Value
    WScript.Echo
    WScript.Echo "NodeDescriptor: " & MonitoredItemArguments.NodeDescriptor
    WScript.Echo "SamplingInterval: " &
MonitoredItemArguments.MonitoringParameters.SamplingInterval
    WScript.Echo "PublishingInterval: " &
MonitoredItemArguments.SubscriptionParameters.PublishingInterval
Next

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Your code would do the processing here
End Sub
```

5.1.2.4.6 OPC Classic Item Changed Event or Callback

When there is a change in a value of an OPC item you have subscribed to, the EasyDAClient object generates an ItemChanged event. For subscription mechanism to be useful, you should hook one or more event handlers to this event.

To be more precise, the ItemChanged event is actually generated in other cases, too.

First of all, you always receive at least one ItemChanged event notification after you make a subscription; this notification either contains the initial data for the item, or an indication that data is not currently available. This behavior allows your application to rely on the component to provide at least some information for each subscribed item.

Secondly, the ItemChanged event is generated every time the component loses connection to the item, and when it reestablishes the connection. This way, your application is informed about any problems related to the item, and can process them accordingly if needed.

You will also receive the ItemChanged notification if the quality of the item changes (not just its actual data value).

The ItemChanged event notification contains an EasyDAItemChangedEventArgs argument. You will find all kind of relevant data in this object. Some properties in this object contain valid information no matter what kind of change the notification is about. These properties are inside the Arguments property.

For further processing, your code should always inspect the value of Exception property of the event arguments. If this property is set to a null reference, the notification carries an actual change in item's data, and the Vtq property has the new value, timestamp and quality of the item, in form of DAVtq object. If the Exception property is not a null reference, there has been an error related to the item, and the Vtq property is not valid. In such case, the Exception property contains information about the problem.

The ItemChanged event handler is called on a thread determined by the EasyDAClient component. For details, please refer to "Multithreading and Synchronization" chapter under "Advanced Topics".

In short, however, we can say that if you are writing e.g. Windows Forms application, the component takes care of calling the event handler on the user interface thread of the form, making it safe for your code to update controls on the form, or do other form-related actions, from the event handler.

5.1.2.4.7 OPC UA Monitored Item Changed Event or Callback

When there is a significant change related to monitored item you have subscribed to, the EasyUAClient object generates a DataChangeNotification event. What constitutes a significant change is given by the data change filter specified when the subscription was created. For subscription mechanism to be useful, you should hook one or more event handlers to this event.

To be more precise, the DataChangeNotification event is actually generated in other cases, too.

First of all, you always receive at least one DataChangeNotification event notification after you make a subscription; this notification either contains the initial data for the item, or an indication that data is not currently available. This behavior allows your application to rely on the component to provide at least some information for each subscribed item.

Secondly, the DataChangeNotification event is generated every time the component loses connection to the monitored item, and when it reestablishes the connection. This way, your application is informed about any problems related to the item, and can process them accordingly if needed.

You will also receive the DataChangeNotification notification if the status of the item changes (not just its actual data value). In fact, you can influence whether the timestamp or value changes trigger a notification, with the Trigger property in UADataChangeFilter.

The DataChangeNotification event notification contains an EasyUADataChangeNotificationEventArgs argument. You will

find all kind of relevant data in this object. Some properties in this object contain valid information no matter what kind of change the notification is about. These properties are in Arguments (containing also information such as SubscriptionParameters, MonitoringParameters, NodeDescriptor, EndpointDescriptor and State).

For further processing, your code should always inspect the value of Exception property of the event arguments. If this property is set to a null reference, the notification carries an actual change in item's data, and the AttributeData property has the new value, timestamps and status of the item, in form of UAAttributeData object. If the Exception property is not a null reference, there has been an error related to the item, and the AttributeData property is not valid. In such case, the Exception property contains information about the problem.

The DataChangeNotification event handler is called on a thread determined by the EasyUAClient component. For details, please refer to "Multithreading and Synchronization" chapter under "Advanced Topics".

In short, however, we can say that if you are writing e.g. Windows Forms application, the component takes care of calling the event handler on the user interface thread of the form, making it safe for your code to update controls on the form, or do other form-related actions, from the event handler.

5.1.2.4.8 Using Callback Methods Instead of Event Handlers (OPC Data)

Using event handlers for processing notifications is a standard way with many advantages. There also situations, however, where event handlers are not very practical. For example, if you want to do fundamentally different processing on different kinds of subscriptions, you end up with all notifications being processed by the same event handler, and you need to put in extra code to distinguish between different kinds of subscriptions they come from. Event handlers also require additional code to set up and tear down.

In This Topic

In order to overcome these problems, OPC Data Client components allow you to pass in a delegate for a callback method to subscription methods. There are subscription methods overloads that accept the callback method parameter. The callback method has the same signature (arguments) as the event handler, and is called by the component in addition to invoking the event handler (if you do not hook a handler to the event, only the callback method will be invoked). You can therefore pass in the delegate for the callback method right into the subscription method call, without setting up event handlers.

The callback method can also be specified using an anonymous delegate or a lambda expression, i.e. without having to declare the method explicitly elsewhere in your code. This is especially useful for short callback methods.

Examples for OPC Classic and XML-DA:

C#

```
// This example shows how subscribe to changes of a single item and display the value
of the item with each change,
// using a callback method specified using lambda expression.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
```

```
namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeItem
    {
        public static void CallbackLambda()
        {
            // Instantiate the client object
            var client = new EasyDAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the value
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000,
                (sender, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);

                    if (eventArgs.Succeeded)
                    {
                        Debug.Assert(eventArgs.Vtq != null);
                        Console.WriteLine(eventArgs.Vtq.ToString());
                    }
                    else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                });

            Console.WriteLine("Processing item changed events for 10 seconds...");
            Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Waiting for 2 seconds...");
            Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of a single item and display the value of
the item with each change,
' using a callback method specified using lambda expression.

Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeItem
        Shared Sub CallbackLambda()
            ' Instantiate the client object
            Dim client = New EasyDAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the value
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000,
                    Sub(sender, eventArgs)
```

```
                        Debug.Assert(eventArgs IsNot Nothing)
                        If eventArgs.Succeeded Then
                            Debug.Assert(eventArgs.Vtq IsNot Nothing)
                            Console.WriteLine(eventArgs.Vtq.ToString())
                        Else
                            Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                        End If
                    End Sub)

            Console.WriteLine("Processing item changed events for 10 seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

C#

```
// This example shows how subscribe to changes of a single item in an OPC XML-DA server
and display the value of the item
// with each change, using a callback method specified using lambda expression.

using System;
using System.Threading;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.DataAccess.Xml
{
    class SubscribeItem
    {
        public static void CallbackLambdaXml()
        {
            // Instantiate the client object
            var client = new EasyDAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the value
            client.SubscribeItem(
                "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                "Dynamic/Analog Types/Int",
                1000,
                (sender, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);

                    if (eventArgs.Succeeded)
                    {
```

```
                        Debug.Assert(eventArgs.Vtq != null);
                        Console.WriteLine(eventArgs.Vtq.ToString());
                    }
                    else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                },
                state: null);

            Console.WriteLine("Processing item changed events for 30 seconds...");
            Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Waiting for 2 seconds...");
            Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of a single item in an OPC XML-DA server
and display the value of the item
' with each change, using a callback method specified using lambda expression.

Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class SubscribeItem
        Shared Sub CallbackLambdaXml()
            ' Instantiate the client object
            Dim client = New EasyDAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the value
            client.SubscribeItem(
                    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                    "Dynamic/Analog Types/Int",
                    1000,
                    Sub(sender, eventArgs)
                        Debug.Assert(eventArgs IsNot Nothing)
                        If eventArgs.Succeeded Then
                            Debug.Assert(eventArgs.Vtq IsNot Nothing)
                            Console.WriteLine(eventArgs.Vtq.ToString())
                        Else
                            Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                        End If
                    End Sub,
                    state:=Nothing)

            Console.WriteLine("Processing item changed events for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
```

```
            client.UnsubscribeAllItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

Examples for OPC UA:

C#

```
// This example shows how to subscribe to changes of a single monitored item, and
display the value of the item with each change
// using a callback method that is provided as lambda expression.

using System;
using OpcLabs.EasyOpc.UA;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeDataChange
    {
        public static void CallbackLambda()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the value
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000,
                (sender, eventArgs) =>
                {
                    if (eventArgs.Succeeded)
                        Console.WriteLine("Value: {0}", eventArgs.AttributeData.Value);
                    else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                });

            Console.WriteLine("Processing data change events for 10 seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();
```

```
            Console.WriteLine("Waiting for 2 seconds...");
            System.Threading.Thread.Sleep(2 * 1000);
        }
    }
}
```

F#

```
// This example shows how to subscribe to changes of a single monitored item, and
display the value of the item with each change
// using a callback method that is provided as a function delegate.

module _EasyUAClient.SubscribeDataChange

open OpcLabs.EasyOpc.UA
open System
open System.Threading

let CallbackFunction =

    let endpointDescriptor =
        new UAEndpointDescriptor("opc.tcp://opcua.demo-this.com:51210/UA/SampleServer")
        // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
        // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

    // Instantiate the client object
    let client = new EasyUAClient()

    Console.WriteLine("Subscribing...");
    // The callback is a delegate that displays the value
    let handle =
        client.SubscribeDataChange(
            endpointDescriptor,
            new UANodeDescriptor("nsu=http://test.org/UA/Data/;i=10853"),
            1000,
            new EasyUADataChangeNotificationEventHandler(
                fun sender eventArgs ->
                    if eventArgs.Succeeded then Console.WriteLine("Value: {0}",
eventArgs.AttributeData.Value)
                    else Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)))

    Console.WriteLine("Processing data change events for 10 seconds...")
    Thread.Sleep(10 * 1000)

    Console.WriteLine("Unsubscribing...")
    client.UnsubscribeAllMonitoredItems()

    Console.WriteLine("Waiting for 2 seconds...")
    Thread.Sleep(2 * 1000)
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item, and
display the value of the item with each change
' using a callback method that is provided as lambda expression.
```

```
Imports OpcLabs.EasyOpc.UA

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeDataChange
        Public Shared Sub CallbackLambda()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the value
            client.SubscribeDataChange( _
                endpointDescriptor, _
                "nsu=http://test.org/UA/Data/;i=10853", _
                1000, _
                Sub(sender, eventArgs)
                    If eventArgs.Succeeded Then
                        Console.WriteLine("Value: {0}", eventArgs.AttributeData.Value)
                    Else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                    End If
                End Sub)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

More...

For subscription methods that work with multiple subscriptions at once, there is also a Callback (in OPC Classic) or DataChangeCallback (in OPC-UA) property in the arguments objects that you can use for the same purpose.

Note that if you specify a non-null callback parameter to the subscription method, the callback method will be invoked in addition to the event handlers. If you use both event handlers and callback methods in the same application, and you do not want the event handlers to process the notifications that are also processed by the callback methods, you can either

- test the Callback (or DataChangeCallback) property in the event arguments of the event handler, and if it is not a null reference, the event has been processed by the callback method and you can ignore it, or
- use two instances of the EasyDAClient (or EasyAEClient or EasyUAClient) object, and set up event handlers on one instance and use the callback methods on the other instance.

5.1.2.5 Calling Methods in OPC-UA

In OPC-UA, Methods represent the function calls of Objects. Methods are called (invoked) and return after completion, whether successful or unsuccessful. Execution times for Methods may vary, depending on the function they are performing. A Method is always a component of an Object.

In This Topic

A single method

If you want to call a method on a specific object node in OPC UA, use the EasyUAClient.CallMethod method, passing it the endpoint descriptor, object node ID, method node ID, and optionally the method input arguments (if the method has any). The input arguments are specified by an array of objects. The number of arguments and their types must conform to the method's requirements.

If the operation is successful, it returns an array of output arguments of the method call. The number of output arguments and their types are given by the UA method.

C#

```
// This example shows how to call a single method, and pass arguments to and from it.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class CallMethod
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            object[] inputs =
            {
                false,
                1,
                2,
                3,
                4,
                5,
```

```
                6,
                7,
                8,
                9,
                10
            };

            TypeCode[] typeCodes =
            {
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double
            };

            // Instantiate the client object
            var client = new EasyUAClient();

            // Perform the operation
            object[] outputs;
            try
            {
                outputs = client.CallMethod(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10755",
                    "nsu=http://test.org/UA/Data/;i=10756",
                    inputs,
                    typeCodes);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            for (int i = 0; i < outputs.Length; i++)
                Console.WriteLine("outputs[{0}]: {1}", i, outputs[i]);

            // Example output:
            //outputs[0]: False
            //outputs[1]: 1
            //outputs[2]: 2
            //outputs[3]: 3
            //outputs[4]: 4
            //outputs[5]: 5
            //outputs[6]: 6
            //outputs[7]: 7
            //outputs[8]: 8
```

```
            //outputs[9]: 9
            //outputs[10]: 10
        }
    }
}
```

VB.NET

```
' This example shows how to call a single method, and pass arguments to and from it.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class CallMethod
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Dim inputs() As Object =
            { _
                False,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            }

            Dim typeCodes() As TypeCode =
            { _
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double
            }

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Perform the operation
```

```
            Dim outputs() As Object
            Try
                outputs = client.CallMethod( _
                    endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;i=10755", _
                    "nsu=http://test.org/UA/Data/;i=10756", _
                    inputs, _
                    typeCodes)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For i As Integer = 0 To outputs.Length - 1
                Console.WriteLine("outputs[{0}]: {1}", i, outputs(i))
            Next i

            ' Example output:
            'outputs[0]: False
            'outputs[1]: 1
            'outputs[2]: 2
            'outputs[3]: 3
            'outputs[4]: 4
            'outputs[5]: 5
            'outputs[6]: 6
            'outputs[7]: 7
            'outputs[8]: 8
            'outputs[9]: 9
            'outputs[10]: 10
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to call a single method, and pass arguments to and from it.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "CallMethod.h"

namespace _EasyUAClient
{
    void CallMethod::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            CComSafeArray<VARIANT> inputs(11);
            inputs.SetAt(0, _variant_t(false));
            inputs.SetAt(1, _variant_t(1));
            inputs.SetAt(2, _variant_t(2));
            inputs.SetAt(3, _variant_t(3));
            inputs.SetAt(4, _variant_t(4));
            inputs.SetAt(5, _variant_t(5));
```

```
            inputs.SetAt(6, _variant_t(6));
            inputs.SetAt(7, _variant_t(7));
            inputs.SetAt(8, _variant_t(8));
            inputs.SetAt(9, _variant_t(9));
            inputs.SetAt(10, _variant_t(10));
            CComVariant vInputs(inputs);

            CComSafeArray<VARIANT> typeCodes(11);
            typeCodes.SetAt(0, _variant_t(TypeCode_Boolean));
            typeCodes.SetAt(1, _variant_t(TypeCode_SByte));
            typeCodes.SetAt(2, _variant_t(TypeCode_Byte));
            typeCodes.SetAt(3, _variant_t(TypeCode_Int16));
            typeCodes.SetAt(4, _variant_t(TypeCode_UInt16));
            typeCodes.SetAt(5, _variant_t(TypeCode_Int32));
            typeCodes.SetAt(6, _variant_t(TypeCode_UInt32));
            typeCodes.SetAt(7, _variant_t(TypeCode_Int64));
            typeCodes.SetAt(8, _variant_t(TypeCode_UInt64));
            typeCodes.SetAt(9, _variant_t(TypeCode_Single));
            typeCodes.SetAt(10, _variant_t(TypeCode_Double));
            CComVariant vTypeCodes(typeCodes);

            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            CComSafeArray<VARIANT> outputs;
            outputs.Attach(ClientPtr->CallMethod(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10755",
                L"nsu=http://test.org/UA/Data/;i=10756",
                &vInputs,
                &vTypeCodes));

            // Display results
            for (int i = outputs.GetLowerBound(); i <= outputs.GetUpperBound(); i++)
            {
                _variant_t output(outputs[i]);
                _variant_t vString;
                vString.ChangeType(VT_BSTR, &output);
                _tprintf(_T("outputs(%d): %s\n"), i, (LPCTSTR)CW2CT((_bstr_t)vString));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to call a single method, and pass arguments to and
// from it.

class procedure CallMethod.Main;
var
  Client: EasyUAClient;
  I: Cardinal;
  Inputs: OleVariant;
```

```
  Outputs: OleVariant;
  TypeCodes: OleVariant;
begin
  Inputs := VarArrayCreate([0, 10], varVariant);
  Inputs[0] := False;
  Inputs[1] := 1;
  Inputs[2] := 2;
  Inputs[3] := 3;
  Inputs[4] := 4;
  Inputs[5] := 5;
  Inputs[6] := 6;
  Inputs[7] := 7;
  Inputs[8] := 8;
  Inputs[9] := 9;
  Inputs[10] := 10;

  TypeCodes := VarArrayCreate([0, 10], varVariant);
  TypeCodes[0] := TypeCode_Boolean;
  TypeCodes[1] := TypeCode_SByte;
  TypeCodes[2] := TypeCode_Byte;
  TypeCodes[3] := TypeCode_Int16;
  TypeCodes[4] := TypeCode_UInt16;
  TypeCodes[5] := TypeCode_Int32;
  TypeCodes[6] := TypeCode_UInt32;
  TypeCodes[7] := TypeCode_Int64;
  TypeCodes[8] := TypeCode_UInt64;
  TypeCodes[9] := TypeCode_Single;
  TypeCodes[10] := TypeCode_Double;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Outputs).VType := varArray or varVariant;
  TVarData(Outputs).VArray := PVarArray(Client.CallMethod(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10755',
    'nsu=http://test.org/UA/Data/;i=10756',
    PSafeArray(TVarData(Inputs).VArray),
    PSafeArray(TVarData(TypeCodes).VArray)));

  // Display results
  for I := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
      try
        WriteLn('outputs(', I, '): ', Outputs[I]);
      except
        on EVariantError do WriteLn('*** Error displaying the value');
      end;
end;
```

Object Pascal

```
// This example shows how to call a single method, and pass arguments to and
// from it.

class procedure CallMethod.Main;
var
```

```
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  Inputs: OleVariant;
  Outputs: OleVariant;
  TypeCodes: OleVariant;
begin
  Inputs := VarArrayCreate([0, 10], varVariant);
  Inputs[0] := False;
  Inputs[1] := 1;
  Inputs[2] := 2;
  Inputs[3] := 3;
  Inputs[4] := 4;
  Inputs[5] := 5;
  Inputs[6] := 6;
  Inputs[7] := 7;
  Inputs[8] := 8;
  Inputs[9] := 9;
  Inputs[10] := 10;

  TypeCodes := VarArrayCreate([0, 10], varVariant);
  TypeCodes[0] := TypeCode_Boolean;
  TypeCodes[1] := TypeCode_SByte;
  TypeCodes[2] := TypeCode_Byte;
  TypeCodes[3] := TypeCode_Int16;
  TypeCodes[4] := TypeCode_UInt16;
  TypeCodes[5] := TypeCode_Int32;
  TypeCodes[6] := TypeCode_UInt32;
  TypeCodes[7] := TypeCode_Int64;
  TypeCodes[8] := TypeCode_UInt64;
  TypeCodes[9] := TypeCode_Single;
  TypeCodes[10] := TypeCode_Double;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  try
    TVarData(Outputs).VType := varArray or varVariant;
    TVarData(Outputs).VArray := PVarArray(Client.CallMethod(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10755',
      'nsu=http://test.org/UA/Data/;i=10756',
      Inputs,
      TypeCodes));
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  for I := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
      WriteLn('outputs(', I, '): ', Outputs[I]);
end;
```

PHP

```php
// This example shows how to call a single method, and pass arguments to and from it.

$inputs[0] = false;
$inputs[1] = 1;
$inputs[2] = 2;
$inputs[3] = 3;
$inputs[4] = 4;
$inputs[5] = 5;
$inputs[6] = 6;
$inputs[7] = 7;
$inputs[8] = 8;
$inputs[9] = 9;
$inputs[10] = 10;

$typeCodes[0] = 3;    // TypeCode.Boolean
$typeCodes[1] = 5;    // TypeCode.SByte
$typeCodes[2] = 6;    // TypeCode.Byte
$typeCodes[3] = 7;    // TypeCode.Int16
$typeCodes[4] = 8;    // TypeCode.UInt16
$typeCodes[5] = 9;    // TypeCode.Int32
$typeCodes[6] = 10;   // TypeCode.UInt32
$typeCodes[7] = 11;   // TypeCode.Int64
$typeCodes[8] = 12;   // TypeCode.UInt64
$typeCodes[9] = 13;   // TypeCode.Single
$typeCodes[10] = 14;  // TypeCode.Double

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $outputs = $Client->CallMethod(
        "http://opcua.demo-this.com:51211/UA/SampleServer",
        "nsu=http://test.org/UA/Data/;i=10755",
        "nsu=http://test.org/UA/Data/;i=10756",
        $inputs,
        $typeCodes);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display results
for ($i = 0; $i < count($outputs); $i++)
{
    printf("outputs[%d]: %s\n", $i, $outputs[$i]);
}
```

Python

```python
# This example shows how to call a single method, and pass arguments to and from it.
```

```
import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

#
inputs = [False, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
typeCodes = [
    3,  # TypeCode.Boolean
    5,  # TypeCode.SByte
    6,  # TypeCode.Byte
    7,  # TypeCode.Int16
    8,  # TypeCode.UInt16
    9,  # TypeCode.Int32
    10, # TypeCode.UInt32
    11, # TypeCode.Int64
    12, # TypeCode.UInt64
    13, # TypeCode.Single
    14  # TypeCode.Double
    ]


# Perform the operation
outputs = client.CallMethod(
    'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10755',
    'nsu=http://test.org/UA/Data/;i=10756',
    inputs,
    typeCodes)
# outputs[0] contains the actual results; outputs[1] and outputs[2] are copies of
inputs and typeCodes.


# Display results
for i, value in enumerate(outputs[0]):
     print('outputs[', i, ']: ', value)

# Example output:
#outputs[0]: False
#outputs[1]: 1
#outputs[2]: 2
#outputs[3]: 3
#outputs[4]: 4
#outputs[5]: 5
#outputs[6]: 6
#outputs[7]: 7
#outputs[8]: 8
#outputs[9]: 9.0
#outputs[10]: 10.0
```

Visual Basic (VB 6.)

```
Rem This example shows how to call a single method, and pass arguments to and from it.

Private Sub CallMethod_Main_Command_Click()
```

```
    OutputText = ""

    Dim inputs(10)
    inputs(0) = False
    inputs(1) = 1
    inputs(2) = 2
    inputs(3) = 3
    inputs(4) = 4
    inputs(5) = 5
    inputs(6) = 6
    inputs(7) = 7
    inputs(8) = 8
    inputs(9) = 9
    inputs(10) = 10

    Dim typeCodes(10)
    typeCodes(0) = 3    ' TypeCode.Boolean
    typeCodes(1) = 5    ' TypeCode.SByte
    typeCodes(2) = 6    ' TypeCode.Byte
    typeCodes(3) = 7    ' TypeCode.Int16
    typeCodes(4) = 8    ' TypeCode.UInt16
    typeCodes(5) = 9    ' TypeCode.Int32
    typeCodes(6) = 10   ' TypeCode.UInt32
    typeCodes(7) = 11   ' TypeCode.Int64
    typeCodes(8) = 12   ' TypeCode.UInt64
    typeCodes(9) = 13   ' TypeCode.Single
    typeCodes(10) = 14  ' TypeCode.Double

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Dim outputs As Variant
    outputs = Client.CallMethod( _
        "http://opcua.demo-this.com:51211/UA/SampleServer", _
        "nsu=http://test.org/UA/Data/;i=10755", _
        "nsu=http://test.org/UA/Data/;i=10756", _
        inputs, _
        typeCodes)
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    Dim i: For i = LBound(outputs) To UBound(outputs)
        On Error Resume Next
        OutputText = OutputText & "outputs(" & i & "): " & outputs(i) & vbCrLf
        If Err <> 0 Then OutputText = OutputText & "*** Error" & vbCrLf ' occurrs with
types not recognized by VB6
        On Error GoTo 0
    Next
End Sub
```

VBScript

```
Rem This example shows how to call a single method, and pass arguments to and from it.

Option Explicit

Dim inputs(10)
inputs(0) = False
inputs(1) = 1
inputs(2) = 2
inputs(3) = 3
inputs(4) = 4
inputs(5) = 5
inputs(6) = 6
inputs(7) = 7
inputs(8) = 8
inputs(9) = 9
inputs(10) = 10

Dim typeCodes(10)
typeCodes(0) = 3    ' TypeCode.Boolean
typeCodes(1) = 5    ' TypeCode.SByte
typeCodes(2) = 6    ' TypeCode.Byte
typeCodes(3) = 7    ' TypeCode.Int16
typeCodes(4) = 8    ' TypeCode.UInt16
typeCodes(5) = 9    ' TypeCode.Int32
typeCodes(6) = 10   ' TypeCode.UInt32
typeCodes(7) = 11   ' TypeCode.Int64
typeCodes(8) = 12   ' TypeCode.UInt64
typeCodes(9) = 13   ' TypeCode.Single
typeCodes(10) = 14  ' TypeCode.Double

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim outputs: outputs = Client.CallMethod( _
    "http://opcua.demo-this.com:51211/UA/SampleServer", _
    "nsu=http://test.org/UA/Data/;i=10755", _
    "nsu=http://test.org/UA/Data/;i=10756", _
    inputs, _
    typeCodes)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim i: For i = LBound(outputs) To UBound(outputs)
    On Error Resume Next
    WScript.Echo "outputs(" & i & "): " & outputs(i)
    If Err <> 0 Then WScript.Echo "*** Error"   ' occurrs with types not recognized by
VBScript
    On Error Goto 0
```

Next

Multiple methods; argument type conversion

For making multiple method calls (on the same or different object nodes) in an efficient manner, use the EasyUAClient.CallMultipleMethods method. You pass in an array of UACallArguments objects, each specifying the location of the object node, the method node, and the input arguments to be passed to the method.

If the originating tool or language cannot directly supply the argument in a type that is required for the method by the OPC UA server, you can have the value converted to the desired type internally by OPC Data Client. The corresponding type information can be passed in some method overrides, or inside a UACallArguments object (using the InputTypeCodes, InputTypeFullName or InputTypes property).

C#

```
// This example shows how to call multiple methods, and pass arguments to and
// from them.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class CallMultipleMethods
    {
        public static void Main1()
        {

            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor = "nsu=http://test.org/UA/Data/;i=10755";

            object[] inputs1 =
            {
                false,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            };
```

```
TypeCode[] typeCodes1 =
{
    TypeCode.Boolean,
    TypeCode.SByte,
    TypeCode.Byte,
    TypeCode.Int16,
    TypeCode.UInt16,
    TypeCode.Int32,
    TypeCode.UInt32,
    TypeCode.Int64,
    TypeCode.UInt64,
    TypeCode.Single,
    TypeCode.Double
};

object[] inputs2 =
{
    false,
    1,
    2,
    3,
    4,
    5,
    6,
    7,
    8,
    9,
    10,
    "eleven"
};

TypeCode[] typeCodes2 =
{
    TypeCode.Boolean,
    TypeCode.SByte,
    TypeCode.Byte,
    TypeCode.Int16,
    TypeCode.UInt16,
    TypeCode.Int32,
    TypeCode.UInt32,
    TypeCode.Int64,
    TypeCode.UInt64,
    TypeCode.Single,
    TypeCode.Double,
    TypeCode.String
};

// Instantiate the client object
var client = new EasyUAClient();

// Perform the operation
ValueArrayResult[] results = client.CallMultipleMethods(new[]
{
    new UACallArguments(endpointDescriptor, nodeDescriptor,
        "nsu=http://test.org/UA/Data/;i=10756", inputs1, typeCodes1),
    new UACallArguments(endpointDescriptor, nodeDescriptor,
```

```
                "nsu=http://test.org/UA/Data/;i=10774", inputs2, typeCodes2)
        });

        // Display results
        for (int i = 0; i < results.Length; i++)
        {
            Console.WriteLine();
            Console.WriteLine("results[{0}]:", i);

            ValueArrayResult result = results[i];
            if (result.Succeeded)
            {
                object[] outputs = result.ValueArray;
                Debug.Assert(outputs != null);

                for (int j = 0; j < outputs.Length; j++)
                    Console.WriteLine("    outputs[{0}]: {1}", j, outputs[j]);
            }
            else
                Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief);
        }

        // Example output:
        //results[0]:
        //outputs[0]: False
        //outputs[1]: 1
        //outputs[2]: 2
        //outputs[3]: 3
        //outputs[4]: 4
        //outputs[5]: 5
        //outputs[6]: 6
        //outputs[7]: 7
        //outputs[8]: 8
        //outputs[9]: 9
        //outputs[10]: 10

        //results[1]:
        //outputs[0]: False
        //outputs[1]: 1
        //outputs[2]: 2
        //outputs[3]: 3
        //outputs[4]: 4
        //outputs[5]: 5
        //outputs[6]: 6
        //outputs[7]: 7
        //outputs[8]: 8
        //outputs[9]: 9
        //outputs[10]: 10
        //outputs[11]: eleven
    }
  }
}
```

VB.NET

```
' This example shows how to call multiple methods, and pass arguments to and
' from them.
```

```
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class CallMultipleMethods
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            Dim nodeDescriptor As UANodeDescriptor =
"nsu=http://test.org/UA/Data/;i=10755"

            Dim inputs1() As Object =
            { _
                False,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            }

            Dim typeCodes1() As TypeCode =
            { _
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double
            }

            Dim inputs2() As Object =
            { _
                False,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
```

```
                8,
                9,
                10,
                "eleven"
            }

            Dim typeCodes2() As TypeCode =
            { _
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double,
                TypeCode.String
            }

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Perform the operation
            Dim results() As ValueArrayResult = client.CallMultipleMethods(New
UACallArguments() _
    { _
        New UACallArguments(endpointDescriptor, nodeDescriptor, _
            "nsu=http://test.org/UA/Data/;i=10756", inputs1, typeCodes1), _
        New UACallArguments(endpointDescriptor, nodeDescriptor, _
            "nsu=http://test.org/UA/Data/;i=10774", inputs2, typeCodes2) _
    } _
            )

            ' Display results
            For i As Integer = 0 To results.Length - 1
                Console.WriteLine()
                Console.WriteLine("results[{0}]:", i)

                Dim result As ValueArrayResult = results(i)
                If result.Succeeded Then
                    Dim outputs As Object() = result.ValueArray
                    Debug.Assert(outputs IsNot Nothing)

                    For j As Integer = 0 To outputs.Length - 1
                        Console.WriteLine("    outputs[{0}]: {1}", j, outputs(j))
                    Next j
                Else
                    Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief)
                End If
            Next i

            ' Example output:
            'outputs[0]: False
            'outputs[1]: 1
```

```
            'outputs[2]: 2
            'outputs[3]: 3
            'outputs[4]: 4
            'outputs[5]: 5
            'outputs[6]: 6
            'outputs[7]: 7
            'outputs[8]: 8
            'outputs[9]: 9
            'outputs[10]: 10

            'results[1]:
            'outputs[0]: False
            'outputs[1]: 1
            'outputs[2]: 2
            'outputs[3]: 3
            'outputs[4]: 4
            'outputs[5]: 5
            'outputs[6]: 6
            'outputs[7]: 7
            'outputs[8]: 8
            'outputs[9]: 9
            'outputs[10]: 10
            'outputs[11]: eleven
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to call multiple methods, and pass arguments to and from
them.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "CallMultipleMethods.h"

namespace _EasyUAClient
{
    void CallMultipleMethods::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            CComSafeArray<VARIANT> inputs1(11);
            inputs1.SetAt(0, _variant_t(false));
            inputs1.SetAt(1, _variant_t(1));
            inputs1.SetAt(2, _variant_t(2));
            inputs1.SetAt(3, _variant_t(3));
            inputs1.SetAt(4, _variant_t(4));
            inputs1.SetAt(5, _variant_t(5));
            inputs1.SetAt(6, _variant_t(6));
            inputs1.SetAt(7, _variant_t(7));
            inputs1.SetAt(8, _variant_t(8));
            inputs1.SetAt(9, _variant_t(9));
            inputs1.SetAt(10, _variant_t(10));

            CComSafeArray<VARIANT> typeCodes1(11);
```

```
            typeCodes1.SetAt(0, _variant_t(TypeCode_Boolean));
            typeCodes1.SetAt(1, _variant_t(TypeCode_SByte));
            typeCodes1.SetAt(2, _variant_t(TypeCode_Byte));
            typeCodes1.SetAt(3, _variant_t(TypeCode_Int16));
            typeCodes1.SetAt(4, _variant_t(TypeCode_UInt16));
            typeCodes1.SetAt(5, _variant_t(TypeCode_Int32));
            typeCodes1.SetAt(6, _variant_t(TypeCode_UInt32));
            typeCodes1.SetAt(7, _variant_t(TypeCode_Int64));
            typeCodes1.SetAt(8, _variant_t(TypeCode_UInt64));
            typeCodes1.SetAt(9, _variant_t(TypeCode_Single));
            typeCodes1.SetAt(10, _variant_t(TypeCode_Double));

            _UACallArgumentsPtr CallArguments1Ptr(__uuidof(UACallArguments));
            CallArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            CallArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10755";
            CallArguments1Ptr->MethodNodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10756";
            CallArguments1Ptr->InputArguments = inputs1;
            CallArguments1Ptr->InputTypeCodes = typeCodes1;

            CComSafeArray<VARIANT> inputs2(12);
            inputs2.SetAt(0, _variant_t(false));
            inputs2.SetAt(1, _variant_t(1));
            inputs2.SetAt(2, _variant_t(2));
            inputs2.SetAt(3, _variant_t(3));
            inputs2.SetAt(4, _variant_t(4));
            inputs2.SetAt(5, _variant_t(5));
            inputs2.SetAt(6, _variant_t(6));
            inputs2.SetAt(7, _variant_t(7));
            inputs2.SetAt(8, _variant_t(8));
            inputs2.SetAt(9, _variant_t(9));
            inputs2.SetAt(10, _variant_t(10));
            inputs2.SetAt(11, _variant_t(L"eleven"));

            CComSafeArray<VARIANT> typeCodes2(12);
            typeCodes2.SetAt(0, _variant_t(TypeCode_Boolean));
            typeCodes2.SetAt(1, _variant_t(TypeCode_SByte));
            typeCodes2.SetAt(2, _variant_t(TypeCode_Byte));
            typeCodes2.SetAt(3, _variant_t(TypeCode_Int16));
            typeCodes2.SetAt(4, _variant_t(TypeCode_UInt16));
            typeCodes2.SetAt(5, _variant_t(TypeCode_Int32));
            typeCodes2.SetAt(6, _variant_t(TypeCode_UInt32));
            typeCodes2.SetAt(7, _variant_t(TypeCode_Int64));
            typeCodes2.SetAt(8, _variant_t(TypeCode_UInt64));
            typeCodes2.SetAt(9, _variant_t(TypeCode_Single));
            typeCodes2.SetAt(10, _variant_t(TypeCode_Double));
            typeCodes2.SetAt(11, _variant_t(TypeCode_String));

            _UACallArgumentsPtr CallArguments2Ptr(__uuidof(UACallArguments));
            CallArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            CallArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10755";
            CallArguments2Ptr->MethodNodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10774";
```

```
            CallArguments2Ptr->InputArguments = inputs2;
            CallArguments2Ptr->InputTypeCodes = typeCodes2;

            CComSafeArray<VARIANT> arguments(2);
            arguments.SetAt(0, _variant_t((IDispatch*)CallArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)CallArguments2Ptr));
            CComVariant vArguments(arguments);

            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->CallMultipleMethods(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _tprintf(_T("\n"));
                _tprintf(_T("results(%d):\n"), i);
                _ValueArrayResultPtr ResultPtr = results[i];

                if (ResultPtr->Exception == NULL)
                {
                    CComSafeArray<VARIANT> outputs = ResultPtr->ValueArray;
                    for (int j = outputs.GetLowerBound(); j <= outputs.GetUpperBound();
j++)
                    {
                        _variant_t output(outputs[j]);
                        _variant_t vString;
                        vString.ChangeType(VT_BSTR, &output);
                        _tprintf(_T("    outputs(%d): %s\n"), i,
(LPCTSTR)CW2CT((_bstr_t)vString));
                    }
                }
                else
                    _tprintf(_T("*** Error: %s\n"), (LPCTSTR)CW2CT(ResultPtr-
>Exception->ToString));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to call multiple methods, and pass arguments to and
// from them.

class procedure CallMultipleMethods.Main;
var
  Arguments: OleVariant;
  CallArguments1, CallArguments2: _UACallArguments;
  Client: EasyUAClient;
  I, J: Cardinal;
  Inputs1, Inputs2: OleVariant;
```

```
  Outputs: OleVariant;
  Result: _ValueArrayResult;
  Results: OleVariant;
  TypeCodes1, TypeCodes2: OleVariant;
begin
  Inputs1 := VarArrayCreate([0, 10], varVariant);
  Inputs1[0] := False;
  Inputs1[1] := 1;
  Inputs1[2] := 2;
  Inputs1[3] := 3;
  Inputs1[4] := 4;
  Inputs1[5] := 5;
  Inputs1[6] := 6;
  Inputs1[7] := 7;
  Inputs1[8] := 8;
  Inputs1[9] := 9;
  Inputs1[10] := 10;

  TypeCodes1 := VarArrayCreate([0, 10], varVariant);
  TypeCodes1[0] := TypeCode_Boolean;
  TypeCodes1[1] := TypeCode_SByte;
  TypeCodes1[2] := TypeCode_Byte;
  TypeCodes1[3] := TypeCode_Int16;
  TypeCodes1[4] := TypeCode_UInt16;
  TypeCodes1[5] := TypeCode_Int32;
  TypeCodes1[6] := TypeCode_UInt32;
  TypeCodes1[7] := TypeCode_Int64;
  TypeCodes1[8] := TypeCode_UInt64;
  TypeCodes1[9] := TypeCode_Single;
  TypeCodes1[10] := TypeCode_Double;

  CallArguments1 := CoUACallArguments.Create;
  CallArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments1.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10756';
  CallArguments1.InputArguments := PSafeArray(TVarData(Inputs1).VArray);
  CallArguments1.InputTypeCodes := PSafeArray(TVarData(TypeCodes1).VArray);

  Inputs2 := VarArrayCreate([0, 11], varVariant);
  Inputs2[0] := False;
  Inputs2[1] := 1;
  Inputs2[2] := 2;
  Inputs2[3] := 3;
  Inputs2[4] := 4;
  Inputs2[5] := 5;
  Inputs2[6] := 6;
  Inputs2[7] := 7;
  Inputs2[8] := 8;
  Inputs2[9] := 9;
  Inputs2[10] := 10;
  Inputs2[11] := 'eleven';

  TypeCodes2 := VarArrayCreate([0, 11], varVariant);
  TypeCodes2[0] := TypeCode_Boolean;
```

```
  TypeCodes2[1] := TypeCode_SByte;
  TypeCodes2[2] := TypeCode_Byte;
  TypeCodes2[3] := TypeCode_Int16;
  TypeCodes2[4] := TypeCode_UInt16;
  TypeCodes2[5] := TypeCode_Int32;
  TypeCodes2[6] := TypeCode_UInt32;
  TypeCodes2[7] := TypeCode_Int64;
  TypeCodes2[8] := TypeCode_UInt64;
  TypeCodes2[9] := TypeCode_Single;
  TypeCodes2[10] := TypeCode_Double;
  TypeCodes2[11] := TypeCode_String;

  CallArguments2 := CoUACallArguments.Create;
  CallArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments2.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10774';
  CallArguments2.InputArguments := PSafeArray(TVarData(Inputs2).VArray);
  CallArguments2.InputTypeCodes := PSafeArray(TVarData(TypeCodes2).VArray);

  Arguments := VarArrayCreate([0, 1], varVariant);
  Arguments[0] := CallArguments1;
  Arguments[1] := CallArguments2;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.CallMultipleMethods(
    PSafeArray(TVarData(Arguments).VArray)));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteLn;
    WriteLn('results(', I, '):');
    Result := IInterface(Results[I]) as _ValueArrayResult;

    if Result.Exception = nil then
      begin
        TVarData(Outputs).VType := varArray or varVariant;
        TVarData(Outputs).VArray := PVarArray(Result.ValueArray);
        for J := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
          try
            WriteLn('    ', 'outputs(', J, '): ', Outputs[J]);
          except
            on EVariantError do WriteLn('*** Error displaying the value');
          end;
      end
    else
      WriteLn('*** Error: ', Result.Exception.ToString);
  end;
end;
```

Object Pascal

```
// This example shows how to call multiple methods, and pass arguments to and
// from them.

class procedure CallMultipleMethods.Main;
var
  Arguments: OleVariant;
  CallArguments1, CallArguments2: _UACallArguments;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I, J: Cardinal;
  Inputs1, Inputs2: OleVariant;
  Outputs: OleVariant;
  Result: _ValueArrayResult;
  Results: OleVariant;
  TypeCodes1, TypeCodes2: OleVariant;
begin
  Inputs1 := VarArrayCreate([0, 10], varVariant);
  Inputs1[0] := False;
  Inputs1[1] := 1;
  Inputs1[2] := 2;
  Inputs1[3] := 3;
  Inputs1[4] := 4;
  Inputs1[5] := 5;
  Inputs1[6] := 6;
  Inputs1[7] := 7;
  Inputs1[8] := 8;
  Inputs1[9] := 9;
  Inputs1[10] := 10;

  TypeCodes1 := VarArrayCreate([0, 10], varVariant);
  TypeCodes1[0] := TypeCode_Boolean;
  TypeCodes1[1] := TypeCode_SByte;
  TypeCodes1[2] := TypeCode_Byte;
  TypeCodes1[3] := TypeCode_Int16;
  TypeCodes1[4] := TypeCode_UInt16;
  TypeCodes1[5] := TypeCode_Int32;
  TypeCodes1[6] := TypeCode_UInt32;
  TypeCodes1[7] := TypeCode_Int64;
  TypeCodes1[8] := TypeCode_UInt64;
  TypeCodes1[9] := TypeCode_Single;
  TypeCodes1[10] := TypeCode_Double;

  CallArguments1 := CoUACallArguments.Create;
  CallArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments1.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10756';
  CallArguments1.InputArguments := PSafeArray(TVarData(Inputs1).VArray);
  CallArguments1.InputTypeCodes := PSafeArray(TVarData(TypeCodes1).VArray);

  Inputs2 := VarArrayCreate([0, 11], varVariant);
  Inputs2[0] := False;
  Inputs2[1] := 1;
```

```
  Inputs2[2] := 2;
  Inputs2[3] := 3;
  Inputs2[4] := 4;
  Inputs2[5] := 5;
  Inputs2[6] := 6;
  Inputs2[7] := 7;
  Inputs2[8] := 8;
  Inputs2[9] := 9;
  Inputs2[10] := 10;
  Inputs2[11] := 'eleven';

  TypeCodes2 := VarArrayCreate([0, 11], varVariant);
  TypeCodes2[0] := TypeCode_Boolean;
  TypeCodes2[1] := TypeCode_SByte;
  TypeCodes2[2] := TypeCode_Byte;
  TypeCodes2[3] := TypeCode_Int16;
  TypeCodes2[4] := TypeCode_UInt16;
  TypeCodes2[5] := TypeCode_Int32;
  TypeCodes2[6] := TypeCode_UInt32;
  TypeCodes2[7] := TypeCode_Int64;
  TypeCodes2[8] := TypeCode_UInt64;
  TypeCodes2[9] := TypeCode_Single;
  TypeCodes2[10] := TypeCode_Double;
  TypeCodes2[11] := TypeCode_String;

  CallArguments2 := CoUACallArguments.Create;
  CallArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments2.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10774';
  CallArguments2.InputArguments := PSafeArray(TVarData(Inputs2).VArray);
  CallArguments2.InputTypeCodes := PSafeArray(TVarData(TypeCodes2).VArray);

  Arguments := VarArrayCreate([0, 1], varVariant);
  Arguments[0] := CallArguments1;
  Arguments[1] := CallArguments2;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.CallMultipleMethods(
    Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteLn;
    WriteLn('results(', I, '):');
    Result := IInterface(Results[I]) as _ValueArrayResult;

    if Result.Exception = nil then
      begin
        TVarData(Outputs).VType := varArray or varVariant;
```

```
        TVarData(Outputs).VArray := PVarArray(Result.ValueArray);
        for J := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
          WriteLn('    ', 'outputs(', J, '): ', Outputs[J]);
      end
    else
      WriteLn('*** Failure: ', Result.Exception.ToString);
  end;
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to call multiple methods, and pass arguments to and from
them.

Private Sub CallMultipleMethods_Main_Command_Click()
    OutputText = ""

    Dim inputs1(10)
    inputs1(0) = False
    inputs1(1) = 1
    inputs1(2) = 2
    inputs1(3) = 3
    inputs1(4) = 4
    inputs1(5) = 5
    inputs1(6) = 6
    inputs1(7) = 7
    inputs1(8) = 8
    inputs1(9) = 9
    inputs1(10) = 10

    Dim typeCodes1(10)
    typeCodes1(0) = 3     ' TypeCode.Boolean
    typeCodes1(1) = 5     ' TypeCode.SByte
    typeCodes1(2) = 6     ' TypeCode.Byte
    typeCodes1(3) = 7     ' TypeCode.Int16
    typeCodes1(4) = 8     ' TypeCode.UInt16
    typeCodes1(5) = 9     ' TypeCode.Int32
    typeCodes1(6) = 10    ' TypeCode.UInt32
    typeCodes1(7) = 11    ' TypeCode.Int64
    typeCodes1(8) = 12    ' TypeCode.UInt64
    typeCodes1(9) = 13    ' TypeCode.Single
    typeCodes1(10) = 14   ' TypeCode.Double

    Dim CallArguments1 As New UACallArguments
    CallArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    CallArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10755"
    CallArguments1.MethodNodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10756"
    ' Use SetXXXX methods instead of array-type property setters in Visual Basic 6.0
    CallArguments1.SetInputArguments inputs1
    CallArguments1.SetInputTypeCodes typeCodes1

    Dim inputs2(11)
    inputs2(0) = False
    inputs2(1) = 1
```

```
    inputs2(2) = 2
    inputs2(3) = 3
    inputs2(4) = 4
    inputs2(5) = 5
    inputs2(6) = 6
    inputs2(7) = 7
    inputs2(8) = 8
    inputs2(9) = 9
    inputs2(10) = 10
    inputs2(11) = "eleven"

    Dim typeCodes2(11)
    typeCodes2(0) = 3     ' TypeCode.Boolean
    typeCodes2(1) = 5     ' TypeCode.SByte
    typeCodes2(2) = 6     ' TypeCode.Byte
    typeCodes2(3) = 7     ' TypeCode.Int16
    typeCodes2(4) = 8     ' TypeCode.UInt16
    typeCodes2(5) = 9     ' TypeCode.Int32
    typeCodes2(6) = 10    ' TypeCode.UInt32
    typeCodes2(7) = 11    ' TypeCode.Int64
    typeCodes2(8) = 12    ' TypeCode.UInt64
    typeCodes2(9) = 13    ' TypeCode.Single
    typeCodes2(10) = 14   ' TypeCode.Double
    typeCodes2(11) = 18   ' TypeCode.String

    Dim CallArguments2 As New UACallArguments
    CallArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    CallArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10755"
    CallArguments2.MethodNodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10774"
    ' Use SetXXXX methods instead of array-type property setters in Visual Basic 6.0
    CallArguments2.SetInputArguments inputs2
    CallArguments2.SetInputTypeCodes typeCodes2

    Dim arguments(1) As Variant
    Set arguments(0) = CallArguments1
    Set arguments(1) = CallArguments2

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    Dim results As Variant
    results = Client.CallMultipleMethods(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "results(" & i & "):" & vbCrLf
        Dim Result As ValueArrayResult: Set Result = results(i)

        If Result.Exception Is Nothing Then
            Dim outputs As Variant: outputs = Result.ValueArray
            Dim j: For j = LBound(outputs) To UBound(outputs)
                On Error Resume Next
```

```
                OutputText = OutputText & Space(4) & "outputs(" & j & "): " &
outputs(j) & vbCrLf
                If Err <> 0 Then OutputText = OutputText & Space(4) & "*** Error" &
vbCrLf ' occurrs with types not recognized by VB6
                On Error GoTo 0
            Next
        Else
            OutputText = OutputText & "*** Error: " & Result.Exception & vbCrLf
        End If
    Next
End Sub
```

VBScript

```
Rem This example shows how to call multiple methods, and pass arguments to and from
them.

Option Explicit

Dim inputs1(10)
inputs1(0) = False
inputs1(1) = 1
inputs1(2) = 2
inputs1(3) = 3
inputs1(4) = 4
inputs1(5) = 5
inputs1(6) = 6
inputs1(7) = 7
inputs1(8) = 8
inputs1(9) = 9
inputs1(10) = 10

Dim typeCodes1(10)
typeCodes1(0) = 3     ' TypeCode.Boolean
typeCodes1(1) = 5     ' TypeCode.SByte
typeCodes1(2) = 6     ' TypeCode.Byte
typeCodes1(3) = 7     ' TypeCode.Int16
typeCodes1(4) = 8     ' TypeCode.UInt16
typeCodes1(5) = 9     ' TypeCode.Int32
typeCodes1(6) = 10    ' TypeCode.UInt32
typeCodes1(7) = 11    ' TypeCode.Int64
typeCodes1(8) = 12    ' TypeCode.UInt64
typeCodes1(9) = 13    ' TypeCode.Single
typeCodes1(10) = 14   ' TypeCode.Double

Dim CallArguments1: Set CallArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UACallArguments")
CallArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
CallArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10755"
CallArguments1.MethodNodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10756"
CallArguments1.InputArguments = inputs1
CallArguments1.InputTypeCodes = typeCodes1

Dim inputs2(11)
```

```
inputs2(0) = False
inputs2(1) = 1
inputs2(2) = 2
inputs2(3) = 3
inputs2(4) = 4
inputs2(5) = 5
inputs2(6) = 6
inputs2(7) = 7
inputs2(8) = 8
inputs2(9) = 9
inputs2(10) = 10
inputs2(11) = "eleven"

Dim typeCodes2(11)
typeCodes2(0) = 3     ' TypeCode.Boolean
typeCodes2(1) = 5     ' TypeCode.SByte
typeCodes2(2) = 6     ' TypeCode.Byte
typeCodes2(3) = 7     ' TypeCode.Int16
typeCodes2(4) = 8     ' TypeCode.UInt16
typeCodes2(5) = 9     ' TypeCode.Int32
typeCodes2(6) = 10    ' TypeCode.UInt32
typeCodes2(7) = 11    ' TypeCode.Int64
typeCodes2(8) = 12    ' TypeCode.UInt64
typeCodes2(9) = 13    ' TypeCode.Single
typeCodes2(10) = 14   ' TypeCode.Double
typeCodes2(11) = 18   ' TypeCode.String

Dim CallArguments2: Set CallArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UACallArguments")
CallArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
CallArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10755"
CallArguments2.MethodNodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10774"
CallArguments2.InputArguments = inputs2
CallArguments2.InputTypeCodes = typeCodes2

Dim arguments(1)
Set arguments(0) = CallArguments1
Set arguments(1) = CallArguments2

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
Dim results: results = Client.CallMultipleMethods(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    WScript.Echo
    WScript.Echo "results(" & i & "):"
    Dim Result: Set Result = results(i)

    If Result.Exception Is Nothing Then
        Dim outputs: outputs = Result.ValueArray
        Dim j: For j = LBound(outputs) To UBound(outputs)
```

```
            On Error Resume Next
            WScript.Echo Space(4) & "outputs(" & j & "): " & outputs(j)
            If Err <> 0 Then WScript.Echo Space(4) & "*** Error"    ' occurrs with types
not recognized by VBScript
            On Error Goto 0
        Next
    Else
        WScript.Echo "*** Error: " & Result.Exception
    End If
Next
```

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

5.1.2.6 Setting Parameters (OPC Data)

While the most information needed to perform OPC tasks is contained in arguments to method calls, there are some component-wide parameters that are not worth repeating in every method call, and also some that have wider effect that influences more than just a single method call. You can obtain and modify these parameters through properties on the EasyDAClient or EasyUAClient object.

For the EasyDAClient object, following are its instance properties, i.e. if you have created multiple EasyDAClient objects, each will have its own copy of them:

- Mode: Specifies common parameters such as allowed and desired methods of accessing the data in the OPC server.
- HoldPeriods: Specifies optimization parameters that reduce the load on the OPC server.
- UpdateRates: Specifies the "hints" for OPC update rates used when other explicit information is missing.
- Timeouts: Specifies the maximum amount of time the various operations are allowed to take.

and

- SynchronizationContext: Contains synchronization context used by the object when performing event notifications.

C#

```
// This example shows how the OPC server can quickly be disconnected after writing a
value into one of its OPC items.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientHoldPeriods
{
    class TopicWrite
    {
        public static void Main1()
        {
            var client = new EasyDAClient();
```

```
            client.InstanceParameters.HoldPeriods.TopicWrite = 100; // in milliseconds

            try
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how the OPC server can quickly be disconnected after writing a
value into one of its OPC items.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientHoldPeriods
    Friend Class TopicWrite
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            client.InstanceParameters.HoldPeriods.TopicWrite = 100 ' in milliseconds

            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
            End Try
        End Sub
    End Class
End Namespace
```

For the EasyUAClient object, following are its instance properties, i.e. if you have created multiple EasyUAClient objects, each will have its own copy of them:

- SynchronizationContext: Contains synchronization context used by the object when performing event notifications.

Instance properties can be modified from your code.

In OPC Data Client.NET and OPC Data Client-UA, if you have placed the EasyDAClient, EasyAEClient or EasyUAClient object on the designer surface, most instance properties can also be directly edited in the Properties window in Visual Studio.

In OPC Data Client-COM, your code can override the defaults if needed, by setting the properties accordingly.

Following properties are static, i.e. shared among all instances of EasyDAClient object:

- EngineParameters: Contains global parameters such as frequencies of internal tasks performed by the component.
- MachineParameters: Contains parameters related to operations that target a specific computer but not a specific OPC server, such as browsing for OPC servers using various methods.
- ClientParameters: Contains parameters that influence operations that target a specific OPC server a whole.
- TopicParameters: Contains parameters that influence operations that target a specific OPC item.

Following properties are static, i.e. shared among all instances of EasyUAClient object:

- EngineParameters: Contains global parameters such as frequencies of internal tasks performed by the component.

Following properties are static unless the EasyUAClient is configured as isolated:

- HostParameters: Contains parameters that influence the behavior of the component on the host level.
- DiscoveryParameters: Contains parameters that influence the behavior of the component for the discovery.
- MonitoredItemParameters: Contains parameters related to operations that target a specific monitored item (e.g. a retrial delay).
- SessionParameters: Contains parameters that influence operations that target an OPC-UA session, such as Endpoint Selection Policy, User Identity, and Keep Alive Interval.
- SubscriptionParameters: Contains parameters that influence operations that target an OPC-UA subscription (e.g. a retrial delay).

Please use the Reference documentation for details on meaning of various properties and their use.

Static properties can be modified from your code. If you want to modify any of the static properties, you must do it before the first instance of EasyDAClient , EasyAEClient or EasyUAObject object is created.

You can also use the following components to set the static properties:

Configuration Component	Sets Static Properties of
EasyAEClientConfiguration component - 	EasyAEClient class
EasyDAClientConfiguration component - 	EasyDAClient class
EasyUAClientConfiguration component - 	EasyUAClient class

In Visual Studio, drag the corresponding component from the Toolbox to the designer surface, select it, and you can then view and modify the settings in the Properties window.

Note that the EasyXXClientConfiguration (where **XX** is either **AE**/**DA**/**UA**) components are just a "peek" to properties of the EasyXXClient that are static in nature. For this reason, you should normally have just one instance of the EasyXXClientConfiguration component in your application, and you should place it in such a way that it gets instantiated before the first operations on the EasyXXClient object take place. You can achieve this easily e.g. by placing the EasyXXClientConfiguration component onto the main (first) form of your application.

The EasyUAClientConfiguration component also has a LogEntry event, which functions as a "projection" of the static EasyUAClient.LogEntry event. You can easily hook an event handler to this event to process log entries generated by the EasyUAClient component.

5.1.2.6.1 User Identity in QuickOPC-UA

OPC UA Servers may require that the user making a connection from the OPC UA client is authenticated, and reject the connection of the authentication fails. In addition, different users may have different permissions (authorization) for various operations on the OPC UA Server.

You can specify the identity of the user making the connection using the UserIdentity property. This property is a UserIdentity object that contains following user token infos:

- AnonymousTokenInfo: An anonymous user.
- UserNameTokenInfo: A user name token (with optional password).
- X509CertificateTokenInfo: A user token represented by an X.509 certificate.
- KerberosTokenInfo: A Kerberos (issued) user token.

Note: The Kerberos token info may represent an explicitly specified user identity (if you set KerberosTokenInfo.NetworkSecurity.CustomNetworkCredential to true and specify additional parameters, such as the user name, password, and domain), or it can represent the current user running the code (if KerberosTokenInfo.NetworkSecurity.CustomNetworkCredential is set to false).

Zero, one, or more user token infos (of different types) may be specified in the UserIdentity object. By default, no user token info is specified. The user token infos are always present (i.e. non-null), but they are only used if they are filled in with data. For example, if you leave the UserName and Password in the UserNameTokenInfo empty, the user name token will not be used. If you, however, start putting values into any of the token infos, you need to fill in everything necessary in that token, otherwise an error may occur. For an AnonymousTokenInfo, the anonymous token is used when its IsConfigured property is set to true.

You can easily create a UserIdentity with certain user token by one of the following static methods:

- UserIdentity.CreateAnonymousIdentity
- UserIdentity.CreateKerberosIdentity
- UserIdentity.CreateUserNameIdentity
- UserIdentity.CreateX509Certificate

When OPC Data Client-UA makes a connection to the OPC UA server, it selects the user token according to its built-in token selection policy. The OPC UA server is interrogated for user token policies available on the endpoint, and the OPC Data Client-UA selects the most appropriate one from them.

When you set the user identity in the above described way, i.e. in the session parameters object, it applies to all sessions (connections) made by that EasyUAClient object. In addition to this, it is also possible to specify the user identity directly for a specific connection, i.e. on the UAEndpointDescriptor object.

There is a UserIdentity property on the UAEndpointDescriptor as well, and the user token infos contained there are merged together with those coming from the session parameters, for each connection made on that endpoint. For details on easier handling user identities specified directly on an endpoint, see **OPC UA Server Endpoints (Section 4.11.1)**.

5.1.2.6.2 Server Diagnostics in OPC-UA

The DiagnosticsMasks property on the SessionParameters allows the developer to specify the types of vendor-specific diagnostics to be returned by the OPC-UA server in the responses. The information from the server responses is then made available in the Diagnostics collection of OperationResult and EasyDataChangeNotificationEventArgs objects.

5.1.3 Imperative Programming Model for OPC Classic A&E

This chapter gives you guidance in how to implement the common tasks that are needed when dealing with OPC Classic Alarms and Events server from the client side. You achieve these tasks by calling methods on the EasyAEClient object.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

5.1.3.1 Obtaining Information (OPC A&E)

Methods described in this chapter allow your application to obtain information from the underlying data source that the OPC Alarms and Events server connects to, or from the OPC server itself (getting condition state). It is assumed that your application already somehow knows how to identify the data it is interested in. If the location of the data is not known upfront, use methods described in Browsing for Information chapter first.

5.1.3.1.1 Getting Condition State (OPC A&E)

In OPC Alarms and Events, information is usually provided in form of event notifications, especially for transient (simple and tracking) events. For condition-related events, however, it is also possible to get (upon request) information about the current state of a specified condition.

If you want to obtain the current state information for the condition instance in an OPC Alarms and Events sever, call the GetConditionState method. You pass in individual arguments for machine name, server class, fully qualified source name, condition name, and optionally an array of event attributes to be returned. You will receive back an AEConditionState object holding the current state information about an OPC condition instance.

C#

```
// This example shows how to obtain current state information for the condition
instance corresponding to a Source and
// certain ConditionName.

using System;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class GetConditionState
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AEConditionState conditionState;
```

```
            try
            {
                conditionState = client.GetConditionState("",
"OPCLabs.KitEventServer.2",
                    "Simulation.ConditionState1", "Simulated");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("ConditionState:");
            Console.WriteLine("    .ActiveSubcondition: {0}",
conditionState.ActiveSubcondition);
            Console.WriteLine("    .Enabled: {0}", conditionState.Enabled);
            Console.WriteLine("    .Active: {0}", conditionState.Active);
            Console.WriteLine("    .Acknowledged: {0}", conditionState.Acknowledged);
            Console.WriteLine("    .Quality: {0}", conditionState.Quality);
            // Remark: IAEConditionState has many more properties
        }
    }
}
```

VB.NET

```
' This example shows how to obtain current state information for the condition instance
corresponding to a Source and
' certain ConditionName.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class GetConditionState
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim conditionState As AEConditionState
            Try
                conditionState = client.GetConditionState("",
"OPCLabs.KitEventServer.2", "Simulation.ConditionState1", "Simulated")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("ConditionState:")
            Console.WriteLine("    .ActiveSubcondition: {0}",
conditionState.ActiveSubcondition)
            Console.WriteLine("    .Enabled: {0}", conditionState.Enabled)
            Console.WriteLine("    .Active: {0}", conditionState.Active)
            Console.WriteLine("    .Acknowledged: {0}", conditionState.Acknowledged)
            Console.WriteLine("    .Quality: {0}", conditionState.Quality)
```

```
            ' Remark: IAEConditionState has many more properties
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain current state information for the condition
instance corresponding to a Source and
Rem certain ConditionName.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim SourceDescriptor: Set SourceDescriptor =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor.QualifiedName = "Simulation.ConditionState1"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim ConditionState: Set ConditionState = Client.GetConditionState(ServerDescriptor,
SourceDescriptor, "Simulated", Array())
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "ConditionState:"
With ConditionState
    WScript.Echo Space(4) & ".ActiveSubcondition: " & .ActiveSubcondition
    WScript.Echo Space(4) & ".Enabled: " & .Enabled
    WScript.Echo Space(4) & ".Active: " & .Active
    WScript.Echo Space(4) & ".Acknowledged: " & .Acknowledged
    WScript.Echo Space(4) & ".Quality: " & .Quality
    Rem Note that IAEConditionState has many more properties
End With
```

In OPC Data Client.NET, you can alternatively pass in a ServerDescriptor in place of machine name and server class arguments.

5.1.3.2 Modifying Information (OPC A&E)

Methods described in this chapter allow your application to modify information in the underlying data source that the OPC server connects to or in the OPC server itself (acknowledging conditions). It is assumed that your application already somehow knows how to identify the data it is interested in. If the location of the data is not known upfront, use methods

described Browsing for Information chapter first.

5.1.3.2.1 Acknowledging a Condition (OPC A&E)

If you want to acknowledge a condition in OPC Alarms and Events server, call the AcknowledgeCondition method. You pass in individual arguments for machine name, server class, fully qualified source name, condition name, and an active time and cookie corresponding to the transition of the condition you are acknowledging.

C#

```
// This example shows how to acknowledge an event condition in the OPC server.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class AcknowledgeCondition
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        static volatile bool _done;

        public static void Main1()
        {
            var eventHandler = new
EasyAENotificationEventHandler(AEClient_Notification);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications for 1 minute...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1" }
            };
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // Give the refresh operation time to complete
            Thread.Sleep(5 * 1000);

            // Trigger an acknowledgeable event
            try
            {
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
```

```
opcException.GetBaseException().Message);
                return;
            }

            _done = false;
            DateTime endTime = DateTime.Now + new TimeSpan(0, 0, 5);
            while ((!_done) && (DateTime.Now < endTime))
                Thread.Sleep(1000);

            // Give some time to also receive the acknowledgement notification
            Thread.Sleep(5 * 1000);

            AEClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void AEClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
                Console.WriteLine("Event.AcknowledgeRequired: {0}",
eventData.AcknowledgeRequired);

                if (eventData.AcknowledgeRequired)
                {
                    Console.WriteLine(">>>>> ACKNOWLEDGING THIS EVENT");
                    try
                    {
                        AEClient.AcknowledgeCondition("", "OPCLabs.KitEventServer.2",
"Simulation.ConditionState1", "Simulated",
                            eventData.ActiveTime, eventData.Cookie);
                    }
                    catch (OpcException opcException)
                    {
                        Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                        return;
                    }
                    Console.WriteLine(">>>>> EVENT ACKNOWLEDGED");
                    _done = true;
                }
            }
```

```
        }
    }
}
```

VB.NET

```
' This example shows how to acknowledge an event condition in the OPC server.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class AcknowledgeCondition

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Private Shared _done As Boolean ' volatile

        Public Shared Sub Main1()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications for 1 minute...")
            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1"}
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter)

            ' Give the refresh operation time to complete
            Thread.Sleep(5 * 1000)

            ' Trigger an acknowledgeable event
            Try
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            _done = False
            Dim endTime As Date = Date.Now + New TimeSpan(0, 0, 5)
            Do While ((Not _done)) AndAlso (Date.Now < endTime)
                Thread.Sleep(1000)
            Loop

            ' Give some time to also receive the acknowledgement notification
            Thread.Sleep(5 * 1000)
```

```
            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            Console.WriteLine()
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Dim eventData As AEEventData = e.EventData
                Console.WriteLine("EventData.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("EventData.Message: {0}", eventData.Message)
                Console.WriteLine("EventData.Active: {0}", eventData.Active)
                Console.WriteLine("EventData.Acknowledged: {0}",
eventData.Acknowledged)
                Console.WriteLine("EventData.AcknowledgeRequired: {0}",
eventData.AcknowledgeRequired)

                If eventData.AcknowledgeRequired Then
                    Console.WriteLine(">>>>> ACKNOWLEDGING THIS EVENT")
                    Try
                        AEClient.AcknowledgeCondition("", "OPCLabs.KitEventServer.2",
"Simulation.ConditionState1", "Simulated", eventData.ActiveTime, eventData.Cookie)
                    Catch opcException As OpcException
                        Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                        Exit Sub
                    End Try
                    Console.WriteLine(">>>>> EVENT ACKNOWLEDGED")
                    _done = True
                End If
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to acknowledge an event condition in the OPC server.

Option Explicit

Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")

WScript.Echo "Hooking event handler..."
WScript.ConnectObject AEClient, "AEClient_"
```

```
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim SourceDescriptor: Set SourceDescriptor =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor.QualifiedName = "Simulation.ConditionState1"

WScript.Echo "Processing event notifications for 1 minute..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.Filter.Sources = Array(SourceDescriptor)
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = AEClient.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Give the refresh operation time to complete: Waiting for 5 seconds..."
WScript.Sleep 5*1000

WScript.Echo "Triggering an acknowledgeable event..."
On Error Resume Next
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim done: done = False
Dim endTime: endTime = Now() + 5*(1/24/60/60)
While (Not done) And (Now() < endTime)
    WScript.Sleep 1000
WEnd

WScript.Echo "Give some time to also receive the acknowledgement notification: Waiting
for 5 seconds..."
WScript.Sleep 5*1000

WScript.Echo "Unsubscribing events..."
AEClient.UnsubscribeEvents handle

WScript.Echo "Unhooking event handler..."
WScript.DisconnectObject AEClient

WScript.Echo "Finished."



Rem Notification event handler
Sub AEClient_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo
```

```
    WScript.Echo "Refresh: " & e.Refresh
    WScript.Echo "RefreshComplete: " & e.RefreshComplete
    If Not (e.EventData Is Nothing) Then
        With e.EventData
            WScript.Echo "EventData.QualifiedSourceName: " & .QualifiedSourceName
            WScript.Echo "EventData.Message: " & .Message
            WScript.Echo "EventData.Active: " & .Active
            WScript.Echo "EventData.Acknowledged: " & .Acknowledged
            WScript.Echo "EventData.AcknowledgeRequired: " & .AcknowledgeRequired

            If .AcknowledgeRequired Then
                WScript.Echo ">>>>> ACKNOWLEDGING THIS EVENT"
                On Error Resume Next
                AEClient.AcknowledgeCondition ServerDescriptor, SourceDescriptor,
"Simulated", _
                    .ActiveTime, .Cookie, "aUser", ""
                If Err.Number <> 0 Then
                    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
                    Exit Sub
                End If
                On Error Goto 0
                WScript.Echo ">>>>> EVENT ACKNOWLEDGED"
                done = True
            End If
        End With
    End If
End Sub
```

Optionally, you can pass in acknowledger ID (who is acknowledging the condition), and a comment string.

You can alternatively pass in a ServerDescriptor in place of machine name and server class arguments.

5.1.3.3 Browsing for Information (OPC A&E)

OPC Data Client contains methods that allow your application to retrieve and enumerate information about OPC Alarms and Events servers that exist on the network, and data available within these servers. Your code can then make use of the information obtained, e.g. to accommodate to configuration changes dynamically.

The methods we are describing here are for programmatic browsing, with no user interface (or when the user interface is provided by our own code).

5.1.3.3.1 Browsing for OPC A&E Servers

If you want to retrieve a list of OPC Alarms and Events servers registered on a local or remote computer, call the BrowseServers method, passing it the name or address of the remote machine (use empty string for local computer).

You will receive back a ServerElementCollection object. If you want to connect to this OPC server later in your code by calling other methods, use the built-in conversion of ServerElement to String, and pass the resulting string as a serverClass argument either directly to the method call, or to a constructor of ServerDescriptor object.

Each ServerElement contains information gathered about one OPC server found on the specified machine, including things like the server's CLSID, ProgID, vendor name, and readable description.

C#

```
// This example shows how to obtain all ProgIDs of all OPC Alarms and Events servers on
the local machine.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class BrowseServers
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            ServerElementCollection serverElements;
            try
            {
                serverElements = client.BrowseServers("");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (ServerElement serverElement in serverElements)
            {
                Debug.Assert(serverElement != null);
                Console.WriteLine("serverElements[\"{0}\"].ProgId: {1}",
serverElement.Clsid, serverElement.ProgId);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain all ProgIDs of all OPC Alarms and Events servers on
the local machine.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class BrowseServers
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim serverElements As ServerElementCollection
```

```
            Try
                serverElements = client.BrowseServers("")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each serverElement As ServerElement In serverElements
                Debug.Assert(serverElement IsNot Nothing)
                Console.WriteLine("serverElements[""{0}""].ProgId: {1}",
serverElement.Clsid, serverElement.ProgId)
            Next serverElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain all ProgIDs of all OPC Alarms and Events servers
on the local machine.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim ServerElements: Set ServerElements = Client.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "ServerElements(""" & ServerElement.UrlString & """).ProgId: " &
ServerElement.ProgId
Next
```

5.1.3.3.2 Browsing for OPC A&E Nodes (Areas and Sources)

Information in an OPC Alarms and Events server is organized in a tree hierarchy (process space), where the branch nodes (*event areas*) serve organizational purposes (similar to folders in a file system), while the leaf nodes actually generate events (similar to files in a file system) – they are called *event sources*. Each node has a "short" name that is unique among other branches or leaves under the same parent branch (or a root). Event sources can be fully identified using a "fully qualified" source name, which determines the OPC event source without a need to further qualify it with its position in the tree. OPC event source may look a process tag (e.g. "FIC101", or "Device1.Block101"), or possibly a device or subsystem identification; their syntax and meaning is fully determined by the particular OPC server they are coming from.

OPC Data Client gives you methods to traverse through the address space information and obtain the information available there. It is also possible to filter the returned nodes by a server specific filter string.

If you want to retrieve a list of all event areas under a given parent area (or under a root) of the OPC server, call the BrowseAreas method. You will receive an AENodeElementCollection object. Each AENodeElement contains information gathered about one sub-area node, such as its name, or an indication whether it has children.

C#

```
// This example shows how to obtain all areas directly under the root (denoted by empty
string for the parent).

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class BrowseAreas
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AENodeElementCollection nodeElements;
            try
            {
                nodeElements = client.BrowseAreas("", "OPCLabs.KitEventServer.2", "");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AENodeElement nodeElement in nodeElements)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine("nodeElements[\"{0}\"]:", nodeElement.Name);
                Console.WriteLine("    .QualifiedName: {0}",
nodeElement.QualifiedName);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain all areas directly under the root (denoted by empty
string for the parent).

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
```

```
    Friend Class BrowseAreas
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim nodeElements As AENodeElementCollection
            Try
                nodeElements = client.BrowseAreas("", "OPCLabs.KitEventServer.2", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each nodeElement As AENodeElement In nodeElements
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine("nodeElements[""{0}""]:", nodeElement.Name)
                Console.WriteLine("    .QualifiedName: {0}", nodeElement.QualifiedName)
            Next nodeElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain all areas directly under the root (denoted by
empty string for the parent).

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseAreas("", "OPCLabs.KitEventServer.2",
"")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo "NodeElements(""" & NodeElement.Name & """):"
    With NodeElement
        WScript.Echo Space(4) & ".QualifiedName: " & .QualifiedName
    End With
Next
```

Similarly, if you want to retrieve a list of event sources under a given parent area (or under a root) of the OPC server, call the BrowseSources method. You will also receive back an AENodeElementCollection object, this time containing the event sources only. You can find information such as the fully qualified source name from the AENodeElement of any event source, extract it and pass it further to methods like GetConditionState or SubscribeEvents.

C#

```
// This example shows how to obtain all sources under the "Simulation" area.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class BrowseSources
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AENodeElementCollection nodeElements;
            try
            {
                nodeElements = client.BrowseSources("", "OPCLabs.KitEventServer.2",
"Simulation");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AENodeElement nodeElement in nodeElements)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine("nodeElements[\"{0}\"]:", nodeElement.Name);
                Console.WriteLine("    .QualifiedName: {0}",
nodeElement.QualifiedName);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain all sources under the "Simulation" area.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class BrowseSources
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()
```

```
            Dim nodeElements As AENodeElementCollection
            Try
                nodeElements = client.BrowseSources("", "OPCLabs.KitEventServer.2",
"Simulation")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each nodeElement As AENodeElement In nodeElements
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine("nodeElements[""{0}""]:", nodeElement.Name)
                Console.WriteLine("    .QualifiedName: {0}", nodeElement.QualifiedName)
            Next nodeElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain all sources under the "Simulation" area.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseSources("",
"OPCLabs.KitEventServer.2", "Simulation")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo "NodeElements(""" & NodeElement.Name & """):"
    With NodeElement
        WScript.Echo Space(4) & ".QualifiedName: " & .QualifiedName
    End With
Next
```

OPC Data Client.NET: The most generic address space browsing method is BrowseNodes. It combines the functionality of BrowseAreas and BrowseSources, and it also allows the widest range of filtering options by passing in an argument of type AEBrowseParameters.

5.1.3.3.3 Querying for OPC A&E Event Categories

Each OPC Alarms and Events server supports a set of specific event categories. The OPC specifications define a set of

recommended categories; however, each OPC server is free to implement some more, vendor-specific event categories as well.

If you want to retrieve a list of all categories available in a given OPC server, call the QueryEventCategories method. You will receive back an AECategoryElementCollection object.

C#

```
// This example shows how to enumerate all event categories provided by the OPC server.
For each category, it displays its Id
// and description.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class QueryEventCategories
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
            {
                Debug.Assert(categoryElement != null);
                Console.WriteLine("CategoryElements[\"{0}\"].Description: {1}",
categoryElement.CategoryId, categoryElement.Description);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to enumerate all event categories provided by the OPC server.
For each category, it displays its Id
' and description.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
```

```
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class QueryEventCategories
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)
                Console.WriteLine("CategoryElements[""{0}""].Description: {1}",
categoryElement.CategoryId, categoryElement.Description)
            Next categoryElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to enumerate all event categories provided by the OPC
server. For each category, it displays its Id
Rem and description.

Option Explicit

Const AEEventTypes_All = 7

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim CategoryElements: Set CategoryElements =
Client.QueryEventCategories(ServerDescriptor, AEEventTypes_All)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim CategoryElement: For Each CategoryElement In CategoryElements
    WScript.Echo "CategoryElements(" & CategoryElement.CategoryId & ").Description: " &
CategoryElement.Description
Next
```

Each AECategoryElement contains information about one OPC event category, such as its (numeric) CategoryId, readable description, and associated event conditions and attributes. The CategoryId can be later used when creating an event filter, and is provided to you in event notifications.

C#

```
// This example shows information available about OPC event category.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._AECategoryElement
{
    class Properties
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
            {
                Debug.Assert(categoryElement != null);
                Debug.Assert(categoryElement.AttributeElements.Keys != null);
                Debug.Assert(categoryElement.ConditionElements.Keys != null);

                Console.WriteLine("Information about category {0}:", categoryElement);
                Console.WriteLine("    .CategoryId: {0}", categoryElement.CategoryId);
                Console.WriteLine("    .Description: {0}",
categoryElement.Description);
                Console.WriteLine("    .ConditionElements:");
                foreach (string conditionKey in categoryElement.ConditionElements.Keys)
                    Console.WriteLine("        {0}", conditionKey);
                Console.WriteLine("    .AttributeElements:");
                foreach (long attributeKey in categoryElement.AttributeElements.Keys)
                    Console.WriteLine("        {0}", attributeKey);
            }
        }
    }
}
```

VB.NET

```
' This example shows information available about OPC event category.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._AECategoryElement

    Friend Class Properties
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)
                Debug.Assert(categoryElement.AttributeElements.Keys IsNot Nothing)
                Debug.Assert(categoryElement.ConditionElements.Keys IsNot Nothing)

                Console.WriteLine("Information about category {0}:", categoryElement)
                Console.WriteLine("    .CategoryId: {0}", categoryElement.CategoryId)
                Console.WriteLine("    .Description: {0}", categoryElement.Description)
                Console.WriteLine("    .ConditionElements:")
                For Each conditionKey As String In
categoryElement.ConditionElements.Keys
                    Console.WriteLine("        {0}", conditionKey)
                Next conditionKey
                Console.WriteLine("    .AttributeElements:")
                For Each attributeKey As Long In categoryElement.AttributeElements.Keys
                    Console.WriteLine("        {0}", attributeKey)
                Next attributeKey
            Next categoryElement
        End Sub
    End Class

End Namespace
```

5.1.3.3.4 Querying for OPC A&E Event Conditions on a Category

The EasyAEClient.QueryCategoryConditions method finds out out event conditions supported by given event category.

5.1.3.3.5 Querying for OPC A&E Event Conditions on a Source

The EasyAEClient.QuerySourceConditions method finds out event conditions associated with the given event source.

C#

```
// This example shows how to enumerate all event conditions associated with the
specified event source.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class QuerySourceConditions
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AEConditionElementCollection conditionElements;
            try
            {
                conditionElements = client.QuerySourceConditions("",
"OPCLabs.KitEventServer.2",
                    "Simulation.ConditionState1");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AEConditionElement conditionElement in conditionElements)
            {
                Debug.Assert(conditionElement != null);
                Console.WriteLine("ConditionElements[\"{0}\"]: {1} subcondition(s)",
                    conditionElement.Name, conditionElement.SubconditionNames.Length);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to enumerate all event conditions associated with the
specified event source.
```

```
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class QuerySourceConditions
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim conditionElements As AEConditionElementCollection
            Try
                conditionElements = client.QuerySourceConditions( _
                    "", "OPCLabs.KitEventServer.2", "Simulation.ConditionState1")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each conditionElement As AEConditionElement In conditionElements
                Debug.Assert(conditionElement IsNot Nothing)
                Console.WriteLine("ConditionElements[""{0}""]: {1} subcondition(s)", _
                                  conditionElement.Name,
conditionElement.SubconditionNames.Length)
            Next conditionElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to enumerate all event conditions associated with the
specified event source.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim SourceDescriptor: Set SourceDescriptor =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor.QualifiedName = "Simulation.ConditionState1"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim ConditionElements: Set ConditionElements =
Client.QuerySourceConditions(ServerDescriptor, SourceDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ConditionElement: For Each ConditionElement In ConditionElements
```

```
    WScript.Echo "ConditionElements(" & ConditionElement.Name & "): " &
(UBound(ConditionElement.SubconditionNames) + 1) & " subcondition(s)"
Next
```

5.1.3.3.6 Querying for OPC A&E Event Attributes

The EasyAEClient.QueryCategoryAttributes method finds out event attributes that the server can provide as part of an event notification within a given event category.

5.1.3.4 Subscribing to Information (OPC A&E)

If your application needs to be informed about events occurring in the process and provided by the OPC Alarms and Events server, it can subscribe to them, and receive notifications.

OPC Data Client contains methods that allow you to subscribe to OPC events, change the subscription parameters, and unsubscribe.

5.1.3.4.1 Subscribing to OPC A&E Events

Subscription is initiated by calling the SubscribeEvents method. The component will call handlers for Notification event for each event that satisfies the filter criteria of the created subscription. Obviously, you first need to hook up event handler for that event, and in order to prevent event loss, you should do it before subscribing.

Events may be generated quite rapidly. Your application needs to specify the *notification rate*, which effectively tells the OPC Alarms and Events server that you do not need to receive event notifications any faster than that.

If you want to subscribe to particular set of OPC Events, call the SubscribeEvents method. You can pass in individual arguments for machine name, server class, and notification rate.

C#

```
// This example shows how to subscribe to events and display the event message with
each notification. It also shows how to
// unsubscribe afterwards.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    partial class SubscribeEvents
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
```

```
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
1000);

                Console.WriteLine("Processing event notifications for 1 minute...");
                Thread.Sleep(60 * 1000);

                client.UnsubscribeEvents(handle);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events and display the event message with each
notification. It also shows how to
' unsubscribe afterwards.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
    Partial Friend Class SubscribeEvents
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Processing event notifications for 1 minute...")
                Thread.Sleep(60 * 1000)

                client.UnsubscribeEvents(handle)
            End Using
        End Sub

        ' Notification event handler
```

```
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to events display the event message with each
notification. It also shows how to
// unsubscribe afterwards.

type
  TClientEventHandlers = class
    // Notification event handler
    procedure OnNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyAENotificationEventArgs);
  end;

procedure TClientEventHandlers.OnNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyAENotificationEventArgs);
begin
  if not eventArgs.Succeeded then
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
  if eventArgs.EventData <> nil then
      WriteLn(eventArgs.EventData.Message);
end;

class procedure SubscribeEvents.Main;
var
  Client: TEasyAEClient;
  ClientEventHandlers: TClientEventHandlers;
  Handle: Integer;
  ServerDescriptor: _ServerDescriptor;
  State: OleVariant;
  SubscriptionParameters: _AESubscriptionParameters;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitEventServer.2';

  // Instantiate the client object and hook events
  Client := TEasyAEClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers.Create;
  Client.OnNotification := ClientEventHandlers.OnNotification;

  WriteLn('Subscribing events...');
```

```
  SubscriptionParameters := CoAESubscriptionParameters.Create;
  SubscriptionParameters.NotificationRate := 1000;
  Handle := Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters, true,
State);

  WriteLn('Processing event notifications for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing events...');
  Client.UnsubscribeEvents(Handle);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to events display the event message with each
notification. It also shows how to
// unsubscribe afterwards.

class DEasyEAClientEvents {
    function Notification($Sender, $E)
    {
        if (!($E->Succeeded))
        {
            printf("*** Failure: %s\n", $E->ErrorMessageBrief);
            Exit();
        }

        if (!is_null($E->EventData))
        {
            print $E->EventData->Message;
            print "\n";
        }
    }
}

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitEventServer.2";

$Client = new COM("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient");
$Events = new DEasyEAClientEvents();
com_event_sink($Client, $Events, "DEasyEAClientEvents");

print "Subscribing events...\n";
$SubscriptionParameters = new
COM("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters");
$SubscriptionParameters->NotificationRate = 1000;
$handle = $Client->SubscribeEvents($ServerDescriptor, $SubscriptionParameters, TRUE,
NULL);

print "Processing event notifications for 1 minute...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);

print "Unsubscribing events...\n";
```

```
$Client->UnsubscribeEvents($handle);

print "Finished.\n";
```

VBScript

```
Rem This example shows how to subscribe to events display the event message with each
notification. It also shows how to
Rem unsubscribe afterwards.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing events..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Processing event notifications for 1 minute..."
WScript.Sleep 60*1000

WScript.Echo "Unsubscribing events..."
Client.UnsubscribeEvents handle

WScript.Echo "Finished."



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not e.Succeeded Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

Optionally, you can specify a subscription filter; it is a separate object of AESubscriptionFilterType type. Other optional parameters are attributes that should be returned in event notifications (separate set of attributes for each event category is needed), and the "active" and "refresh when active" flags. You can also pass in a State argument of any type. When any event notification is generated, the State argument is then passed to the Notification event handler in the EasyAENotificationEventArgs.Arguments.State object.

You can alternatively pass in a ServerDescriptor in place of machine name and server class arguments. You can also replace the individual notification rate, subscription filter, and returned attributes arguments by passing in an AESubscriptionParameters object.

The State argument is typically used to provide some sort of correlation between objects in your application, and the event notifications. For example, if you are programming an HMI application and you want the event handler to update the control that displays the event messages, you may want to set the State argument to the control object itself. When the event notification arrives, you simply update the control indicated by the State property of EasyAENotificationEventArgs, without having to look it up.

The "refresh when active" flag enables a functionality that is useful if you want to keep a "copy" of condition states (that primarily exist in the OPC server) in your application. When this flag is set, the component will automatically perform a subscription Refresh (see further below) after the connection is first time established, and also each time it is reestablished (after a connection loss). This way, the component assures that your code will get notifications that allow you to "reconstruct" the state of event conditions at any given moment.

Note: It is NOT an error to subscribe to the same set of events twice (or more times), even with precisely the same parameters. You will receive separate subscription handles, and with regard to your application, this situation will look no different from subscribing to different set of events.

5.1.3.4.2 Specifying event filters (OPC A&E)

Examples:

Filtering by category or categories:

C#

```
// This example shows how to filter the events by their category.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    partial class SubscribeEvents
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        public static void FilterByCategories()
        {
            var eventHandler = new
EasyAENotificationEventHandler(AEClient_Notification_FilterByCategories);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Categories = new long[] { 15531778 }
            };
            // You can also filter using event types, severity, areas, and sources.
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);
```

```
            // Allow time for initial refresh
            Thread.Sleep(5 * 1000);

            // Set some events to active state.
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", true);

            Thread.Sleep(10 * 1000);

            AEClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void AEClient_Notification_FilterByCategories(object sender,
EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            Console.WriteLine(e);
            if (!e.Succeeded)
                return;

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.CategoryId: {0}", eventData.CategoryId);
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to filter the events by their category.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Partial Friend Class SubscribeEvents

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()
```

```
        Public Shared Sub FilterByCategories()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification_FilterByCategories)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications...")
            Dim subscriptionFilter As New AESubscriptionFilter() With _
                {.Categories = New Long() {15531778}}
            ' You can also filter using event types, severity, areas, and sources.
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", Nothing, subscriptionFilter)

            ' Allow time for initial refresh
            Thread.Sleep(5 * 1000)

            ' Set some events to active state.
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True)

            Thread.Sleep(10 * 1000)

            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification_FilterByCategories(ByVal sender As
Object, ByVal e As EasyAENotificationEventArgs)
            Console.WriteLine()
            Console.WriteLine(e)
            If Not e.Succeeded Then
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            Dim eventData As AEEventData = e.EventData
            If e.EventData IsNot Nothing Then
                Console.WriteLine("Event.CategoryId: {0}", eventData.CategoryId)
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("Event.Message: {0}", eventData.Message)
                Console.WriteLine("Event.Active: {0}", eventData.Active)
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged)
            End If
        End Sub
    End Class
End Namespace
```

Filtering by source or sources:

C#

```
// This example shows how to set the filtering criteria to be used for the event
```

```
subscription.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._AESubscriptionFilter
{
    class Properties
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        public static void Main1()
        {
            var eventHandler = new
EasyAENotificationEventHandler(aeClient_Notification);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1",
"Simulation.ConditionState3" }
            };
            // You can also filter using event types, categories, severity, and areas.
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // Allow time for initial refresh
            Thread.Sleep(5 * 1000);

            // Set some events to active state.
            try
            {
                // The activation below will come from a source contained in a filter
and the notification will arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
                // The activation below will come from a source that is not contained
in a filter and the notification will not arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Thread.Sleep(10 * 1000);

            AEClient.UnsubscribeEvents(handle);
```

```
        }

        // Notification event handler
        static void aeClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to set the filtering criteria to be used for the event
subscription.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._AESubscriptionFilter

    Friend Class Properties

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Public Shared Sub Main1()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications...")
            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1", "Simulation.ConditionState3"}
            ' You can also filter using event types, categories, severity, and areas.
            Dim handle As Integer = AEClient.SubscribeEvents("",
```

```
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter)

            ' Allow time for initial refresh
            Thread.Sleep(5 * 1000)

            ' Set some events to active state.
            Try
                ' The activation below will come from a source contained in a filter
and the notification will arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
                ' The activation below will come from a source that is not contained in
a filter and the notification will not arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Thread.Sleep(10 * 1000)

            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            Console.WriteLine()
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Dim eventData As AEEventData = e.EventData
                Console.WriteLine("EventData.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("EventData.Message: {0}", eventData.Message)
                Console.WriteLine("EventData.Active: {0}", eventData.Active)
                Console.WriteLine("EventData.Acknowledged: {0}",
eventData.Acknowledged)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to set the filtering criteria to be used for the event
subscription.

Option Explicit
```

```
Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject AEClient, "AEClient_"

WScript.Echo "Processing event notifications..."
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"
Dim SubscriptionFilter: Set SubscriptionFilter =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionFilter")
Dim SourceDescriptor1: Set SourceDescriptor1 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor1.QualifiedName = "Simulation.ConditionState1"
Dim SourceDescriptor2: Set SourceDescriptor2 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor2.QualifiedName = "Simulation.ConditionState3"
SubscriptionFilter.Sources = Array(SourceDescriptor1, SourceDescriptor2)
Rem You can also filter using event types, categories, severity, and areas.
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.Filter = SubscriptionFilter
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = AEClient.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

Rem Allow time for initial refresh
WScript.Sleep 5*1000

WScript.Echo "Set some events to active state..."
On Error Resume Next
Rem The activation below will come from a source contained in a filter and the
notification will arrive.
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True
Rem The activation below will come from a source that is not contained in a filter and
the notification will not arrive.
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
AEClient.UnsubscribeEvents handle



Rem Notification event handler
Sub AEClient_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
```

```
    End If

    WScript.Echo
    WScript.Echo "Refresh: " & e.Refresh
    WScript.Echo "RefreshComplete: " & e.RefreshComplete

    If Not (e.EventData Is Nothing) Then
        With e.EventData
            WScript.Echo "EventData.QualifiedSourceName: " & .QualifiedSourceName
            WScript.Echo "EventData.Message: " & .Message
            WScript.Echo "EventData.Active: " & .Active
            WScript.Echo "EventData.Acknowledged: " & .Acknowledged
        End With
    End If
End Sub
```

5.1.3.4.3 Changing Existing Subscription (OPC A&E)

It is not necessary to unsubscribe and then subscribe again if you want to change parameters of existing subscription (such as its notification rate). Instead, change the parameters by calling the ChangeEventSubscription method, passing it the subscription handle, and the new parameters (notification rate, and optionally a filter and an "active" flag).

C#

```
// This example shows how to subscribe to events display the event message with each
notification. It also shows how to
// unsubscribe afterwards.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class ChangeEventSubscription
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
500);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
```

```
                Console.WriteLine("Changing subscription...");
                client.ChangeEventSubscription(handle, 5 * 1000);

                Console.WriteLine("Waiting for 50 seconds...");
                Thread.Sleep(50 * 1000);

                client.UnsubscribeEvents(handle);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events display the event message with each
notification. It also shows how to
' unsubscribe afterwards.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class ChangeEventSubscription
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 500)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Changing subscription...")
                client.ChangeEventSubscription(handle, 5 * 1000)

                Console.WriteLine("Waiting for 50 seconds...")
                Thread.Sleep(50 * 1000)
```

```
                client.UnsubscribeEvents(handle)
            End Using
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to change the notification rate of an existing subscription.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 500
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Changing subscription..."
Client.ChangeEventSubscription handle, 5*1000, SubscriptionParameters.Filter, True

WScript.Echo "Waiting for 50 seconds..."
WScript.Sleep 50*1000

Client.UnsubscribeEvents handle



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If
```

```
    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

5.1.3.4.4 Unsubscribing from OPC A&E Events

A single subscription

In This Topic

If you no longer want to receive event notifications, you need to unsubscribe from them. To unsubscribe from events that you have previously subscribed to, call the UnsubscribeEvents method, passing it the subscription handle.

C#

```
// This example shows how to unsubscribe from specific event notifications.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class UnsubscribeEvents
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
1000);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeEvents(handle);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Notification event handler
```

```
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to unsubscribe from specific event notifications.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class UnsubscribeEvents
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeEvents(handle)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
```

```
End Namespace
```

VBScript

```
Rem This example shows how to unsubscribe from specific event notifications.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeEvents handle

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

All subscriptions

You can also unsubscribe from all events you have previously subscribed to (on the same instance of EasyAEClient object) by calling the UnsubscribeAllEvents method.

C#

```
// This example shows how to unsubscribe from all event notifications.
```

```
using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class UnsubscribeAllEvents
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                Console.WriteLine("Subscribing...");
                client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllEvents();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to unsubscribe from all event notifications.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class UnsubscribeAllEvents
        Public Shared Sub Main1()
```

```
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Console.WriteLine("Subscribing...")
                client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeAllEvents()

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to unsubscribe from all event notifications.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Client.SubscribeEvents ServerDescriptor, SubscriptionParameters, True, Nothing

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllEvents
```

```
WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

If you are no longer using the parent EasyAEClient object, you should unsubscribe from the events, or dispose of the EasyAEClient object, which will do the same for you. Otherwise, the subscriptions will internally be still alive, and may cause problems related to COM reference counting.

5.1.3.4.5 Refreshing Condition States (OPC A&E)

Your application can obtain the current state of all conditions which are active, or which are inactive but unacknowledged, by requesting a "refresh" from an event subscription. The component will respond by sending the appropriate event notifications to the application, via the event handlers, for all conditions selected by the event subscription filter. When invoking the event handler, the component will indicate whether the invocation is for a refresh or is an original notification. Refresh and original event notifications will not be mixed in the same event notifications.

If you want to force a refresh, call the RefreshEventSubscription method, passing it the subscription handle.

C#

```
// This example shows how to for a refresh for all active conditions and inactive,
unacknowledged conditions.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class RefreshEventSubscription
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        public static void Main1()
        {
            var eventHandler = new
```

```
EasyAENotificationEventHandler(AEClient_Notification);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1",
"Simulation.ConditionState2", "Simulation.ConditionState3" }
            };
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // The component will perform auto-refresh at this point, give it time to
happen
            Console.WriteLine("Waiting for 10 seconds...");
            Thread.Sleep(10 * 1000);

            // Set some events to active state, which will cause them to appear in
refresh
            Console.WriteLine("Activating conditions and waiting for 10 seconds...");
            try
            {
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
            Thread.Sleep(10 * 1000);

            Console.WriteLine("Refreshing subscription and waiting for 10 seconds...");
            AEClient.RefreshEventSubscription(handle);
            Thread.Sleep(10 * 1000);

            AEClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void AEClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
```

```
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to for a refresh for all active conditions and inactive,
unacknowledged conditions.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class RefreshEventSubscription

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Public Shared Sub Main1()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications...")
            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1", "Simulation.ConditionState2",
"Simulation.ConditionState3"}
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter)

            ' The component will perform auto-refresh at this point, give it time to
happen
            Console.WriteLine("Waiting for 10 seconds...")
            Thread.Sleep(10 * 1000)

            ' Set some events to active state, which will cause them to appear in
refresh
            Console.WriteLine("Activating conditions and waiting for 10 seconds...")
            Try
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
```

```
opcException.GetBaseException().Message)
                Exit Sub
            End Try
            Thread.Sleep(10 * 1000)

            Console.WriteLine("Refreshing subscription and waiting for 10 seconds...")
            AEClient.RefreshEventSubscription(handle)
            Thread.Sleep(10 * 1000)

            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            Console.WriteLine()
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Dim eventData As AEEventData = e.EventData
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("Event.Message: {0}", eventData.Message)
                Console.WriteLine("Event.Active: {0}", eventData.Active)
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to for a refresh for all active conditions and inactive,
unacknowledged conditions.

Option Explicit

Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject AEClient, "AEClient_"

WScript.Echo "Processing event notifications..."
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"
Dim SourceDescriptor1: Set SourceDescriptor1 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor1.QualifiedName = "Simulation.ConditionState1"
Dim SourceDescriptor2: Set SourceDescriptor2 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor2.QualifiedName = "Simulation.ConditionState1"
```

```
Dim SourceDescriptor3: Set SourceDescriptor3 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor3.QualifiedName = "Simulation.ConditionState1"
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.Filter.Sources = Array(SourceDescriptor1, SourceDescriptor2,
SourceDescriptor3)
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = AEClient.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

Rem The component will perform auto-refresh at this point, give it time to happen
WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

Rem Set some events to active state, which will cause them to appear in refresh
On Error Resume Next
WScript.Echo "Activating conditions and waiting for 10 seconds..."
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Sleep 10*1000

WScript.Echo "Refreshing subscription and waiting for 10 seconds..."
AEClient.RefreshEventSubscription handle
WScript.Sleep 10*1000

AEClient.UnsubscribeEvents handle



Rem Notification event handler
Sub AEClient_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo
    WScript.Echo "Refresh: " & e.Refresh
    WScript.Echo "RefreshComplete: " & e.RefreshComplete

    If Not (e.EventData Is Nothing) Then
        With e.EventData
            WScript.Echo "EventData.QualifiedSourceName: " & .QualifiedSourceName
            WScript.Echo "EventData.Message: " & .Message
            WScript.Echo "EventData.Active: " & .Active
            WScript.Echo "EventData.Acknowledged: " & .Acknowledged
        End With
    End If
End Sub
```

5.1.3.4.6 OPC A&E Notification Event

When an OPC Alarms and Events server generates an event, the EasyAEClient object generates a Notification event. For subscription mechanism to be useful, you should hook one or more event handlers to this event.

To be more precise, the Notification event is actually generated in other cases, too - if there is any significant occurrence related to the event subscription. This can be for three reasons:

1. You receive the Notification when a successful connection (or re-connection) is made. In this case, the Exception and EventData properties of the event arguments are null references.
2. You receive the Notification when there is a problem with the event subscription, and it is disconnected. In this case, the Exception property contains information about the error. The EventData property is a null reference.
3. You receive one additional Notification after the component has sent you all notifications for the forced "refresh". In this case, the RefreshComplete property of the event arguments is set to 'true', and the Exception and EventData properties contain null references.

The notification for the Notification event contains an EasyAENotificationEventArgs argument. You will find all kind of relevant data in this object. Some properties in this object contain valid information under all circumstances. These properties are e.g. Arguments (including Arguments.State). Other properties, such as EventData, contain null references when there is no associated information for them. When the EventData property is not a null reference, it contains an AEEventData object describing the detail of the actual OPC event received from the OPC Alarms and Events server.

Before further processing, your code should always inspect the value of Exception property of the event arguments. If this property is not a null reference, there has been an error related to the event subscription, the Exception property contains information about the problem, and the EventData property does not contain a valid object.

If the Exception property is a null reference, the notification may be informing you about the fact that a "forced" refresh is complete (in this case, the RefreshComplete property is 'true'), or that an event subscription has been successfully connected or re-connected (in this case, the EventData property is a null reference). If none of the previous applies, the EventData property contains a valid AEEventData object with details about the actual OPC event generated by the OPC server.

Pseudo-code for the full **Notification** event handler may look similar to this:

> **if** notificationEventArgs.Exception is not null **then**
>
> *An error occurred and the subscription is disconnected, handle it (or ignore)*
>
> **else if** notificationEventArgs.RefreshComplete **then**
>
> *A "refresh" is complete; handle it (only needed if you are invoking a refresh explicitly)*
>
> **else if** notificationEventArgs.EventData is null **then**
>
> *Subscription has been successfully connected or re-connected, handle it (or ignore)*
>
> **else**
>
> *Handle the OPC event, details are in notificationEventArgs.EventData. You may use notificationEventArgs.Refresh flag for distinguishing refreshes from original notifications.*

The Notification event handler is called on a thread determined by the EasyAEClient component. For details, please refer to "Multithreading and Synchronization" chapter under "Advanced Topics".

C#

```
// This example shows how to subscribe to events with specified event attributes, and
obtain the attribute values in event
// notifications.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAENotificationEventArgs
{
    class AttributeValues
    {
        public static void Main1()
        {
            var aeClient = new EasyAEClient();
            var daClient = new EasyDAClient();

            var eventHandler = new
EasyAENotificationEventHandler(aeClient_Notification);
            aeClient.Notification += eventHandler;

            // Inactivate the event condition (we will later activate it and receive
the notification)
            daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Inactivate", true);

            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1" }
            };

            // Prepare a dictionary holding requested event attributes for each event
category
            // The event category IDs and event attribute IDs are hard-coded here, but
can be obtained from the OPC
            // server by querying as well.
            var returnedAttributesByCategory = new AEAttributeSetDictionary
            {
                [0x00ECFF02] = new long[] {0x00EB0003, 0x00EB0008}
            };

            Console.WriteLine("Subscribing to events...");
            int handle = aeClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter,
                returnedAttributesByCategory);

            // Give the refresh operation time to complete
            Thread.Sleep(5 * 1000);

            // Trigger an event carrying specified attributes (activate the condition)
```

```
            try
            {
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
                    "SimulateEvents.ConditionState1.AttributeValues.15400963", 123456);
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
                    "SimulateEvents.ConditionState1.AttributeValues.15400968", "Some
string value");
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Processing event notifications for 10 seconds...");
            Thread.Sleep(10 * 1000);

            aeClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void aeClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (!e.Refresh && (e.EventData != null))
            {
                // Display all received event attribute IDs and their corresponding
values
                Console.WriteLine("Event attribute count: {0}",
e.EventData.AttributeValues.Count);
                foreach (KeyValuePair<long, object> pair in
e.EventData.AttributeValues)
                    Console.WriteLine("    {0}: {1}", pair.Key, pair.Value);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events with specified event attributes, and
obtain the attribute values in event
' notifications.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel
```

```
Namespace DocExamples.AlarmsAndEvents._EasyAENotificationEventArgs
    Friend Class AttributeValues
        Public Shared Sub Main1()
            Dim aeClient = New EasyAEClient()
            Dim daClient = New EasyDAClient()

            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
aeClient_Notification)
            AddHandler aeClient.Notification, eventHandler

            ' Inactivate the event condition (we will later activate it and receive the
notification)
            daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Inactivate", True)

            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1"}

            ' Prepare a dictionary holding requested event attributes for each event
category
            ' The event category IDs and event attribute IDs are hard-coded here, but
can be obtained from the OPC
            ' server by querying as well.
            Dim returnedAttributesByCategory = New AEAttributeSetDictionary()
            returnedAttributesByCategory(&HECFF02) = New Long() {&HEB0003, &HEB0008}

            Console.WriteLine("Subscribing to events...")
            Dim handle As Integer = aeClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter,
returnedAttributesByCategory)

            ' Give the refresh operation time to complete
            Thread.Sleep(5 * 1000)

            ' Trigger an event carrying specified attributes (activate the condition)
            Try
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.AttributeValues.15400963", 123456)
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.AttributeValues.15400968", "Some string value")
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Processing event notifications for 10 seconds...")
            Thread.Sleep(10 * 1000)

            aeClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub aeClient_Notification(ByVal sender As Object, ByVal e As
```

```
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            If (Not e.Refresh) AndAlso (e.EventData IsNot Nothing) Then
                ' Display all received event attribute IDs and their corresponding
values
                Console.WriteLine("Event attribute count: {0}",
e.EventData.AttributeValues.Count)
                For Each pair As KeyValuePair(Of Long, Object) In
e.EventData.AttributeValues
                    Console.WriteLine("    {0}: {1}", pair.Key, pair.Value)
                Next pair
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example subscribe to events, and displays rich information available with each
event notification.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Processing event notifications for 1 minute..."
WScript.Sleep 60*1000

Client.UnsubscribeEvents handle



Rem Notification event handler
Sub Client_Notification(Sender, e)
    On Error Resume Next
    WScript.Echo
    WScript.Echo "e.Exception.Message: " & e.Exception.Message
    WScript.Echo "e.Exception.Source: " & e.Exception.Source
    WScript.Echo "e.Exception.ErrorCode: " & e.Exception.ErrorCode
    WScript.Echo "e.Arguments.State: " & e.Arguments.State
    WScript.Echo "e.Arguments.ServerDescriptor.MachineName: " &
```

```
e.Arguments.ServerDescriptor.MachineName
    WScript.Echo "e.Arguments.ServerDescriptor.ServerClass: " &
e.Arguments.ServerDescriptor.ServerClass
    WScript.Echo "e.Arguments.SubscriptionParameters.Active: " &
e.Arguments.SubscriptionParameters.Active
    WScript.Echo "e.Arguments.SubscriptionParameters.NotificationRate: " &
e.Arguments.SubscriptionParameters.NotificationRate
    Rem IMPROVE: Display Arguments.SubscriptionParameters.Filter details
    WScript.Echo "e.Arguments.SubscriptionParameters.Filter: " &
e.Arguments.SubscriptionParameters.Filter
    Rem IMPROVE: Display Arguments.SubscriptionParameters.ReturnedAttributesByCategory
details
    WScript.Echo "e.Arguments.SubscriptionParameters.ReturnedAttributesByCategory: " &
e.Arguments.SubscriptionParameters.ReturnedAttributesByCategory
    WScript.Echo "e.Refresh: " & e.Refresh
    WScript.Echo "e.RefreshComplete: " & e.RefreshComplete
    WScript.Echo "e.EnabledChanged: " & e.EnabledChanged
    WScript.Echo "e.ActiveChanged: " & e.ActiveChanged
    WScript.Echo "e.AcknowledgedChanged: " & e.AcknowledgedChanged
    WScript.Echo "e.QualityChanged: " & e.QualityChanged
    WScript.Echo "e.SeverityChanged: " & e.SeverityChanged
    WScript.Echo "e.SubconditionChanged: " & e.SubconditionChanged
    WScript.Echo "e.MessageChanged: " & e.MessageChanged
    WScript.Echo "e.AttributeChanged: " & e.AttributeChanged
    WScript.Echo "e.EventData.QualifiedSourceName: " & e.EventData.QualifiedSourceName
    WScript.Echo "e.EventData.Time: " & e.EventData.Time
    WScript.Echo "e.EventData.TimeLocal: " & e.EventData.TimeLocal
    WScript.Echo "e.EventData.Message: " & e.EventData.Message
    WScript.Echo "e.EventData.EventType: " & e.EventData.EventType
    WScript.Echo "e.EventData.CategoryId: " & e.EventData.CategoryId
    WScript.Echo "e.EventData.Severity: " & e.EventData.Severity
    Rem IMPROVE: Display EventData.AttributeValues details
    WScript.Echo "e.EventData.AttributeValues: " & e.EventData.AttributeValues
    WScript.Echo "e.EventData.ConditionName: " & e.EventData.ConditionName
    WScript.Echo "e.EventData.SubconditionName: " & e.EventData.SubconditionName
    WScript.Echo "e.EventData.Enabled: " & e.EventData.Enabled
    WScript.Echo "e.EventData.Active: " & e.EventData.Active
    WScript.Echo "e.EventData.Acknowledged: " & e.EventData.Acknowledged
    WScript.Echo "e.EventData.Quality: " & e.EventData.Quality
    WScript.Echo "e.EventData.AcknowledgeRequired: " & e.EventData.AcknowledgeRequired
    WScript.Echo "e.EventData.ActiveTime: " & e.EventData.ActiveTime
    WScript.Echo "e.EventData.ActiveTimeLocal: " & e.EventData.ActiveTimeLocal
    WScript.Echo "e.EventData.Cookie: " & e.EventData.Cookie
    WScript.Echo "e.EventData.ActorId: " & e.EventData.ActorId
End Sub
```

5.1.3.4.7 Using Callback Methods Instead of Event Handlers (OPC A&E)

The subscription methods also allow you to directly specify the callback method (delegate) to be invoked for each event notification you are subscribing to.

For detailed discussion on this subject, please refer to “Using Callback Methods Instead of Event Handlers” under the “OPC Data Access Tasks” chapter. All information presented there applies to OPC Alarms and Events as well.

C#

```
// This example shows how to subscribe to events and display the event message with
each notification, using a callback method
// specified using lambda expression.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    partial class SubscribeEvents
    {
        public static void CallbackLambda()
        {
            // Instantiate the client object
            var client = new EasyAEClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the event message
            client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
                (sender, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);
                    if (!eventArgs.Succeeded)
                    {
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                        return;
                    }
                    if (eventArgs.EventData != null)
                        Console.WriteLine(eventArgs.EventData.Message);
                });

            Console.WriteLine("Processing event notifications for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllEvents();

            Console.WriteLine("Waiting for 2 seconds...");
            Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events and display the event message with each
notification, using a callback method
' specified using lambda expression.

Imports OpcLabs.EasyOpc.AlarmsAndEvents

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
    Partial Friend Class SubscribeEvents
        Public Shared Sub CallbackLambda()
            ' Instantiate the client object
            Dim client = New EasyAEClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the event message
            client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
                    Sub(sender, eventArgs)
                        Debug.Assert(eventArgs IsNot Nothing)
                        If Not eventArgs.Succeeded Then
                            Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                            Exit Sub
                        End If
                        If eventArgs.EventData IsNot Nothing Then
                            Console.WriteLine(eventArgs.EventData.Message)
                        End If
                    End Sub)

            Console.WriteLine("Processing event notifications for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllEvents()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

5.1.3.5 Setting Parameters (OPC A&E)

While the most information needed to perform OPC tasks is contained in arguments to method calls, there are some component-wide parameters that are not worth repeating in every method call, and also some that have wider effect that influences more than just a single method call. You can obtain and modify these parameters through properties on the EasyAEClient object.

Following are instance properties, i.e. if you have created multiple EasyAEClient object, each will have its own copy of them:

- Mode: Allows you to influence how EasyOPC performs various operations on OPC Alarms and Events servers.
- HoldPeriods: Specifies optimization parameters that reduce the load on the OPC server.

Instance properties can be modified from your code.

In OPC Data Client.NET, if you have placed the EasyAEClient object on the designer surface, the instance properties can also be directly edited in the Properties window in Visual Studio.

In OPC Data Client-COM, your code can set different values to these properties if needed.

Following properties are static, i.e. shared among all instances of EasyAEClient object:

- EngineParameters: Contains global parameters such as frequencies of internal tasks performed by the component.
- MachineParameters: Contains parameters related to operations that target a specific computer but not a specific OPC server, such as browsing for OPC servers using various methods.
- ClientParameters: Contains parameters that influence operations that target a specific OPC server a whole.
- LinkParameters: Contains parameters that influence how EasyOPC works with live OPC event subscriptions.

Static properties can only be modified from your code.

Please use the Reference documentation for details on meaning of various properties and their use.

5.1.4 Imperative Programming Model for OPC UA Alarms & Conditions

This chapter gives you guidance in how to implement the common tasks that are needed when dealing with OPC UA Alarms & Conditions vents server from the client side. You achieve these tasks by calling methods on the EasyUAClient object (the same object as for OPC UA Data Access).

Note: Functionality in this chapter that has to do specifically with Alarms & Conditions is described in Part 9 of the OPC Unified Architecture Specification, however many related and relevant pieces are scattered around other parts of the specification as well.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

5.1.4.1 Obtaining Information (OPC UA Alarms & Conditions)

For obtaining information related to Alarms & Conditions (such as the state of conditions), OPC UA does not introduce any new mechanism over what we have already described for OPC Data (in OPC UA). Please refer to the “Obtaining Information” under “Imperative Programming Model for OPC Data (Class and UA)” for all necessary information.

Methods (and extension methods) such as EasyUAClient.Read, ReadMultiple, ReadValue, ReadMultipleValues can be used.

5.1.4.2 Modifying Information (OPC UA Alarms & Conditions)

Methods described in this chapter allow your application to modify information in the underlying data source that the OPC server connects to or in the OPC server itself (acknowledging conditions). It is assumed that your application already somehow knows how to identify the data it is interested in. If the location of the data is not known upfront, use methods described Browsing for Information chapter first.

The operations performed by the methods described here are actually implemented inside the OPC server. The OPC specification defines OPC UA methods for that, and as such, you could use e.g. one of the EasyUAClient.CallMethod

overloads in order to achieve the same outcome. You would have to, however, look up and specify the Node ID of the desired method, and properly assemble the array of input arguments (with proper types), and so on. The methods described below make it easier, and do the boring part for you. They are all implemented in a specialized client object for OPC UA Alarms and Conditions (EasyUAAlarmsAndConditionsClient Class), which you can derive from the IEasyUAClient, or instantiate standalone.

The easiest way to access the functionality is usually to call the AsAlarmsAndConditionsClient Method on the IEasyUAClient Interface you already have. See **Specialized Client Objects (Section 6.4)** for more details. For an example of how to do that, see **Examples - OPC UA Alarms&Conditions - Acknowledge an event (Section 13.2.6.1)**.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

5.1.4.2.1 Conditions - Disabling/Enabling, and Applying Comments

In order to change a condition instance to the Disabled state, use the IEasyUAAlarmsAndConditionsClient.Disable method.

In order to change a condition instance to the enabled state, use the IEasyUAAlarmsAndConditionsClient.Enable method.

In order to apply a comment to a specific state of a condition instance, use the IEasyUAAlarmsAndConditionsClient.AddComment method.

5.1.4.2.2 Dialogs - Responding

In order to respond to pass the selected response option and end the dialog, use the IEasyUAAlarmsAndConditionsClient.Respond method.

5.1.4.2.3 Acknowledgeable Conditions - Acknowledging and Confirmation

In order to acknowledge an event notification for a condition instance state (where AckedState is FALSE), use the IEasyUAAlarmsAndConditionsClient.Acknowledge extension method.

C#

```
// This example shows how to acknowledge an OPC UA event.

using System;
using System.Threading;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class Acknowledge
    {
```

```
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client objects
            var client = new EasyUAClient();
            IEasyUAAlarmsAndConditionsClient alarmsAndConditionsClient =
client.AsAlarmsAndConditionsClient();

            UANodeId nodeId = null;
            byte[] eventId = null;
            var anEvent = new ManualResetEvent(initialState: false);

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                new UAEventFilterBuilder(
                    UAFilterElements.Equals(
                    UABaseEventObject.Operands.NodeId,
                        new UANodeId(expandedText:
"nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=1:Colours/EastTank?Yellow")),
                    UABaseEventObject.AllFields),
                (sender, eventArgs) =>
                {
                    if (!eventArgs.Succeeded)
                    {
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief);
                        return;
                    }
                    if (eventArgs.EventData != null)
                    {
                        UABaseEventObject baseEventObject = eventArgs.EventData.BaseEvent;
                        Console.WriteLine(baseEventObject);

                        // Make sure we do not catch the event more than once
                        if (anEvent.WaitOne(0))
                            return;

                        nodeId = baseEventObject.NodeId;
                        eventId = baseEventObject.EventId;

                        anEvent.Set();
                    }
                },
                state:null);

            Console.WriteLine("Waiting for an event for 30 seconds...");
            if (!anEvent.WaitOne(30*1000))
            {
                Console.WriteLine("Event not received.");
                return;
            }

            Console.WriteLine("Acknowledging an event...");
            try
            {
                alarmsAndConditionsClient.Acknowledge(
                    endpointDescriptor,
                    nodeId,
                    eventId,
                    "Acknowledged by an automated example code.");
            }
            catch (UAException uaException)
            {
```

```
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
            }

            Console.WriteLine("Waiting for 5 seconds...");
            Thread.Sleep(5 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            Thread.Sleep(5 * 1000);
        }



        // Example output:
        //Subscribing...
        //Waiting for an event for 30 seconds...
        //[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
        //Acknowledging an event...
        //Waiting for 5 seconds...
        //[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
        //Unsubscribing...
        //Waiting for 5 seconds...
    }
}
```

Object Pascal

```
// This example shows how to acknowledge an OPC UA event.

type
  THelperMethods11 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_AllFields: _UAAttributeFieldCollection; static;
  end;

type
  TClientEventHandlers11 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
    private
      FAnEvent: Boolean;
      FEventId: OleVariant;
      FNodeId: _UANodeId;
    published
      property AnEvent: Boolean read FAnEvent write FAnEvent;
      property EventId: OleVariant read FEventId;
      property NodeId: _UANodeId read FNodeId;
  end;

procedure TClientEventHandlers11.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
```

```
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  BaseEventObject: _UABaseEventObject;
begin
  if not eventArgs.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
    Exit;
  end;

  if eventArgs.EventData <> nil then
  begin
    BaseEventObject := eventArgs.EventData.BaseEvent;
      WriteLn(BaseEventObject.ToString);

    // Make sure we do not catch the event more than once
    if FAnEvent then
      Exit;

    FNodeId := BaseEventObject.NodeId;
    TVarData(FEventId).VType := varArray or varVariant;
    TVarData(FEventId).VArray := PVarArray(BaseEventObject.EventId);
    FAnEvent := True;
  end;
end;

class procedure Acknowledge.Main;
var
  AlarmsAndConditionsClient: _EasyUAAlarmsAndConditionsClient;
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers11;
  EndpointDescriptor: string;
  EndTime: Cardinal;
  EventFilter: _UAEventFilter;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  Operand1: _UASimpleAttributeOperand;
  Operand2: _UALiteralOperand;
  NodeDescriptor: _UANodeDescriptor;
  NodeId: _UANodeId;
  ServerNodeId: _UANodeID;
  WhereClause: _UAContentFilterElement;
begin
  // Define which server we will work with.
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Event filter: Events with specific node ID.
  Operand1 := THelperMethods11.UABaseEventObject_Operands_NodeId;
  NodeId := CoUANodeId.Create;
  NodeId.ExpandedText := 'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Yellow';
  Operand2 := CoUALiteralOperand.Create;
  Operand2.Value := NodeId;
  WhereClause := CoUAContentFilterElement.Create;
  WhereClause.FilterOperator := UAFilterOperator_Equals;
  WhereClause.FilterOperands.Add(Operand1);
  WhereClause.FilterOperands.Add(Operand2);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := THelperMethods11.UABaseEventObject_AllFields;
  EventFilter.WhereClause := WhereClause;

  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.EventFilter := EventFilter;
```

```
  MonitoringParameters.QueueSize := 1000;
  MonitoringParameters.SamplingInterval := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;
  MonitoredItemArguments.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments.NodeDescriptor.NodeId := ServerNodeId;

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers11.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;
  AlarmsAndConditionsClient := Client.AsAlarmsAndConditionsClient;
  ClientEventHandlers.AnEvent := False;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  WriteLn('Subscribing...');
  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Waiting for an event for 30 seconds...');
  EndTime := Ticks + 30*1000;
  repeat
      if ClientEventHandlers.AnEvent or (EndTime < Ticks) then
        Break;
      PumpSleep(1000);
  until False;

  if ClientEventHandlers.AnEvent then
  begin
    WriteLn('Acknowledging an event...');
    try
      NodeDescriptor := CoUANodeDescriptor.Create;
      NodeDescriptor.NodeId := ClientEventHandlers.NodeId;
      AlarmsAndConditionsClient.Acknowledge(
       MonitoredItemArguments.EndpointDescriptor,
       NodeDescriptor,
       ClientEventHandlers.EventId,
       'Acknowledged by an automated example code.');
    except
      on E: EOleException do
      begin
        WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
        Exit;
      end;
    end;
  end
  else
  begin
    WriteLn('Event not received.');
  end;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

```
// Example output:
// Subscribing...
// Waiting for an event for 30 seconds...
// [EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
// Acknowledging an event...
// Waiting for 5 seconds...
// [EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
// Unsubscribing...
// Waiting for 5 seconds...


class function THelperMethods11.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods11.UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames :=
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods11.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods11.UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
end;

class function THelperMethods11.UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
end;

class function THelperMethods11.UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods11.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods11.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;
```

```
class function THelperMethods11.UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
end;

class function THelperMethods11.UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
end;

class function THelperMethods11.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods11.UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;

class function THelperMethods11.UABaseEventObject_AllFields: _UAAttributeFieldCollection;
var
  Fields: _UAAttributeFieldCollection;
begin
  Fields := CoUAAttributeFieldCollection.Create;
  Fields.Add(UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventType.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Time.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_LocalTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Message.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Severity.ToUAAttributeField);

  Result := Fields;
end;
```

PHP

```
// This example shows how to acknowledge an OPC UA event.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

const UAFilterOperator_Equals = 1;

class ClientEvents {
    private $AnEvent = false;
    private $EventId;
    private $NodeId;

    function EventNotification($Sender, $E)
    {
        if (!$E->Succeeded) {
            printf(" *** Failure: %s\n", $E->ErrorMessageBrief);
            return;
        }

        if (!is_null($E->EventData)) {
            $BaseEventObject = $E->EventData->BaseEvent;
            printf("%s\n", $BaseEventObject);

            // Make sure we do not catch the event more than once
           if ($this->AnEvent)
               return;
```

```
            $this->NodeId = $BaseEventObject->NodeId;
            $this->EventId = $BaseEventObject->EventId;
            $this->AnEvent = true;
        }
    }

    public function getAnEvent() {
        return $this->AnEvent;
    }

    public function getEventId() {
        return $this->EventId;
    }

    public function getNodeId() {
        return $this->NodeId;
    }
}


// Define which server we will work with.
$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Event filter: Events with specific node ID.
$Operand1 = UABaseEventObject_Operands_NodeId();
$NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId->ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Yellow";
$Operand2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand2->Value = $NodeId;
$WhereClause = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$WhereClause->FilterOperator = UAFilterOperator_Equals;
$WhereClause->FilterOperands->Add($Operand1);
$WhereClause->FilterOperands->Add($Operand2);

$EventFilter = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter->SelectClauses = UABaseEventObject_AllFields();
$EventFilter->WhereClause = $WhereClause;

$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";

$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->EventFilter = $EventFilter;
$MonitoringParameters->QueueSize = 1000;
$MonitoringParameters->SamplingInterval = 1000;

$MonitoredItemArguments = new COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments->AttributeId = UAAttributeId_EventNotifier;
$MonitoredItemArguments->EndpointDescriptor = $EndpointDescriptor;
$MonitoredItemArguments->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments->NodeDescriptor->NodeId = $ServerNodeId;

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");
$AlarmsAndConditionsClient = $Client->AsAlarmsAndConditionsClient;

$arguments[0] = $MonitoredItemArguments;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Waiting for an event for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while ((time() < $startTime + 30) and !
($ClientEvents->getAnEvent()));
```

```
if ($ClientEvents->getAnEvent()) {
    printf("Acknowledging an event...\n");
    try
    {
        $NodeDescriptor = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
        $NodeDescriptor->NodeId = $ClientEvents->getNodeId();
        $AlarmsAndConditionsClient->Acknowledge(
           $EndpointDescriptor,
           $NodeDescriptor,
           $ClientEvents->getEventId(),
           "Acknowledged by an automated example code.");
    }
    catch (com_exception $e)
    {
          printf("Failure: %s\n", $e->getMessage());
    }
}
else {
    printf("Event not received.\n");
}

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_EventId() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventId");
}

function UABaseEventObject_Operands_EventType() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventType");
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
```

```
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}

function UABaseEventObject_Operands_ReceiveTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/ReceiveTime");
}

function UABaseEventObject_Operands_LocalTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/LocalTime");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}


function UABaseEventObject_AllFields() {
    $Fields = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
    $Fields->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventType()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_ReceiveTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_LocalTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField);

    return $Fields;
}
```

VB.NET

```
' This example shows how to obtain acknowledge an event.

Imports System
Imports System.Threading
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions.Extensions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class Acknowledge
        Public Shared Sub Main1()

            ' Instantiate the client object
            Dim client = New EasyUAClient()
            Dim alarmsAndConditionsClient As IEasyUAAlarmsAndConditionsClient =
client.AsAlarmsAndConditionsClient()

            Dim nodeId As UANodeId = Nothing
            Dim eventId As Byte() = Nothing
            Dim anEvent = New ManualResetEvent(initialState:=False)

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent(
```

```
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
                UAObjectIds.Server,
                1000,
                New UAEventFilterBuilder(
                    UAFilterElements.Equals(
                        UABaseEventObject.Operands.NodeId,
                        New
UANodeId(expandedText:="nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Colours/EastTank?
Yellow")),
                        UABaseEventObject.AllFields),
                Sub(sender, eventArgs)
                    If Not eventArgs.Succeeded Then
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief)
                        Return
                    End If
                    If eventArgs.EventData IsNot Nothing Then
                        Dim baseEventObject = eventArgs.EventData.BaseEvent
                        Console.WriteLine(baseEventObject)

                        ' Make sure we do not catch the event more than once
                        If anEvent.WaitOne(0) Then
                            Return
                        End If

                        nodeId = baseEventObject.NodeId
                        eventId = baseEventObject.EventId

                        anEvent.Set()
                    End If
                End Sub,
                state:=Nothing)

            Console.WriteLine("Waiting for an event for 30 seconds...")
            If Not anEvent.WaitOne(30 * 1000) Then
                Console.WriteLine("Event not received")
                Return
            End If

            Console.WriteLine("Acknowledging an event...")
            Try
                alarmsAndConditionsClient.Acknowledge(
                    "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
                    nodeId,
                    eventId,
                    "Acknowledged by an automated example code")
            Catch uaException As UAException
                Console.WriteLine("Failure: {0}", uaException.GetBaseException().Message)
            End Try

            Console.WriteLine("Waiting for 5 seconds...")
            Thread.Sleep(5 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Thread.Sleep(5 * 1000)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to acknowledge an OPC UA event.

Option Explicit

Const UAAttributeId_NodeId = 1
```

```
Const UAAttributeId_EventNotifier = 12

Const UAFilterOperator_Equals = 1

' Define which server we will work with.
Dim EndpointDescriptor: Set EndpointDescriptor = CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
EndpointDescriptor.UrlString = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer"

' Instantiate the client objects and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"
Dim AlarmsAndConditionsClient: Set AlarmsAndConditionsClient = Client.AsAlarmsAndConditionsClient

'
Dim NodeId
Dim EventId
Dim anEvent: anEvent = False ' Some tools have event objects, but VBScript doesn't, we will use a boolean
flag instead.

' Prepare arguments
Dim arguments(0)
Set arguments(0) = CreateMonitoredItemArguments

WScript.Echo "Subscribing..."
Client.SubscribeMultipleMonitoredItems arguments

WScript.Echo "Waiting for an event for 30 seconds..."
Dim endTime: endTime = Now() + 30*(1/24/60/60)
While (Not anEvent) And (Now() < endTime)
    WScript.Sleep 1000
WEnd
If Not anEvent Then
    WScript.Echo "Event not received."
    WScript.Quit
End If

WScript.Echo "Acknowledging an event..."
Dim NodeDescriptor: Set NodeDescriptor = CreateObject("OpcLabs.EasyOpc.UA.UANodeDescriptor")
Set NodeDescriptor.NodeId = NodeId
On Error Resume Next
AlarmsAndConditionsClient.Acknowledge EndpointDescriptor, NodeDescriptor, EventId, "Acknowledged by an
automated example code."
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
End If
On Error Goto 0

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Function ObjectTypeIds_BaseEventType
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.StandardName = "BaseEventType"
    Set ObjectTypeIds_BaseEventType = NodeId
End Function

Function UAFilterElements_SimpleAttribute(TypeId, simpleRelativeBrowsePathString)
    Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
    Dim Operand: Set Operand = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
```

```
    Set Operand.TypeId.NodeId = TypeId
    Set Operand.QualifiedNames =
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection
    Set UAFilterElements_SimpleAttribute = Operand
End Function

Function UABaseEventObject_Operands_NodeId
    Dim Operand: Set Operand = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Operand.TypeId.NodeId.StandardName = "BaseEventType"
    Operand.AttributeId = UAAttributeId_NodeId
    Set UABaseEventObject_Operands_NodeId = Operand
End Function

Function UABaseEventObject_Operands_EventId
    Set UABaseEventObject_Operands_EventId =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventId")
End Function

Function UABaseEventObject_Operands_EventType
    Set UABaseEventObject_Operands_EventType =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventType")
End Function

Function UABaseEventObject_Operands_SourceNode
    Set UABaseEventObject_Operands_SourceNode =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceNode")
End Function

Function UABaseEventObject_Operands_SourceName
    Set UABaseEventObject_Operands_SourceName =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceName")
End Function

Function UABaseEventObject_Operands_Time
    Set UABaseEventObject_Operands_Time = UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
"/Time")
End Function

Function UABaseEventObject_Operands_ReceiveTime
    Set UABaseEventObject_Operands_ReceiveTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/ReceiveTime")
End Function

Function UABaseEventObject_Operands_LocalTime
    Set UABaseEventObject_Operands_LocalTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/LocalTime")
End Function

Function UABaseEventObject_Operands_Message
    Set UABaseEventObject_Operands_Message =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Message")
End Function

Function UABaseEventObject_Operands_Severity
    Set UABaseEventObject_Operands_Severity =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Severity")
End Function

Function UABaseEventObject_AllFields
    Dim Fields: Set Fields = CreateObject("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection")

    Fields.Add UABaseEventObject_Operands_NodeId.ToUAAttributeField

    Fields.Add UABaseEventObject_Operands_EventId.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_EventType.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceNode.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceName.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Time.ToUAAttributeField
```

```
    Fields.Add UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_LocalTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Message.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Severity.ToUAAttributeField

    Set UABaseEventObject_AllFields = Fields
End Function

Function CreateMonitoredItemArguments
    ' Event filter: Events with specific node ID.
    Dim Operand1: Set Operand1 = UABaseEventObject_Operands_NodeId
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Yellow"
    Dim Operand2: Set Operand2 = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand")
    Set Operand2.Value = NodeId
    Dim WhereClause: Set WhereClause =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement")
    WhereClause.FilterOperator = UAFilterOperator_Equals
    WhereClause.FilterOperands.Add Operand1
    WhereClause.FilterOperands.Add Operand2

    Dim EventFilter: Set EventFilter = CreateObject("OpcLabs.EasyOpc.UA.UAEventFilter")
    Set EventFilter.SelectClauses = UABaseEventObject_AllFields
    Set EventFilter.WhereClause = WhereClause

    Dim ServerNodeId: Set ServerNodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    ServerNodeId.StandardName = "Server"

    Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
    Set MonitoringParameters.EventFilter = EventFilter
    MonitoringParameters.QueueSize = 1000
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments: Set MonitoredItemArguments =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
    MonitoredItemArguments.AttributeId = UAAttributeId_EventNotifier
    MonitoredItemArguments.EndpointDescriptor = EndpointDescriptor
    Set MonitoredItemArguments.MonitoringParameters = MonitoringParameters
    Set MonitoredItemArguments.NodeDescriptor.NodeId = ServerNodeId

    Set CreateMonitoredItemArguments = MonitoredItemArguments
End Function

Sub Client_EventNotification(Sender, EventArgs)
    If Not EventArgs.Succeeded Then
        WScript.Echo "*** Failure: " & EventArgs.ErrorMessageBrief
        Exit Sub
    End If

    If Not (EventArgs.EventData Is Nothing) Then
        Dim BaseEventObject: Set BaseEventObject = EventArgs.EventData.BaseEvent
        WScript.Echo BaseEventObject

        ' Make sure we do not catch the event more than once
        If anEvent Then
            Exit Sub
        End If

        Set NodeId = BaseEventObject.NodeId
        EventId = BaseEventObject.EventId

        anEvent = True
    End If
End Sub
```

```
' Example output:
'Subscribing...
'Waiting for an event for 30 seconds...
'[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
'Acknowledging an event...
'Waiting for 5 seconds...
'[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
'Unsubscribing...
'Waiting for 5 seconds...
```

In order to confirm an event notification for a condition instance state (where ConfirmedState is FALSE), use the IEasyUAAlarmsAndConditionsClient.Confirm method.

5.1.4.2.4 Alarms - Shelving and Unshelving

In order to set the alarm condition to the Unshelved state, use the IEasyUAAlarmsAndConditionsClient.Unshelve extension method.

In order to set the alarm condition to the TimedShelved state, use the IEasyUAAlarmsAndConditionsClient.TimedShelve extension method.

In order to set the alarm condition to the OneShotShelved state, use the IEasyUAAlarmsAndConditionsClient.OneShotShelve method.

5.1.4.3 Browsing for Information (OPC UA Alarms & Conditions)

OPC Data Client contains methods that allow your application to retrieve and enumerate information about OPC UA Alarm & Conditions servers that exist on the network, and data available within these servers. Your code can then make use of the information obtained, e.g. to accommodate to configuration changes dynamically.

The methods we are describing here are for programmatic browsing, with no user interface (or when the user interface is provided by our own code).

5.1.4.3.1 Discovering OPC UA Servers (OPC UA Alarms & Conditions)

Please refer to the “Discovering OPC UA Servers” chapter under “Imperative Programming Model for OPC Data (Class and UA)” for all necessary information. OPC Unified Architecture, as the name suggests, allows to treat multiple functionality areas commonly, and the same OPC Servers that provide Data Access functionality are (or can be) also Alarms & Conditions servers.

5.1.4.3.2 Browsing for OPC UA Event Sources

In order to browse for event sources, you can use the pre-made browse parameters available in

the UABrowseParameters.EventSources static property, and call the IEasyUAClient.Browse Method with them. Alternatively, you can use a dedicated extension method, IEasyUAClient.BrowseEventSources, to perform browsing for event sources easily.

C#

```
// This example shows how to browse objects under the "Objects" node and display event
sources.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class BrowseEventSources
    {
        public static void Overload2()
        {
            // Start browsing from the "Objects" node
            try
            {
                BrowseFrom(UAObjectIds.ObjectsFolder);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }

        private static void BrowseFrom(UANodeDescriptor nodeDescriptor)
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine("Parent node: {0}", nodeDescriptor);

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain event sources
            UANodeElementCollection eventSourceNodeElementCollection =
client.BrowseEventSources(
                endpointDescriptor, nodeDescriptor);

            // Display event sources
            if (eventSourceNodeElementCollection.Count != 0)
            {
                Console.WriteLine();
                Console.WriteLine("Event sources:");
                foreach (UANodeElement eventSourceNodeElement in
```

```
eventSourceNodeElementCollection)
                    Console.WriteLine(eventSourceNodeElement);
            }

            // Obtain objects
            UANodeElementCollection objectNodeElementCollection = client.BrowseObjects(
                endpointDescriptor, nodeDescriptor);

            // Recurse
            foreach (UANodeElement objectNodeElement in objectNodeElementCollection)
                BrowseFrom(objectNodeElement);
        }

        // Example output (truncated):
        //
        //
        //Parent node: ObjectsFolder
        //
        //
        //Parent node: Server
        //
        //Event sources:
        //Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Green (Object)
        //Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Yellow (Object)
        //
        //
        //Parent node: Server_ServerCapabilities
        //...
    }
}
```

Object Pascal

```
// This example shows how to browse objects under the "Objects" node and display event
sources.

class procedure BrowseEventSources.Main;
var
  ObjectNodeId: _UANodeId;
begin
  ObjectNodeId := CoUANodeId.Create;
  ObjectNodeId.StandardName := 'Objects';
  try
    BrowseFrom(ObjectNodeId);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
end;

class procedure BrowseEventSources.BrowseFrom(NodeId: _UANodeId);
var
```

```
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  EventSourceNodeElement: _UANodeElement;
  EventSourceNodeElementEnumerator: IEnumVariant;
  EventSourceNodeElements: _UANodeElementCollection;
  ObjectNodeElement: _UANodeElement;
  ObjectNodeElementEnumerator: IEnumVariant;
  ObjectNodeElements: _UANodeElementCollection;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  WriteLn;
  WriteLn;
  WriteLn('Parent node: ', NodeId.ToString);

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain event sources
  EventSourceNodeElements := Client.BrowseEventSources(EndpointDescriptor,
NodeId.ExpandedText);

  // Display event sources
  if EventSourceNodeElements.Count <> 0 then
  begin
    WriteLn;
    WriteLn('Event sources:');
    EventSourceNodeElementEnumerator := EventSourceNodeElements.GetEnumerator;
    while (EventSourceNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
    begin
      EventSourceNodeElement := IUnknown(Element) as _UANodeElement;
      WriteLn(EventSourceNodeElement.ToString);
    end;
  end;

  // Obtain objects
  ObjectNodeElements := Client.BrowseObjects(EndpointDescriptor, NodeId.ExpandedText);

  // Recurse
  ObjectNodeElementEnumerator := ObjectNodeElements.GetEnumerator;
  while (ObjectNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    ObjectNodeElement := IUnknown(Element) as _UANodeElement;
    BrowseFrom(ObjectNodeElement.NodeId);
  end;
end;
```

PHP

```
// This example shows how to browse objects under the "Objects" node and display event
sources.


// Start browsing from the "Objects" node
```

```
$ObjectNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ObjectNodeId->StandardName = "Objects";
try
{
    BrowseFrom($ObjectNodeId);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

function BrowseFrom($NodeId) {
    $EndpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer";

    printf("\n");
    printf("\n");
    printf("Parent node: %s\n", $NodeId);

    // Instantiate the client object
    $Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

    // Obtain event sources
    $EventSourceNodeElements = $Client->BrowseEventSources($EndpointDescriptor,
$NodeId->ExpandedText);

    // Display event sources
    if ($EventSourceNodeElements->Count != 0) {
        printf("\n");
        printf("Event sources:\n");
        foreach ($EventSourceNodeElements as $EventSourceNodeElement)
        {
             printf("%s\n", $EventSourceNodeElement);
        }
     }

    // Obtain objects
    $ObjectNodeElements = $Client->BrowseObjects($EndpointDescriptor, $NodeId-
>ExpandedText);

    // Recurse
    foreach ($ObjectNodeElements as $ObjectNodeElement)
    {
         BrowseFrom($ObjectNodeElement->NodeId);
    }
}

// Example output (truncated):
//
//
//Parent node: ObjectsFolder
//
//
//Parent node: Server
//
//Event sources:
```

```
//Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
//Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
//
//
//Parent node: Server_ServerCapabilities
//...
```

VB.NET

```
' This example shows how to browse objects under the "Objects" node and display event
sources.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class BrowseEventSources
        Public Shared Sub Overload2()

            ' Start browsing from the "Objects" node
            Try
                BrowseFrom(UAObjectIds.ObjectsFolder)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try
        End Sub

        Private Shared Sub BrowseFrom(nodeDescriptor As UANodeDescriptor)

            Console.WriteLine()
            Console.WriteLine()
            Console.WriteLine("Parent node: {0}", nodeDescriptor)

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain notifiers
            Dim eventSourceNodeElementCollection As UANodeElementCollection =
client.BrowseEventSources( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Display notifiers
            Console.WriteLine()
            Console.WriteLine("Event sources:")
            For Each eventSourceNodeElement As UANodeElement In
eventSourceNodeElementCollection
                Console.WriteLine(eventSourceNodeElement)
            Next eventSourceNodeElement
```

```
            ' Obtain objects
            Dim objectNodeElementCollection = client.BrowseObjects( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Recurse
            For Each objectNodeElement As UANodeElement In objectNodeElementCollection
                BrowseFrom(objectNodeElement)
            Next objectNodeElement
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to browse objects under the "Objects" node and display event
sources.

Option Explicit

' Start browsing from the "Objects" node
Dim ObjectsNodeId: Set ObjectsNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ObjectsNodeId.StandardName = "Objects"
On Error Resume Next
BrowseFrom ObjectsNodeId
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0



Sub BrowseFrom(NodeId)
    Dim endpointDescriptor
    endpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"

    WScript.Echo
    WScript.Echo
    WScript.Echo "Parent node: " & NodeId

    ' Instantiate the client object
    Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

    ' Obtain event sources
    Dim EventSourceNodeElementCollection: Set EventSourceNodeElementCollection =
Client.BrowseEventSources( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Display event sources
    If EventSourceNodeElementCollection.Count <> 0 Then
        WScript.Echo
        WScript.Echo "Event sources:"
        Dim EventSourceNodeElement: For Each EventSourceNodeElement In
```

```
EventSourceNodeElementCollection
            WScript.Echo EventSourceNodeElement
        Next
    End If

    ' Obtain objects
    Dim ObjectNodeElementCollection: Set ObjectNodeElementCollection =
Client.BrowseObjects( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Recurse
    Dim ObjectNodeElement: For Each ObjectNodeElement In ObjectNodeElementCollection
        BrowseFrom ObjectNodeElement.NodeId
    Next
End Sub



' Example output (truncated):
'
'
'Parent node: ObjectsFolder
'
'
'Parent node: Server
'
'Event sources:
'Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
'Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
'
'
'Parent node: Server_ServerCapabilities
'...
```

5.1.4.3.3 Browsing for OPC UA Notifiers

In order to browse for notifiers, you can use pre-made browse parameters available in the UABrowseParameters.Notifiers static property, and call the IEasyUAClient.Browse method. Alternatively, you can use a dedicated extension method, IEasyUAClient.BrowseNotifiers, to perform browsing for notifiers easily.

C#

```
// This example shows how to browse objects under the "Objects" node and display
notifiers.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples.AlarmsAndConditions
{
    class BrowseNotifiers
    {
        public static void Overload2()
        {
            // Start browsing from the "Objects" node
            try
            {
                BrowseFrom(UAObjectIds.ObjectsFolder);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }

        private static void BrowseFrom(UANodeDescriptor nodeDescriptor)
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine("Parent node: {0}", nodeDescriptor);

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain notifiers
            UANodeElementCollection notifierNodeElementCollection =
client.BrowseNotifiers(
                endpointDescriptor, nodeDescriptor);

            // Display notifiers
            if (notifierNodeElementCollection.Count != 0)
            {
                Console.WriteLine();
                Console.WriteLine("Notifiers:");
                foreach (UANodeElement notifierNodeElement in
notifierNodeElementCollection)
                    Console.WriteLine(notifierNodeElement);
            }

            // Obtain objects
            UANodeElementCollection objectNodeElementCollection = client.BrowseObjects(
                endpointDescriptor, nodeDescriptor);

            // Recurse
            foreach (UANodeElement objectNodeElement in objectNodeElementCollection)
                BrowseFrom(objectNodeElement);
        }

        // Example output (truncated):
        //
        //
```

```
        //Parent node: ObjectsFolder
        //
        //
        //Parent node: Server
        //
        //Notifiers:
        //Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Green (Object)
        //Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Yellow (Object)
        //
        //
        //Parent node: Server_ServerCapabilities
        //...
    }
}
```

Object Pascal

```
// This example shows how to browse objects under the "Objects" node and display
notifiers.

class procedure BrowseNotifiers.Main;
var
  ObjectNodeId: _UANodeId;
begin
  ObjectNodeId := CoUANodeId.Create;
  ObjectNodeId.StandardName := 'Objects';
  try
    BrowseFrom(ObjectNodeId);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
end;

class procedure BrowseNotifiers.BrowseFrom(NodeId: _UANodeId);
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NotifierNodeElement: _UANodeElement;
  NotifierNodeElementEnumerator: IEnumVariant;
  NotifierNodeElements: _UANodeElementCollection;
  ObjectNodeElement: _UANodeElement;
  ObjectNodeElementEnumerator: IEnumVariant;
  ObjectNodeElements: _UANodeElementCollection;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  WriteLn;
  WriteLn;
```

```
  WriteLn('Parent node: ', NodeId.ToString);

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain notifiers
  NotifierNodeElements := Client.BrowseNotifiers(EndpointDescriptor,
NodeId.ExpandedText);

  // Display notifires
  if NotifierNodeElements.Count <> 0 then
  begin
    WriteLn;
    WriteLn('Notifiers:');
    NotifierNodeElementEnumerator := NotifierNodeElements.GetEnumerator;
    while (NotifierNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
    begin
      NotifierNodeElement := IUnknown(Element) as _UANodeElement;
      WriteLn(NotifierNodeElement.ToString);
    end;
  end;

  // Obtain objects
  ObjectNodeElements := Client.BrowseObjects(EndpointDescriptor, NodeId.ExpandedText);

  // Recurse
  ObjectNodeElementEnumerator := ObjectNodeElements.GetEnumerator;
  while (ObjectNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    ObjectNodeElement := IUnknown(Element) as _UANodeElement;
    BrowseFrom(ObjectNodeElement.NodeId);
  end;
end;

// Example output (truncated):
//
//
//Parent node: ObjectsFolder
//
//
//Parent node: Server
//
//Notifiers:
//Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
//Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
//
//
//Parent node: Server_ServerCapabilities
//...
```

PHP

```
// This example shows how to browse objects under the "Objects" node and display
notifiers.
```

```
// Start browsing from the "Objects" node
$ObjectNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ObjectNodeId->StandardName = "Objects";
try
{
    BrowseFrom($ObjectNodeId);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

function BrowseFrom($NodeId) {
    $EndpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer";

    printf("\n");
    printf("\n");
    printf("Parent node: %s\n", $NodeId);

    // Instantiate the client object
    $Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

    // Obtain notifiers
    $NotifierNodeElements = $Client->BrowseNotifiers($EndpointDescriptor, $NodeId-
>ExpandedText);

    // Display notifires
    if ($NotifierNodeElements->Count != 0) {
        printf("\n");
        printf("Notifiers:\n");
        foreach ($NotifierNodeElements as $NotifierNodeElement)
        {
             printf("%s\n", $NotifierNodeElement);
        }
    }

    // Obtain objects
    $ObjectNodeElements = $Client->BrowseObjects($EndpointDescriptor, $NodeId-
>ExpandedText);

    // Recurse
    foreach ($ObjectNodeElements as $ObjectNodeElement)
    {
         BrowseFrom($ObjectNodeElement->NodeId);
    }
}

// Example output (truncated):
//
//
//Parent node: ObjectsFolder
//
//
```

```
//Parent node: Server
//
//Notifiers:
//Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
//Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
//
//
//Parent node: Server_ServerCapabilities
//...
```

VB.NET

```
' This example shows how to browse objects under the "Objects" node and display
notifiers.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class BrowseNotifiers
        Public Shared Sub Overload2()

            ' Start browsing from the "Objects" node
            Try
                BrowseFrom(UAObjectIds.ObjectsFolder)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try
        End Sub

        Private Shared Sub BrowseFrom(nodeDescriptor As UANodeDescriptor)

            Console.WriteLine()
            Console.WriteLine()
            Console.WriteLine("Parent node: {0}", nodeDescriptor)

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain notifiers
            Dim notifierNodeElementCollection As UANodeElementCollection =
client.BrowseNotifiers( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Display notifiers
            Console.WriteLine()
            Console.WriteLine("Notifiers:")
            For Each notifierNodeElement As UANodeElement In
notifierNodeElementCollection
```

```
                Console.WriteLine(notifierNodeElement)
            Next notifierNodeElement

            ' Obtain objects
            Dim objectNodeElementCollection = client.BrowseObjects( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Recurse
            For Each objectNodeElement As UANodeElement In objectNodeElementCollection
                BrowseFrom(objectNodeElement)
            Next objectNodeElement
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to browse objects under the "Objects" node and display
notifiers.

Option Explicit

' Start browsing from the "Objects" node
Dim ObjectsNodeId: Set ObjectsNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ObjectsNodeId.StandardName = "Objects"
On Error Resume Next
BrowseFrom ObjectsNodeId
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0



Sub BrowseFrom(NodeId)
    Dim endpointDescriptor
    endpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"

    WScript.Echo
    WScript.Echo
    WScript.Echo "Parent node: " & NodeId

    ' Instantiate the client object
    Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

    ' Obtain notifiers
    Dim NotifierNodeElementCollection: Set NotifierNodeElementCollection =
Client.BrowseNotifiers( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Display notifiers
    If NotifierNodeElementCollection.Count <> 0 Then
```

```
        WScript.Echo
        WScript.Echo "Notifiers:"
        Dim NotifierNodeElement: For Each NotifierNodeElement In
NotifierNodeElementCollection
            WScript.Echo NotifierNodeElement
        Next
    End If

    ' Obtain objects
    Dim ObjectNodeElementCollection: Set ObjectNodeElementCollection =
Client.BrowseObjects( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Recurse
    Dim ObjectNodeElement: For Each ObjectNodeElement In ObjectNodeElementCollection
        BrowseFrom ObjectNodeElement.NodeId
    Next
End Sub



' Example output (truncated):
'
'
'Parent node: ObjectsFolder
'
'
'Parent node: Server
'
'Notifiers:
'Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
'Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
'
'
'Parent node: Server_ServerCapabilities
'...
```

5.1.4.4 Subscribing to Information (OPC UA Alarms & Conditions)

If your application needs to be informed about events occurring in the process and provided by the OPC UA Alarms & Conditions server, it can subscribe to them, and receive notifications.

OPC Data Client contains methods that allow you to subscribe to OPC events, change the subscription parameters, and unsubscribe.

5.1.4.4.1 Subscribing to OPC UA Events

Generally speaking, you subscribe to OPC UA Events similarly to data changes, but by using monitoring parameters that contain an event filter instead of data change filter. Besides the UAMonitoringParameters.DataChangeFilter property, the UAMonitoringParameters class also has an EventFilter property (of UAEventFilter type). The developer can specify either a data change filter, or an event filter, with monitoring parameters.

In This Topic

As the UAMonitoringParameters are part of the information that gets passed (directly or indirectly) to any EasyUAClient.SubscribeMonitoredItem or SubscribeMultipleMonitoredItems call, the introduction of the EventFilter property and the EventNotification event (both described above) are, in fact, the only core features that already in itself allow the Alarms & Conditions support to work. Many other features simply provide better developer experience above this.

A single event

Instead of putting together the monitoring parameters with an event filter, you can subscribe to events a bit easier using the IEasyUAClient.SubscribeEvent extension method, with several overloads. Similarly to the IEasyUAClient.SubscribeDataChange extension method (for data changes), the SubscribeEvent method provides a simplified way to subscribe to events.

C#

```
// This example shows how to subscribe to event notifications and display each incoming event.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    partial class SubscribeEvent
    {
        public static void Overload1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs
e)
        {
            // Display the event
            Console.WriteLine(e);
```

```
        }


        // Example output (truncated):
        //Subscribing...
        //Processing event notifications for 30 seconds...
        //[] Success
        //[] Success; Refresh; RefreshInitiated
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
        //[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
        //[] Success; Refresh; RefreshComplete
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42
```

```
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48
PM
        //[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49
PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50
PM
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to event notifications and display each incoming event.

type
  TClientEventHandlers18 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers18.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
  if eventArgs.Succeeded then
    WriteLn(eventArgs.ToString)
  else
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
end;

class procedure SubscribeEvent.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
```

```
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers18;
  EndpointDescriptor: string;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers18.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(EndpointDescriptor, UAObjectIds_Server, 1000);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);

end;

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was acknoweledged."
@11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has increased."
@9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has increased."
@9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
```

```
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has increased."
@9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has increased."
@9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was acknoweledged."
@10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50 PM
//...
```

PHP

```
// This example shows how to subscribe to event notifications and display each incoming event.


class ClientEvents {
    function EventNotification($Sender, $E)
    {
        // Display the event
        if ($E->Succeeded) {
            printf("%s\n", $E);
```

```
        }
        else {
            printf(" *** Failure: %s\n", $E->ErrorMessageBrief);
        }
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeEvent($EndpointDescriptor, UAObjectIds_Server, 1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was acknoweledged."
@11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has increased."
@9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has increased."
@9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has increased."
```

```
@11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has increased."
@9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has increased."
@9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was acknoweledged."
@10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50 PM
//...
```

VB.NET

```
' This example shows how to subscribe to event notifications and display each incoming event.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class SubscribeEvent
        Public Shared Sub Overload1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification
```

```
            Console.WriteLine("Subscribing...")
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000)

            Console.WriteLine("Processing event notifications for 10 seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As
EasyUAEventNotificationEventArgs)
            ' Display the event
            Console.WriteLine(e)
        End Sub
    End Class
End Namespace
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to event notifications and display each incoming
event.

' The client object, with events
'Public WithEvents Client1 As EasyUAClient

Private Sub SubscribeEvent_Main_Command_Click()
    OutputText = ""

    Set Client1 = New EasyUAClient

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Call Client1.SubscribeEvent("opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", "nsu=http://opcfoundation.org/UA/;i=2253",
1000)

    OutputText = OutputText & "Processing event notifications for 30 seconds..." & vbCrLf
    Pause 30000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Client1.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client1 = Nothing
End Sub

Private Sub Client1_EventNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUAEventNotificationEventArgs)
    ' Display the event
    If EventArgs.Succeeded Then
        OutputText = OutputText & EventArgs & vbCrLf
    Else
        OutputText = OutputText & EventArgs.ErrorMessageBrief & vbCrLf
```

```
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to event notifications and display each incoming
event.

Option Explicit

Const uaObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Client.SubscribeEvent "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
uaObjectIds_Server, 1000

WScript.Echo "Processing event notifications for 30 seconds..."
WScript.Sleep 30*1000



Sub Client_EventNotification(Sender, e)
    ' Display the event
    WScript.Echo e
End Sub



' Example output (truncated):
'Subscribing...
'Processing event notifications for 30 seconds...
'[] Success
'[] Success; Refresh; RefreshInitiated
'[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated" @9/10/2019
8:08:23 PM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:13 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was acknoweledged."
@11/9/2019 9:56:23 AM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:17 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has increased."
@9/10/2019 8:09:07 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has increased."
@9/10/2019 8:10:09 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:02 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:16 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 2:56:37 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has increased."
@9/10/2019 8:09:11 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has increased."
@9/10/2019 8:10:19 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated" @9/10/2019
```

```
8:08:25 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:12 PM
'[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 2:56:37 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:04 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has increased."
@9/10/2019 8:08:58 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has increased."
@9/10/2019 8:09:48 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 2:56:37 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was acknoweledged."
@10/14/2019 4:00:21 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:03 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has increased."
@9/10/2019 8:09:02 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has increased."
@9/10/2019 8:09:59 PM
'[] Success; Refresh; RefreshComplete
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48 PM
'[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
'[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased." @11/9/2019
2:56:48 PM
'[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50 PM
'...
```

Multiple events

Multiple events are more efficiently subscribed using the SubscribeMultipleMonitoredItems method:

C#

```
// This example shows how to subscribe to multiple events.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class SubscribeMultipleMonitoredItems
    {
        public static void Events()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments("firstState",
                            endpointDescriptor,
                        UAObjectIds.Server,
                        new UAMonitoringParameters(1000, new UAEventFilterBuilder(

UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500),
                            UABaseEventObject.AllFields)))
                        { AttributeId = UAAttributeId.EventNotifier },
                    new EasyUAMonitoredItemArguments("secondState",
                            endpointDescriptor,
                        UAObjectIds.Server,
                        new UAMonitoringParameters(2000, new UAEventFilterBuilder(
                            UAFilterElements.Equals(
                                UABaseEventObject.Operands.SourceNode,
                                new
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor")),

                            UABaseEventObject.AllFields)))
                        { AttributeId = UAAttributeId.EventNotifier },
                });

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs
e)
```

```
        {
            // Display the event
            Console.WriteLine(e);
        }



        // Example output (truncated):
        //Subscribing...
        //Processing monitored item changed events for 30 seconds...
        //[firstState] Success
        //[secondState] Success
        //[firstState] Success; Refresh; RefreshInitiated
        //[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
        //[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
        //[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
        //[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
        //[firstState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[firstState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[firstState] Success; Refresh; RefreshComplete
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:08 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:08 PM
        //[secondState] Success; Refresh; RefreshInitiated
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm is
active." @11/8/2019 7:48:07 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm
severity has increased." @9/10/2019 8:09:02 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm
severity has increased." @9/10/2019 8:09:59 PM
        //[secondState] Success; Refresh; RefreshComplete
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:09 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:09 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:10 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:10 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:11 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:11 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:12 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:12 PM
        //[firstState] Success; (10 field results) [EastTank] 500! "The alarm severity has
increased." @11/8/2019 7:48:13 PM
```

```
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:13 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:13 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:14 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:14 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:15 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:15 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:16 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:16 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:17 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:17 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:18 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:18 PM
        //[secondState] Success; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/8/2019 7:48:18 PM
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to multiple events.

type
  THelperMethods19 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_AllFields: _UAAttributeFieldCollection; static;
  end;

type
  TClientEventHandlers19 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers19.Client_EventNotification(
  ASender: TObject;
```

```
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
    WriteLn(eventArgs.ToString);
end;


class procedure SubscribeMultipleMonitoredItems.Events;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers19;
  EndpointDescriptor: string;
  EventFilter1, EventFilter2: _UAEventFilter;
  MonitoredItemArguments1, MonitoredItemArguments2: _EasyUAMonitoredItemArguments;
  MonitoringParameters1, MonitoringParameters2: _UAMonitoringParameters;
  Operand11, Operand21: _UASimpleAttributeOperand;
  Operand12, Operand22: _UALiteralOperand;
  ServerNodeId1, ServerNodeId2: _UANodeId;
  SourceNodeId: _UANodeId;
  WhereClause1, WhereClause2: _UAContentFilterElement;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  // set MonitoredItemArguments1
  // Event filter: The severity is >= 500.
  Operand11 := THelperMethods19.UABaseEventObject_Operands_Severity;
  Operand12 := CoUALiteralOperand.Create;
  Operand12.Value := 500;
  WhereClause1 := CoUAContentFilterElement.Create;
  WhereClause1.FilterOperator := UAFilterOperator_GreaterThanOrEqual;
  WhereClause1.FilterOperands.Add(Operand11);
  WhereClause1.FilterOperands.Add(Operand12);

  EventFilter1 := CoUAEventFilter.Create;
  EventFilter1.SelectClauses := THelperMethods19.UABaseEventObject_AllFields;
  EventFilter1.WhereClause := WhereClause1;

  ServerNodeId1 := CoUANodeId.Create;
  ServerNodeId1.StandardName := 'Server';

  MonitoringParameters1 := CoUAMonitoringParameters.Create;
  MonitoringParameters1.EventFilter := EventFilter1;
  MonitoringParameters1.QueueSize := 1000;
  MonitoringParameters1.SamplingInterval := 1000;

  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.AttributeId := UAAttributeId_EventNotifier;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters1;
  MonitoredItemArguments1.NodeDescriptor.NodeId := ServerNodeId1;
  MonitoredItemArguments1.State := 'firstState';

  // set MonitoredItemArguments2
  // Event filter: The event comes from a specified source node.
  Operand21 := THelperMethods19.UABaseEventObject_Operands_SourceNode;
  SourceNodeId := CoUANodeId.Create;
  SourceNodeId.ExpandedText := 'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Metals/SouthMotor';
  Operand22 := CoUALiteralOperand.Create;
```

```
  Operand22.Value := SourceNodeId;
  WhereClause2 := CoUAContentFilterElement.Create;
  WhereClause2.FilterOperator := UAFilterOperator_Equals;
  WhereClause2.FilterOperands.Add(Operand21);
  WhereClause2.FilterOperands.Add(Operand22);

  EventFilter2 := CoUAEventFilter.Create;
  EventFilter2.SelectClauses := THelperMethods19.UABaseEventObject_AllFields;
  EventFilter2.WhereClause := WhereClause2;

  ServerNodeId2 := CoUANodeId.Create;
  ServerNodeId2.StandardName := 'Server';

  MonitoringParameters2 := CoUAMonitoringParameters.Create;
  MonitoringParameters2.EventFilter := EventFilter2;
  MonitoringParameters2.QueueSize := 1000;
  MonitoringParameters2.SamplingInterval := 2000;

  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.AttributeId := UAAttributeId_EventNotifier;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters2;
  MonitoredItemArguments2.NodeDescriptor.NodeId := ServerNodeId2;
  MonitoredItemArguments2.State := 'secondState';


  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers19.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  Arguments := VarArrayCreate([0, 1], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;

  WriteLn('Subscribing...');
  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing monitored item changed events for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;

class function THelperMethods19.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods19.UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand;
```

```
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames :=
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods19.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods19.UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
end;

class function THelperMethods19.UABaseEventObject_Operands_EventType:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
end;

class function THelperMethods19.UABaseEventObject_Operands_SourceNode:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods19.UABaseEventObject_Operands_SourceName:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods19.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;

class function THelperMethods19.UABaseEventObject_Operands_ReceiveTime:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
end;

class function THelperMethods19.UABaseEventObject_Operands_LocalTime:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
end;
```

```
class function THelperMethods19.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods19.UABaseEventObject_Operands_Severity:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;

class function THelperMethods19.UABaseEventObject_AllFields: _UAAttributeFieldCollection;
var
  Fields: _UAAttributeFieldCollection;
begin
  Fields := CoUAAttributeFieldCollection.Create;
  Fields.Add(UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventType.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Time.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_LocalTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Message.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Severity.ToUAAttributeField);

  Result := Fields;
end;
```

PHP

```
// This example shows how to subscribe to multiple events.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

const UAFilterOperator_Equals = 1;
const UAFilterOperator_GreaterThanOrEqual = 5;

class ClientEvents {
    function EventNotification($Sender, $E)
    {
        // Display the event
        printf("%s\n", $E);
    }
}

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// set MonitoredItemArguments1
// Event filter: The severity is >= 500.
$Operand11 = UABaseEventObject_Operands_Severity();
$Operand12 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand12->Value = 500;
$WhereClause1 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$WhereClause1->FilterOperator = UAFilterOperator_GreaterThanOrEqual;
$WhereClause1->FilterOperands->Add($Operand11);
$WhereClause1->FilterOperands->Add($Operand12);

$EventFilter1 = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter1->SelectClauses = UABaseEventObject_AllFields();
```

```
$EventFilter1->WhereClause = $WhereClause1;

$ServerNodeId1 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId1->StandardName = "Server";

$MonitoringParameters1 = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters1->EventFilter = $EventFilter1;
$MonitoringParameters1->QueueSize = 1000;
$MonitoringParameters1->SamplingInterval = 1000;

$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->AttributeId = UAAttributeId_EventNotifier;
$MonitoredItemArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters1;
$MonitoredItemArguments1->NodeDescriptor->NodeId = $ServerNodeId1;
$MonitoredItemArguments1->State = "firstState";

// set MonitoredItemArguments2
// Event filter: The event comes from a specified source node.
$Operand21 = UABaseEventObject_Operands_SourceNode();
$SourceNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$SourceNodeId->ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Metals/SouthMotor";
$Operand22 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand22->Value = $SourceNodeId;
$WhereClause2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$WhereClause2->FilterOperator = UAFilterOperator_Equals;
$WhereClause2->FilterOperands->Add($Operand21);
$WhereClause2->FilterOperands->Add($Operand22);

$EventFilter2 = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter2->SelectClauses = UABaseEventObject_AllFields();
$EventFilter2->WhereClause = $WhereClause2;

$ServerNodeId2 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId2->StandardName = "Server";

$MonitoringParameters2 = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters2->EventFilter = $EventFilter2;
$MonitoringParameters2->QueueSize = 1000;
$MonitoringParameters2->SamplingInterval = 2000;

$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->AttributeId = UAAttributeId_EventNotifier;
$MonitoredItemArguments2->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters2;
$MonitoredItemArguments2->NodeDescriptor->NodeId = $ServerNodeId2;
$MonitoredItemArguments2->State = "secondState";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);
```

```
printf("Processing monitored item changed events for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser-
>ParseRelative($simpleRelativeBrowsePathString)->ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_EventId() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventId");
}

function UABaseEventObject_Operands_EventType() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventType");
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}

function UABaseEventObject_Operands_ReceiveTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/ReceiveTime");
}

function UABaseEventObject_Operands_LocalTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/LocalTime");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
```

```
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}


function UABaseEventObject_AllFields() {
    $Fields = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
    $Fields->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventType()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_ReceiveTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_LocalTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField);

    return $Fields;
}

// Example output (truncated):
//Subscribing...
//Processing monitored item changed events for 30 seconds...
//[firstState] Success
//[secondState] Success
//[firstState] Success; Refresh; RefreshInitiated
//[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[firstState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[firstState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[firstState] Success; Refresh; RefreshComplete
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:08
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:08
PM
//[secondState] Success; Refresh; RefreshInitiated
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm is active."
@11/8/2019 7:48:07 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[secondState] Success; Refresh; RefreshComplete
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:09
PM
```

```
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:09
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:10
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:10
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:11
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:11
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:12
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:12
PM
//[firstState] Success; (10 field results) [EastTank] 500! "The alarm severity has increased."
@11/8/2019 7:48:13 PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:13
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:13
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:14
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:14
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:15
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:15
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:16
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:16
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:17
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:17
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:18
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:18
PM
//[secondState] Success; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/8/2019 7:48:18 PM
//...
```

VB.NET

```
' This example shows how to subscribe to multiple events.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub Events()

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
```

```
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")

            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments("firstState", _
                        "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", _
                        UAObjectIds.Server, _
                        New UAMonitoringParameters(1000, New UAEventFilterBuilder( _

UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500), _
                            UABaseEventObject.AllFields))) _
                        With {.AttributeId = UAAttributeId.EventNotifier}, _
                    New EasyUAMonitoredItemArguments("secondState", _
                        "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", _
                        UAObjectIds.Server, _
                        New UAMonitoringParameters(2000, New UAEventFilterBuilder( _
                            UAFilterElements.Equals( _
                                UABaseEventObject.Operands.SourceNode, _
                                New
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor")),
_
                            UABaseEventObject.AllFields))) _
                        With {.AttributeId = UAAttributeId.EventNotifier} _
                } _
             )

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As
EasyUAEventNotificationEventArgs)
            ' Display the event
            Console.WriteLine(e)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to subscribe to multiple events.

Option Explicit

Const UAAttributeId_NodeId = 1
Const UAAttributeId_EventNotifier = 12

Const UAFilterOperator_Equals = 1
Const UAFilterOperator_GreaterThanOrEqual = 5

Dim endpointDescriptor
endpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer"
```

```
' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

Dim arguments(1)
Set arguments(0) = CreateMonitoredItemArguments1
Set arguments(1) = CreateMonitoredItemArguments2

WScript.Echo "Subscribing..."
Client.SubscribeMultipleMonitoredItems arguments

WScript.Echo "Processing monitored item changed events for 30 seconds..."
WScript.Sleep 30 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Function ObjectTypeIds_BaseEventType
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.StandardName = "BaseEventType"
    Set ObjectTypeIds_BaseEventType = NodeId
End Function

Function UAFilterElements_SimpleAttribute(TypeId, simpleRelativeBrowsePathString)
    Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
    Dim Operand: Set Operand =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Set Operand.TypeId.NodeId = TypeId
    Set Operand.QualifiedNames =
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection
    Set UAFilterElements_SimpleAttribute = Operand
End Function

Function UABaseEventObject_Operands_NodeId
    Dim Operand: Set Operand =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Operand.TypeId.NodeId.StandardName = "BaseEventType"
    Operand.AttributeId = UAAttributeId_NodeId
    Set UABaseEventObject_Operands_NodeId = Operand
End Function

Function UABaseEventObject_Operands_EventId
    Set UABaseEventObject_Operands_EventId =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventId")
End Function

Function UABaseEventObject_Operands_EventType
    Set UABaseEventObject_Operands_EventType =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventType")
End Function

Function UABaseEventObject_Operands_SourceNode
    Set UABaseEventObject_Operands_SourceNode =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceNode")
End Function
```

```
Function UABaseEventObject_Operands_SourceName
    Set UABaseEventObject_Operands_SourceName =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceName")
End Function

Function UABaseEventObject_Operands_Time
    Set UABaseEventObject_Operands_Time =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Time")
End Function

Function UABaseEventObject_Operands_ReceiveTime
    Set UABaseEventObject_Operands_ReceiveTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/ReceiveTime")
End Function

Function UABaseEventObject_Operands_LocalTime
    Set UABaseEventObject_Operands_LocalTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/LocalTime")
End Function

Function UABaseEventObject_Operands_Message
    Set UABaseEventObject_Operands_Message =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Message")
End Function

Function UABaseEventObject_Operands_Severity
    Set UABaseEventObject_Operands_Severity =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Severity")
End Function

Function UABaseEventObject_AllFields
    Dim Fields: Set Fields = CreateObject("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection")

    Fields.Add UABaseEventObject_Operands_NodeId.ToUAAttributeField

    Fields.Add UABaseEventObject_Operands_EventId.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_EventType.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceNode.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceName.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Time.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_LocalTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Message.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Severity.ToUAAttributeField

    Set UABaseEventObject_AllFields = Fields
End Function

Function CreateMonitoredItemArguments1
    ' Event filter: The severity is >= 500.
    Dim Operand1: Set Operand1 = UABaseEventObject_Operands_Severity
    Dim Operand2: Set Operand2 = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand")
    Operand2.Value = 500
    Dim WhereClause: Set WhereClause =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement")
    WhereClause.FilterOperator = UAFilterOperator_GreaterThanOrEqual
    WhereClause.FilterOperands.Add Operand1
    WhereClause.FilterOperands.Add Operand2

    Dim EventFilter: Set EventFilter = CreateObject("OpcLabs.EasyOpc.UA.UAEventFilter")
    Set EventFilter.SelectClauses = UABaseEventObject_AllFields
```

```
    Set EventFilter.WhereClause = WhereClause

    Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    ServerNodeId.StandardName = "Server"

    Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
    Set MonitoringParameters.EventFilter = EventFilter
    MonitoringParameters.QueueSize = 1000
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments: Set MonitoredItemArguments =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
    MonitoredItemArguments.AttributeId = UAAttributeId_EventNotifier
    MonitoredItemArguments.EndpointDescriptor.UrlString = endpointDescriptor
    Set MonitoredItemArguments.MonitoringParameters = MonitoringParameters
    Set MonitoredItemArguments.NodeDescriptor.NodeId = ServerNodeId
    MonitoredItemArguments.State = "firstState"

    Set CreateMonitoredItemArguments1 = MonitoredItemArguments
End Function

Function CreateMonitoredItemArguments2
    ' Event filter: The event comes from a specified source node.
    Dim Operand1: Set Operand1 = UABaseEventObject_Operands_SourceNode
    Dim SourceNodeId: Set SourceNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    SourceNodeId.ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Metals/SouthMotor"
    Dim Operand2: Set Operand2 = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand")
    Set Operand2.Value = SourceNodeId
    Dim WhereClause: Set WhereClause =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement")
    WhereClause.FilterOperator = UAFilterOperator_Equals
    WhereClause.FilterOperands.Add Operand1
    WhereClause.FilterOperands.Add Operand2

    Dim EventFilter: Set EventFilter = CreateObject("OpcLabs.EasyOpc.UA.UAEventFilter")
    Set EventFilter.SelectClauses = UABaseEventObject_AllFields
    Set EventFilter.WhereClause = WhereClause

    Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    ServerNodeId.StandardName = "Server"

    Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
    Set MonitoringParameters.EventFilter = EventFilter
    MonitoringParameters.QueueSize = 1000
    MonitoringParameters.SamplingInterval = 2000

    Dim MonitoredItemArguments: Set MonitoredItemArguments =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
    MonitoredItemArguments.AttributeId = UAAttributeId_EventNotifier
    MonitoredItemArguments.EndpointDescriptor.UrlString = endpointDescriptor
    Set MonitoredItemArguments.MonitoringParameters = MonitoringParameters
    Set MonitoredItemArguments.NodeDescriptor.NodeId = ServerNodeId
    MonitoredItemArguments.State = "secondState"

    Set CreateMonitoredItemArguments2 = MonitoredItemArguments
End Function
```

```
Sub Client_EventNotification(Sender, e)
    ' Display the event
    WScript.Echo e
End Sub



' Example output (truncated):
'Subscribing...
'Processing monitored item changed events for 30 seconds...
'[firstState] Success
'[secondState] Success
'[firstState] Success; Refresh; RefreshInitiated
'[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
'[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
'[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
'[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
'[firstState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[firstState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[firstState] Success; Refresh; RefreshComplete
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:08
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:08
PM
'[secondState] Success; Refresh; RefreshInitiated
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm is active."
@11/8/2019 7:48:07 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
'[secondState] Success; Refresh; RefreshComplete
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:09
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:09
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:10
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:10
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:11
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:11
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:12
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:12
PM
'[firstState] Success; (10 field results) [EastTank] 500! "The alarm severity has increased."
```

```
@11/8/2019 7:48:13 PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:13
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:13
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:14
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:14
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:15
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:15
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:16
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:16
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:17
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:17
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:18
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:18
PM
'[secondState] Success; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/8/2019 7:48:18 PM
'...
```

5.1.4.4.2 Specifying Event Filters (OPC UA Alarms & Conditions)

The UAEventFilter object (used for specifying the event filters) is probably the most complicated part of the OPC UA Alarms & Condition features. It has two main parts: Select clauses (contained in the SelectClauses property, of UAAttributeFieldCollection type), and a Where clause (contained in the WhereClause property, of UAContentFilterElement type). The Select clauses contain a collection of the attribute fields to return with each event in a notification. The Where clause contains the criteria limiting the notifications.

In order to make the creation of some event filters appear shorter in the code, a UAEventFilterBuilder class is provided. Instances of this builder can be implicitly converted to the event filter itself (hence you can create just the UAEventFilterBuilder objects, and use them at all places where the UAEventFilter object is expected). The builder contains a Where clause, and is a collection of Select clauses. You can therefore use the C# collection initializer to specify the Select clauses.

In COM (tools like VB6, Delphi etc.), creation of OPC UA Event Filters requires longer code. For related information, see http://kb.opclabs.com/Creating_an_OPC_UA_event_filter_in_COM .

5.1.4.4.2.1 The Select clauses

Each Select clause of the event filter is a UAAttributeField object, which contains an Operand property. You can optionally set its the State property to an object of your choice, and the same object will then be made available to you in the event notification containing the operand. An Operand is of the UASimpleAttributeOperand type,

and specifies primarily a TypeId node in the server, and a QualifiedNames collection (of UAQualifiedNameCollection type), which is a so-called simple relative path (inside that type) that leads to the required node. A simple relative path can only contain forward, "any hierarchical" references.

Conversion methods, and conversion operators exist between the UAQualifiedNameCollection (for simple relative paths) and UABrowsePathElementCollection (for general relative paths). Conversion from UAQualifiedNameCollection to UABrowsePathElementCollection is always possible and has an implicit conversion operator, conversion from UABrowsePathElementCollection to UAQualifiedNameCollection is only possible for simple relative path, and has an explicit conversion operator instead.

In order to make the specification of the most commonly used Select clauses easier, it is possible to use pre-defined clauses provided in the static UABaseEventObject.Operands class (for example, UABaseEventObject. Operands.Message). The UABaseEventObject.AllFields property can be used to obtain Select clauses for all fields of the UA base event type at once.

C#

```
// This example shows how to select fields for event notifications.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class SelectClauses
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                new UAAttributeFieldCollection
                {
                    // Select specific fields using standard operand symbols
                    UABaseEventObject.Operands.NodeId,
                    UABaseEventObject.Operands.SourceNode,
                    UABaseEventObject.Operands.SourceName,
                    UABaseEventObject.Operands.Time,

                    // Select specific fields using an event type ID and a simple relative path
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message"),
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Severity"),
                });

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            Console.WriteLine();

            // Display the event
            if (e.EventData == null)
            {
                Console.WriteLine(e);
                return;
            }
            Console.WriteLine("All fields:");
            foreach (KeyValuePair<UAAttributeField, ValueResult> pair in e.EventData.FieldResults)
```

```
            {
                UAAttributeField attributeField = pair.Key;
                ValueResult valueResult = pair.Value;
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult);
            }
            // Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}",
                e.EventData.FieldResults[UABaseEventObject.Operands.SourceName]);
            // Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}",
                e.EventData.FieldResults[UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")]);
        }
    }
}
```

Object Pascal

```
// This example shows how to select fields for event notifications.

type
  THelperMethods17 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString: string):
_UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
  end;

type
  TClientEventHandlers17 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers17.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  AttributeField: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  EntryEnumerator: IEnumVARIANT;
  ValueResult: OleVariant;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;

  WriteLn('All fields:');

  EntryEnumerator := eventArgs.EventData.FieldResults.GetEnumerator;
  while (EntryEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    AttributeField := IUnknown(Element.Key) as _UAAttributeField;
    ValueResult := IUnknown(Element.Value) as _ValueResult;
    WriteLn(' ', AttributeField.ToString, ' -> ', ValueResult.ToString);
  end;

  // Extracting specific fields using an event type ID and a simple relative path
  WriteLn('Source name: ',
eventArgs.EventData.FieldResults.Item[THelperMethods17.UABaseEventObject_Operands_SourceName.ToUAAttributeField].ToString);

  WriteLn('Message: ',
eventArgs.EventData.FieldResults.Item[THelperMethods17.UABaseEventObject_Operands_Message.ToUAAttributeField].ToString);
end;

class procedure SelectClauses.Main;
```

```
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers17;
  EndpointDescriptor: string;
  EventFilter: _UAEventFilter;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  SelectClauses: UAAttributeFieldCollection;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers17.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');

  SelectClauses := CoUAAttributeFieldCollection.Create;
  // Select specific fields using an event type ID and a simple relative path
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_Time.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_Message.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_Severity.ToUAAttributeField);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := SelectClauses;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;


class function THelperMethods17.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods17.UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
```

```
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames := BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods17.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods17.UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods17.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods17.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;

class function THelperMethods17.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods17.UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;
```

PHP

```
// This example shows how to select fields for event notifications.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

class ClientEvents {
    function EventNotification($Sender, $E)
    {
        printf("\n");

        // Display the event
        if (is_null($E->EventData)) {
            printf("%s\n", $E);
            return;
        }

        printf("All fields:\n");

        foreach ($E->EventData->FieldResults as $Pair)
        {
            $AttributeField = $Pair->Key;
            $ValueResult = $Pair->Value;
            printf("  %s -> %s\n", $AttributeField, $ValueResult);
        }

        // Extracting specific fields using an event type ID and a simple relative path
        printf("Source name: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_SourceName()-
>ToUAAttributeField()));
        printf("Message: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_Message()-
>ToUAAttributeField()));
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";
```

```
$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");

$SelectClauses = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
// Select specific fields using an event type ID and a simple relative path
$SelectClauses->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField());

$EventFilter = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter->SelectClauses = $SelectClauses;

$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->EventFilter = $EventFilter;
$MonitoringParameters->QueueSize = 1000;
$MonitoringParameters->SamplingInterval = 1000;

$MonitoredItemArguments = new COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments->NodeDescriptor->NodeId->StandardName = "Server";
$MonitoredItemArguments->MonitoringParameters = $MonitoringParameters;
// $MonitoredItemArguments->SubscriptionParameters->PublishingInterval = 0;
$MonitoredItemArguments->AttributeId = UAAttributeId_EventNotifier;

$Arguments[0] = $MonitoredItemArguments;

$Client->SubscribeMultipleMonitoredItems($Arguments);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
```

```
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}


// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//
//[] Success
//
//[] Success; Refresh; RefreshInitiated
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The dialog was activated {System.String}
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The alarm was acknoweledged. {System.String}
//
//...
```

VB.NET

```
' This example shows how to select fields for event notifications.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class SelectClauses
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            ' Select specific fields using standard operand symbols
            ' Select specific fields using an event type ID and a simple relative path
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000, _
                New UAAttributeFieldCollection() From
```

```
                {
                    UABaseEventObject.Operands.NodeId,
                    UABaseEventObject.Operands.SourceNode,
                    UABaseEventObject.Operands.SourceName,
                    UABaseEventObject.Operands.Time,
 _
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message"),
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Severity")
                })

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As EasyUAEventNotificationEventArgs)
            Console.WriteLine()

            ' Display the event
            If e.EventData Is Nothing Then
                Console.WriteLine(e)
                Exit Sub
            End If
            For Each pair In e.EventData.FieldResults
                Dim attributeField = pair.Key
                Dim valueResult = pair.Value
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult)
            Next pair
            ' Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}", _
                e.EventData.FieldResults(UABaseEventObject.Operands.SourceName))
            ' Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}", _
                e.EventData.FieldResults(UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")))
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to select fields for event notifications.

type
  TClientEventHandlers4 = class
    procedure OnEventNotification(
      Sender: TObject;
      sender0: OleVariant;
      eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers4.OnEventNotification(
  Sender: TObject;
  sender0: OleVariant;
  eventArgs: _EasyUAEventNotificationEventArgs);

  function ToUAAttributeField(Operand: UASimpleAttributeOperand): UAAttributeField;
  var
    AttributeField: UAAttributeField;
  begin
    AttributeField := CoUAAttributeField.Create;
    AttributeField.Operand := Operand;
    ToUAAttributeField := AttributeField;
  end;

  function UAFilterElements_SimpleAttribute(TypeId: UANodeId; SimpleRelativeBrowsePathString: string):
UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
    BrowsePathParser: UABrowsePathParser;
  begin
    BrowsePathParser := CoUABrowsePathParser.Create;
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId := TypeId;
    Operand.QualifiedNames := BrowsePathParser.ParseRelative(SimpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
```

```
    UAFilterElements_SimpleAttribute := Operand;
  end;

  function ObjectTypeIds_BaseEventType: UANodeId;
  var
    NodeId: UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    ObjectTypeIds_BaseEventType := NodeId;
  end;

var
  Count: Cardinal;
  Element: OleVariant;
  EntryEnumerator: IEnumVariant;
  Entry: DictionaryEntry2;
  AttributeField: UAAttributeField;
  AValueResult: ValueResult;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;
  WriteLn('All fields:');
  EntryEnumerator := eventArgs.EventData.FieldResults.GetEnumerator;
  while EntryEnumerator.Next(1, Element, Count) = S_OK do
  begin
    Entry := IUnknown(Element) as DictionaryEntry2;
    AttributeField := IUnknown(Entry.key) as UAAttributeField;
    AValueResult := IUnknown(Entry.Value) as ValueResult;
    WriteLn('  ', AttributeField.ToString, ' -> ', AValueResult.ToString);
  end;
  // Extracting a specific field using an event type ID and a simple relative path
  WriteLn('Source name: ',
    eventArgs.EventData.FieldResults[ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/SourceName'))].ToString);
  WriteLn('Message: ',
    eventArgs.EventData.FieldResults[ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/Message'))].ToString);
end;

class procedure SelectClauses.Main;

  function ToUAAttributeField(Operand: UASimpleAttributeOperand): UAAttributeField;
  var
    AttributeField: UAAttributeField;
  begin
    AttributeField := CoUAAttributeField.Create;
    AttributeField.Operand := Operand;
    ToUAAttributeField := AttributeField;
  end;

  function ObjectTypeIds_BaseEventType: UANodeId;
  var
    NodeId: UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    ObjectTypeIds_BaseEventType := NodeId;
  end;

  function UABaseEventObject_Operands_NodeId: UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
  begin
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId.StandardName := 'BaseEventType';
    Operand.AttributeId := UAAttributeId_NodeId;
    UABaseEventObject_Operands_NodeId := Operand;
  end;

  function UAFilterElements_SimpleAttribute(TypeId: UANodeId; SimpleRelativeBrowsePathString: string):
UASimpleAttributeOperand;
  var
```

```
    Operand: UASimpleAttributeOperand;
    BrowsePathParser: UABrowsePathParser;
  begin
    BrowsePathParser := CoUABrowsePathParser.Create;
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId := TypeId;
    Operand.QualifiedNames := BrowsePathParser.ParseRelative(SimpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
    UAFilterElements_SimpleAttribute := Operand;
  end;

var
  Arguments: OleVariant;
  EvsClient: TEvsEasyUAClient;
  Client: EasyUAClient;
  ClientEventHandlers: TClientEventHandlers4;
  Handle: Cardinal;
  HandleArray: OleVariant;
  MonitoredItemArguments: EasyUAMonitoredItemArguments;
  MonitoringParameters: UAMonitoringParameters;
  EventFilter: UAEventFilter;
  SelectClauses: UAAttributeFieldCollection;
begin
  // Instantiate the client object and hook events
  EvsClient := TEvsEasyUAClient.Create(nil);
  Client := EvsClient.ComServer;
  ClientEventHandlers := TClientEventHandlers4.Create;
  EvsClient.OnEventNotification := @ClientEventHandlers.OnEventNotification;

  WriteLn('Subscribing...');

  SelectClauses := CoUAAttributeFieldCollection.Create;
  // Select specific fields using an event type ID and a simple relative path
  SelectClauses.Add(ToUAAttributeField(UABaseEventObject_Operands_NodeId));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity')));

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := SelectClauses;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(PSafeArray(TVarData(Arguments).VArray)));

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Done...');
end;
```

5.1.4.4.2.2 The Where clause

The Where clause of the event filter is represented using the UAContentFilterElement object. This objects contains the filter operand to be evaluated (the FilterOperator property, which is a UAFilterOperator enumeration), and the operands used by the selected operator (the FilterOperands property, which is a UAFilterOperandCollection). The UAFilterOperator enumeration contains all operators supported by OPC UA (Equals, IsNull, GreaterThan, LessThan, GreaterThanOrEqual, LessThanOrEqual, Like, Not, Between, InList, And, Or, Cast, InView, OfType, RelatedTo, BitwiseAnd, BitwiseOr). The UAFilterOperandCollection is a collection of content filter operands (UAFilterOperand).

The UAFilterOperand is a base class for four different types of filter operands: UAAttributeOperand (an operand consisting of an attribute of a node in a type, with optional relative path and alias), UAContentFilterElement (essentially, an operator with operands), UALiteralOperand (an operand consisting of a literal value), or UASimpleAttributeOperand (an operand consisting of an attribute of a node in a type, with optional simple relative path, and no alias).

In order to make the creation of Where clauses appear shorter in the code, a UAContentFilterElementBuilder class is provided. Instances of this builder can be implicitly converted to the content filter element itself (hence you can create just the UAContentFilterElementBuilder objects, and use them at all places where the UAContentFilterElement object is expected). The builder contains an operator and is a collection of operands; you can use the C# collection initializer to specify the operands.

For further simplification of the coding, the UAFilterElements static class provides an easy way to construct basic filter elements (literal, attribute, or a simple attribute), and elements with various operators. Using the UAFilterElements class is for most cases the recommended way to create the Where clause of the event filter. For example, UAFilterElements.GreaterThanOrEqual creates a Where clause with the '>=' operator. There are methods for each of the OPC UA operators, with various overloads. The UAFilterElements.Attribute, UAFilterElements.Literal and UAFilterElements.SimpleAttribute methods can be used to create other types of filter elements. Moreover, useful overloads exist that assure that you can actually use .NET objects directly as values of operands, and they will be interpreted as literal elements. With use of all these code simplifying features, you can write, for example, the following Where clause:

```
UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500).
```

C#

```
// This example shows how to specify criteria for event notifications.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class WhereClause
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                new UAEventFilterBuilder(
                    // Either the severity is >= 500, or the event comes from a specified source node
                    UAFilterElements.Or(
                        UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500),
                        UAFilterElements.Equals(
                            UABaseEventObject.Operands.SourceNode,
                            new
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor"))),
                    UABaseEventObject.AllFields));

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }
```

```
        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            // Display the event
            Console.WriteLine(e);
        }
    }
}
```

Object Pascal

```
// This example shows how to specify criteria for event notifications.

type
  THelperMethods10 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString: string):
_UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_AllFields: _UAAttributeFieldCollection; static;
  end;

type
  TClientEventHandlers10 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers10.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
  WriteLn(eventArgs.ToString);
end;

class procedure WhereClause.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers10;
  Element1, Element2: _UAContentFilterElement;
  EndpointDescriptor: string;
  EventFilter: _UAEventFilter;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  Operand1, Operand3: _UASimpleAttributeOperand;
  Operand2, Operand4: _UALiteralOperand;
  SourceNodeId: _UANodeId;
  WhereClause: _UAContentFilterElement;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers10.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');

  WhereClause := CoUAContentFilterElement.Create;

  // Either the severity is >= 500, or the event comes from a specified source node
  Operand1 := THelperMethods10.UABaseEventObject_Operands_Severity;
  Operand2 := CoUALiteralOperand.Create;
```

```
  Operand2.Value := 500;
  Element1 := CoUAContentFilterElement.Create;
  Element1.FilterOperator := UAFilterOperator_GreaterThanOrEqual;
  Element1.FilterOperands.Add(Operand1);
  Element1.FilterOperands.Add(Operand2);
  Operand3 := THelperMethods10.UABaseEventObject_Operands_SourceNode;
  SourceNodeId := CoUANodeId.Create;
  SourceNodeId.ExpandedText := 'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor';
  Operand4 := CoUALiteralOperand.Create;
  Operand4.Value := SourceNodeId;
  Element2 := CoUAContentFilterElement.Create;
  Element2.FilterOperator := UAFilterOperator_Equals;
  Element2.FilterOperands.Add(Operand3);
  Element2.FilterOperands.Add(Operand4);
  WhereClause.FilterOperator := UAFilterOperator_Or;
  WhereClause.FilterOperands.Add(Element1);
  WhereClause.FilterOperands.Add(Element2);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := THelperMethods10.UABaseEventObject_AllFields;
  EventFilter.WhereClause := WhereClause;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;


class function THelperMethods10.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods10.UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames := BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods10.UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
```

```
end;

class function THelperMethods10.UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
end;

class function THelperMethods10.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods10.UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods10.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods10.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;

class function THelperMethods10.UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
end;

class function THelperMethods10.UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
end;

class function THelperMethods10.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods10.UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;

class function THelperMethods10.UABaseEventObject_AllFields: _UAAttributeFieldCollection;
var
  Fields: _UAAttributeFieldCollection;
begin
  Fields := CoUAAttributeFieldCollection.Create;
  Fields.Add(UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventType.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Time.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_LocalTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Message.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Severity.ToUAAttributeField);

  Result := Fields;
end;
```

PHP

```
// This example shows how to specify criteria for event notifications.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

const UAFilterOperator_Equals = 1;
```

```
const UAFilterOperator_GreaterThanOrEqual = 5;
const UAFilterOperator_Or = 12;

const UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';

class ClientEvents {
    function EventNotification($Sender, $E)
    {
        // Display the event
        printf("%s\n", $E);
    }
}

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

$WhereClause = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");

// Either the severity is >= 500, or the event comes from a specified source node
$Operand1 = UABaseEventObject_Operands_Severity();
$Operand2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand2->Value = 500;
$Element1 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$Element1->FilterOperator = UAFilterOperator_GreaterThanOrEqual;
$Element1->FilterOperands->Add($Operand1);
$Element1->FilterOperands->Add($Operand2);
$Operand3 = UABaseEventObject_Operands_SourceNode();
$SourceNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$SourceNodeId->ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor";
$Operand4 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand4->Value = $SourceNodeId;
$Element2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$Element2->FilterOperator = UAFilterOperator_Equals;
$Element2->FilterOperands->Add($Operand3);
$Element2->FilterOperands->Add($Operand4);
$WhereClause->FilterOperator = UAFilterOperator_Or;
$WhereClause->FilterOperands->Add($Element1);
$WhereClause->FilterOperands->Add($Element2);

$EventFilter = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter->SelectClauses = UABaseEventObject_AllFields();
$EventFilter->WhereClause = $WhereClause;

$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoringParameters->EventFilter = $EventFilter;
$MonitoringParameters->QueueSize = 1000;

$MonitoredItemArguments = new COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments->NodeDescriptor->NodeId->StandardName = "Server";
$MonitoredItemArguments->MonitoringParameters = $MonitoringParameters;
/$MonitoredItemArguments->SubscriptionParameters->PublishingInterval = 0;
$MonitoredItemArguments->AttributeId = UAAttributeId_EventNotifier;

$arguments[0] = $MonitoredItemArguments;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Processing monitored item changed events for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}
```

```
function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_EventId() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventId");
}

function UABaseEventObject_Operands_EventType() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventType");
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}

function UABaseEventObject_Operands_ReceiveTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/ReceiveTime");
}

function UABaseEventObject_Operands_LocalTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/LocalTime");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}

function UABaseEventObject_AllFields() {
    $Fields = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
    $Fields->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventType()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_ReceiveTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_LocalTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField);

    return $Fields;
}
```

VB.NET

```
' This example shows how to specify criteria for event notifications.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel
```

```
Namespace UADocExamples.AlarmsAndConditions
    Friend Class WhereClause
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            ' Either the severity is >= 500, or the event comes from a specified source node
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000, _
                New UAEventFilterBuilder( _
                    UAFilterElements.Or( _
                        UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500),
                        UAFilterElements.Equals( _
                            UABaseEventObject.Operands.SourceNode,
                            New
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor"))),
                    UABaseEventObject.AllFields))

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As EasyUAEventNotificationEventArgs)
            ' Display the event
            Console.WriteLine(e)
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to specify criteria for event notifications.

type
  TClientEventHandlers3 = class
    procedure OnEventNotification(
      Sender: TObject;
      sender0: OleVariant;
      eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers3.OnEventNotification(
  Sender: TObject;
  sender0: OleVariant;
  eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
  WriteLn(eventArgs.ToString);
end;

class procedure WhereClause.Main;

  function ToUAAttributeField(Operand: UASimpleAttributeOperand): UAAttributeField;
  var
    AttributeField: UAAttributeField;
  begin
    AttributeField := CoUAAttributeField.Create;
    AttributeField.Operand := Operand;
    ToUAAttributeField := AttributeField;
  end;

  function ObjectTypeIds_BaseEventType: UANodeId;
  var
    NodeId: UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    ObjectTypeIds_BaseEventType := NodeId;
```

```
  end;

  function UAFilterElements_SimpleAttribute(TypeId: UANodeId; SimpleRelativeBrowsePathString: string):
UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
    BrowsePathParser: UABrowsePathParser;
  begin
    BrowsePathParser := CoUABrowsePathParser.Create;
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId := TypeId;
    Operand.QualifiedNames := BrowsePathParser.ParseRelative(SimpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
    UAFilterElements_SimpleAttribute := Operand;
  end;

  function UABaseEventObject_Operands_NodeId: UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
  begin
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId.StandardName := 'BaseEventType';
    Operand.AttributeId := UAAttributeId_NodeId;
    UABaseEventObject_Operands_NodeId := Operand;
  end;

  function UABaseEventObject_Operands_EventId: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_EventId := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
  end;

  function UABaseEventObject_Operands_EventType: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_EventType := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
  end;

  function UABaseEventObject_Operands_SourceNode: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_SourceNode := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/SourceNode');
  end;

  function UABaseEventObject_Operands_SourceName: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_SourceName := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/SourceName');
  end;

  function UABaseEventObject_Operands_Time: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_Time := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
  end;

  function UABaseEventObject_Operands_ReceiveTime: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_ReceiveTime := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/ReceiveTime');
  end;

  function UABaseEventObject_Operands_LocalTime: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_LocalTime := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
  end;

  function UABaseEventObject_Operands_Message: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_Message := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
  end;

  function UABaseEventObject_Operands_Severity: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_Severity := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
  end;

  function UABaseEventObject_AllFields: UAAttributeFieldCollection;
  var
    Fields: UAAttributeFieldCollection;
  begin
    Fields := CoUAAttributeFieldCollection.Create;
```

```
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_NodeId));

    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_EventId));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_EventType));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_SourceNode));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_SourceName));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_Time));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_ReceiveTime));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_LocalTime));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_Message));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_Severity));

    UABaseEventObject_AllFields := Fields;
  end;

var
  Arguments: OleVariant;
  EvsClient: TEvsEasyUAClient;
  Client: EasyUAClient;
  ClientEventHandlers: TClientEventHandlers3;
  Handle: Cardinal;
  HandleArray: OleVariant;
  MonitoredItemArguments: EasyUAMonitoredItemArguments;
  MonitoringParameters: UAMonitoringParameters;
  EventFilter: UAEventFilter;
  WhereClause: UAContentFilterElement;
  Operand1: UASimpleAttributeOperand;
  Operand2: UALiteralOperand;
  Operand3: UASimpleAttributeOperand;
  Operand4: UALiteralOperand;
  SourceNodeId: UANodeId;
  Element1, Element2: UAContentFilterElement;
  BaseEventType: UANodeId;
begin
  // Instantiate the client object and hook events
  EvsClient := TEvsEasyUAClient.Create(nil);
  Client := EvsClient.ComServer;
  ClientEventHandlers := TClientEventHandlers3.Create;
  EvsClient.OnEventNotification := @ClientEventHandlers.OnEventNotification;

  WriteLn('Subscribing...');

  WhereClause := CoUAContentFilterElement.Create;
  BaseEventType := CoUaNodeId.Create;
  BaseEventType.StandardName := 'BaseEventType';

  // Either the severity is >= 500, or the event comes from a specified source node
  Operand1 := UABaseEventObject_Operands_Severity;
  Operand2 := CoUALiteralOperand.Create;
  Operand2.Value := 500;
  Element1 := CoUAContentFilterElement.Create;
  Element1.FilterOperator := UAFilterOperator_GreaterThanOrEqual;
  Element1.FilterOperands.Add(Operand1);
  Element1.FilterOperands.Add(Operand2);
  Operand3 := UABaseEventObject_Operands_SourceNode;
  SourceNodeId := CoUANodeId.Create;
  SourceNodeId.ExpandedText := 'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor';
  Operand4 := CoUALiteralOperand.Create;
  Operand4.Value := SourceNodeId;
  Element2 := CoUAContentFilterElement.Create;
  Element2.FilterOperator := UAFilterOperator_Equals;
  Element2.FilterOperands.Add(Operand3);
  Element2.FilterOperands.Add(Operand4);
  WhereClause.FilterOperator := UAFilterOperator_Or;
  WhereClause.FilterOperands.Add(Element1);
  WhereClause.FilterOperands.Add(Element2);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := UABaseEventObject_AllFields;
  EventFilter.WhereClause := WhereClause;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';
```

```
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(PSafeArray(TVarData(Arguments).VArray)));

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Done...');
end;
```

5.1.4.4.3 Changing Existing Subscription (OPC UA Alarms & Conditions)

It is not necessary to unsubscribe and then subscribe again if you want to change parameters of existing subscription, such as its sampling interval, or the event filter. Instead, change the parameters by calling the ChangeMonitoredItemSubscription method, passing it the subscription handle, and the new parameters.

For changing parameters of multiple subscriptions in an efficient manner, call the ChangeMultipleMonitoredItemSubscriptions method.

5.1.4.4.4 Unsubscribing from OPC UA Events

If you no longer want to receive event notifications, you need to unsubscribe from them. To unsubscribe from a single monitored item, call the UnsubscribeMonitoredItem method, passing it the (event) subscription handle. To unsubscribe from events in an efficient manner, call the UnsubscribeMultipleMonitoredItems method (instead of calling UnsubscribeMonitoredItem in a loop), passing it an array of (event) subscription handles.

You can unsubscribe from all monitored items you have previously subscribed to (on the same instance of EasyUAClient object) by calling the UnsubscribeAllMonitoredItems method.

5.1.4.4.5 Refreshing Condition States (OPC UA Alarms & Conditions)

The UAMonitoredItemArguments class contains a Boolean AutoConditionRefresh property. When set (this is the default), a UA ConditionRefresh will be automatically performed when needed to keep the condition information up-to-date.

The component handles special events reserved for condition refresh automatically, and translates them to event notifications with a special form of event arguments. Event notifications (EasyUAEventNotificationArgs) that related to condition refresh have their Refresh property set to 'true'. In addition, a start of condition refresh is indicated by a

notification with event arguments that have their RefreshInitiated property set to 'true'. Correspondingly, an end of condition refresh is indicated by a notification with event arguments that have their RefreshComplete property set to 'true'.

5.1.4.4.6 OPC UA A&C Notification Event

When an OPC UA Alarms & Conditions server generates an event, the EasyUAClient object generates an EventNotification event. For subscription mechanism to be useful, you should hook one or more event handlers to this event. This event is raised for every event generated by a subscribed OPC monitored item.

To be more precise, the EventNotification event is actually generated in other cases, too - if there is any significant occurrence related to the event subscription. This can be for three reasons:

1. You receive the EventNotification when a successful connection (or re-connection) is made. In this case, the Exception and EventData properties of the event arguments are null references.
2. You receive the EventNotification when there is a problem with the event subscription, and it is disconnected. In this case, the Exception property contains information about the error. The EventData property is a null reference.
3. You receive one additional EventNotification when the component is about to send you notifications for condition refresh, and one additional EventNotification after the component has sent you all notifications for the condition refresh. In these cases, the RefreshInitiated or RefreshComplete property of the event arguments is set to 'true', and the Exception and EventData properties contain null references.

The notification for the EventNotification event contains an EasyUAEventNotificationEventArgs argument. You will find all kind of relevant data in this object. Some properties in this object contain valid information under all circumstances - e.g. Arguments (including Arguments.State). Other properties, such as EventData, contain null references when there is no associated information for them. When the EventData property is not a null reference, it contains a UAEventData object describing the detail of the actual OPC event received from the OPC UA Alarms & Conditions server.

Before further processing, your code should always inspect the value of Exception property of the event arguments. If this property is not a null reference, there has been an error related to the event subscription, the Exception property contains information about the problem, and the EventData property does not contain a valid object.

If the Exception property is a null reference, the notification may be informing you about the fact that a condition refresh is being initiated or is complete (in this case, the RefreshInitiated or RefreshComplete property is 'true'), or that an event subscription has been successfully connected or re-connected (in this case, the EventData property is a null reference). If none of the previous applies, the EventData property contains a valid UAEventData object with details about the actual OPC event generated by the OPC server.

Pseudo-code for the full EventNotification event handler may look similar to this:

> **if** notificationEventArgs.Exception is not null **then**
>
> *An error occurred and the subscription is disconnected, handle it (or ignore)*
>
> **else if** notificationEventArgs.RefreshInitiated **then**
>
> *A "refresh" has been initiated; handle it*
>
> **else if** notificationEventArgs.RefreshComplete **then**
>
> *A "refresh" is complete; handle it*
>
> **else if** notificationEventArgs.EventData is null **then**
>
> *Subscription has been successfully connected or re-connected, handle it (or ignore)*
>
> **else**
>
> *Handle the OPC event, details are in notificationEventArgs.EventData. You may use notificationEventArgs.Refresh flag for distinguishing refreshes from original notifications.*

The EventNotification event handler is called on a thread determined by the EasyUAClient component. For details, please refer to "Multithreading and Synchronization" chapter under "Advanced Topics".

The arguments of each event notification (EasyUAEventNotificationEventArgs) have an EventData property (of type UAEventData, described further below), which contains details about the OPC UA event. In addition, common information such as the copy of the input arguments to the subscription, i.e. identification of the endpoint and node (monitored item) which has generated the event, is included. The Exception property (and several related properties) indicates success (when it is a null reference), or a failure (in which case it contains an exception object describing the error).

The UAEventData object contains details about an OPC UA event, i.e. the information that the OPC UA server sends with the event notification. Primarily, it has a FieldResults property, which is a dictionary containing results for each of the fields in the select clauses of the event filter. Each field results is a ValueResult object, i.e. it can either contain the actual value, or an error indication.

C#

```
// This example shows how to display all fields of incoming events, or extract specific fields.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
```

```
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class FieldResults
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            Console.WriteLine();

            // Display the event
            if (e.EventData == null)
            {
                Console.WriteLine(e);
                return;
            }
            Console.WriteLine("All fields:");
            foreach (KeyValuePair<UAAttributeField, ValueResult> pair in e.EventData.FieldResults)
            {
                UAAttributeField attributeField = pair.Key;
                ValueResult valueResult = pair.Value;
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult);
            }

            // Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}",
                e.EventData.FieldResults[UABaseEventObject.Operands.SourceName]);

            // Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}",
                e.EventData.FieldResults[UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")]);
        }



        // Example output (truncated):
        //Subscribing...
        //Processing event notifications for 30 seconds...
        //
        //[] Success
        //
        //[] Success; Refresh; RefreshInitiated
        //
        //All fields:
        //  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
        //  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
        //  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
        //  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
        //  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
        //  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
        //  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
```

```
        //Source name: Success; EastTank {System.String}
        //Message: Success; The dialog was activated {System.String}
        //
        //All fields:
        //  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
        //  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType
{OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
        //  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
        //  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
        //  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
        //  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
        //  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
        //Source name: Success; EastTank {System.String}
        //Message: Success; The alarm was acknoweledged. {System.String}
        //
        //...
    }
}
```

Object Pascal

```
// This example shows how to display all fields of incoming events, or extract specific fields.

type
  THelperMethods15 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString: string):
_UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
  end;

type
  TClientEventHandlers15 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers15.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  AttributeField: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  EntryEnumerator: IEnumVARIANT;
  ValueResult: OleVariant;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;

  WriteLn('All fields:');

  EntryEnumerator := eventArgs.EventData.FieldResults.GetEnumerator;
  while (EntryEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    AttributeField := IUnknown(Element.Key) as _UAAttributeField;
    ValueResult := IUnknown(Element.Value) as _ValueResult;
    WriteLn('  ', AttributeField.ToString, ' -> ', ValueResult.ToString);
  end;

  // Extracting specific fields using an event type ID and a simple relative path
  WriteLn('Source name: ',
eventArgs.EventData.FieldResults.Item[THelperMethods15.UABaseEventObject_Operands_SourceName.ToUAAttributeField].ToString);
```

```
  WriteLn('Message: ',
eventArgs.EventData.FieldResults.Item[THelperMethods15.UABaseEventObject_Operands_Message.ToUAAttributeField].ToString);
end;

class procedure FieldResults.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers15;
  EndpointDescriptor: string;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers15.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(EndpointDescriptor, UAObjectIds_Server, 1000);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//
//[] Success
//
///[] Success; Refresh; RefreshInitiated
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The dialog was activated {System.String}
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The alarm was acknoweledged. {System.String}
//
//...

class function THelperMethods15.ObjectTypeIds_BaseEventType: _UANodeId;
```

```
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods15.UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames := BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods15.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods15.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;
```

PHP

```
// This example shows how to display all fields of incoming events, or extract specific fields.


class ClientEvents {
    function EventNotification($Sender, $E)
    {
        printf("\n");

        // Display the event
        if (is_null($E->EventData)) {
            printf("%s\n", $E);
            return;
        }

        printf("All fields:\n");

        foreach ($E->EventData->FieldResults as $Pair)
        {
            $AttributeField = $Pair->Key;
            $ValueResult = $Pair->Value;
            printf("  %s -> %s\n", $AttributeField, $ValueResult);
        }

        // Extracting specific fields using an event type ID and a simple relative path
        printf("Source name: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_SourceName()-
>ToUAAttributeField()));
        printf("Message: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_Message()-
>ToUAAttributeField()));
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeEvent($EndpointDescriptor, UAObjectIds_Server, 1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
```

```
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//
//[] Success
//
//[] Success; Refresh; RefreshInitiated
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The dialog was activated {System.String}
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The alarm was acknoweledged. {System.String}
//
//...
```

VB.NET

```
' This example shows how to display all fields of incoming events, or extract specific fields.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
```

```
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class FieldResults
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000)

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As EasyUAEventNotificationEventArgs)
            Console.WriteLine()

            ' Display the event
            If e.EventData Is Nothing Then
                Console.WriteLine(e)
                Exit Sub
            End If
            Console.WriteLine("All fields:")
            For Each pair In e.EventData.FieldResults
                Dim attributeField = pair.Key
                Dim valueResult = pair.Value
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult)
            Next pair
            ' Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}", _
                e.EventData.FieldResults(UABaseEventObject.Operands.SourceName))
            ' Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}", _
                e.EventData.FieldResults(UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")))
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to display all fields of incoming events, or extract specific fields.

Option Explicit

Const uaObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253"

Dim endpointDescriptor
endpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Client.SubscribeEvent endpointDescriptor, uaObjectIds_Server, 1000

WScript.Echo "Processing event notifications for 30 seconds..."
WScript.Sleep 30*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5*1000
```

```
Function UAFilterElements_SimpleAttribute(TypeId, simpleRelativeBrowsePathString)
    Dim BrowsePathParser: Set BrowsePathParser = CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
    Dim QualifiedNames: Set QualifiedNames =
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNAmeCollection

    Dim SimpleAttributeOperand: Set SimpleAttributeOperand =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Set SimpleAttributeOperand.TypeId.NodeId = TypeId
    Set SimpleAttributeOperand.QualifiedNames = QualifiedNames

    Set UAFilterElements_SimpleAttribute = SimpleAttributeOperand
End Function

Function ObjectTypeIds_BaseEventType
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.StandardName = "BaseEventType"
    Set ObjectTypeIds_BaseEventType = NodeId
End Function

Function UABaseEventObject_Operands_Message
    Set UABaseEventObject_Operands_Message = UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Message")
End Function

Function UABaseEventObject_Operands_SourceName
    Set UABaseEventObject_Operands_SourceName = UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
"/SourceName")
End Function

Sub Client_EventNotification(Sender, e)
    WScript.Echo

    ' Display the event
    If e.EventData Is Nothing Then
        WScript.Echo e
        Exit Sub
    End If
    WScript.Echo "All fields:"
    Dim Pair: For Each Pair In e.EventData.FieldResults
        Dim AttributeField: Set AttributeField = Pair.Key
        Dim ValueResult: Set ValueResult = Pair.Value
        WScript.Echo "  " & AttributeField & " -> " & ValueResult
    Next

    ' Extracting specific fields using an event type ID and a simple relative path
    WScript.Echo "Source name: " &
e.EventData.FieldResults.Item(UABaseEventObject_Operands_SourceName.ToUAAttributeField)
    WScript.Echo "Message: " & e.EventData.FieldResults.Item(UABaseEventObject_Operands_Message.ToUAAttributeField)
End Sub



' Example output (truncated):
'Subscribing...
'Processing event notifications for 30 seconds...
'
'[] Success
'
'[] Success; Refresh; RefreshInitiated
'
'All fields:
'  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
'  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
'  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
'  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
'  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
'  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
'  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
'Source name: Success; EastTank {System.String}
'Message: Success; The dialog was activated {System.String}
'
'All fields:
'  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
```

```
'  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
'  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
'  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
'  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
'  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
'  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
'  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
'Source name: Success; EastTank {System.String}
'Message: Success; The alarm was acknoweledged. {System.String}
'
'...
```

In order to make access to at least some of these results easier (without having to go through indexing an element in the dictionary), the UAEventData object also contains a BaseEvent property (of UABaseEventObject type). The UABaseEventObject class contains properties that directly correspond to information contained in the base event type (from which all UA events are derived), such as SourceName, Time, Message, Severity, etc. In case of an error related to a particular field, the corresponding property will contain its default value (e.g. an empty string for a Message).

C#

```
// This example shows how to display specific fields of incoming events that are derived from base event type.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class BaseEvent
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            Console.WriteLine();

            // Display the event
            if (e.EventData == null)
            {
                Console.WriteLine(e);
                return;
            }
            UABaseEventObject baseEventObject = e.EventData.BaseEvent;
            Console.WriteLine("Source name: {0}", baseEventObject.SourceName);
            Console.WriteLine("Message: {0}", baseEventObject.Message);
            Console.WriteLine("Severity: {0}", baseEventObject.Severity);
        }
    }
}
```

Object Pascal

```
// This example shows how to display specific fields of incoming events that are derived from base event type.

type
  TClientEventHandlers12 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers12.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  BaseEventObject: _UABaseEventObject;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;

  BaseEventObject := eventArgs.EventData.BaseEvent;
  WriteLn('Source name: ', BaseEventObject.SourceName);
  WriteLn('Message: ', BaseEventObject.Message);
  WriteLn('Severity: ', BaseEventObject.Severity);
end;

class procedure BaseEvent.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers12;
  EndpointDescriptor: string;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers12.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(EndpointDescriptor, UAObjectIds_Server, 1000);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to display specific fields of incoming events that are derived from base event type.


class ClientEvents {
    function EventNotification($Sender, $E)
    {
        printf("\n");

        // Display the event
        if (is_null($E->EventData)) {
```

```
            printf("%s\n", $E);
            return;
        }

        $BaseEventObject = $E->EventData->BaseEvent;
        printf("Source name: %s\n", $BaseEventObject->SourceName);
        printf("Message: %s\n", $BaseEventObject->Message);
        printf("Severity: %s\n", $BaseEventObject->Severity);
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeEvent($EndpointDescriptor, UAObjectIds_Server, 1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

VB.NET

```
' This example shows how to display specific fields of incoming events that are derived from base event type.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class BaseEvent
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000)

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As EasyUAEventNotificationEventArgs)
            Console.WriteLine()

            ' Display the event
            If e.EventData Is Nothing Then
                Console.WriteLine(e)
                Exit Sub
            End If
            Dim baseEventObject = e.EventData.BaseEvent
            Console.WriteLine("Source name: {0}", baseEventObject.SourceName)
            Console.WriteLine("Message: {0}", baseEventObject.Message)
            Console.WriteLine("Severity: {0}", baseEventObject.Severity)
        End Sub
    End Class
End Namespace
```

It should be noted that the FieldResults directory always contains results for the fields you have specified in Select clause of the event filter, but it can contain some additional fields as well. This happens when these fields are necessary for internal processing inside the component. Your code should be able to ignore the results that it has not requested.

Note: Some data returned use complex data types defined by OPC UA, which OPC Data Client transforms to its own .NET types. For example, the UATimeZoneData class contains data returned in the UABaseEventObject.LocalTime property (defines the local time that may or may not take daylight saving time into account).

5.1.4.4.7 Using Callback Methods Instead of Event Handler (OPC UA Alarms & Conditions)

The subscription methods also allow you to directly specify the callback method (delegate) to be invoked for each event notification you are subscribing to.

For detailed discussion on this subject, please refer to "Using Callback Methods Instead of Event Handlers" under the "OPC Data Access Tasks" chapter. All information presented there applies to OPC UA Alarms & Conditions as well.

The EasyUAMonitoredItemArguments object has an EventCallback property. This property specifies an optional method to be invoked for each event notification generated by the monitored item.

There is also an IEasyUAClient.SubscribeMonitoredItem extension method overload that accepts an event callback (EasyUAEventNotificationEventHandler).

C#

```
// This example shows how to subscribe to event notifications and display each incoming
event
// using a callback method that is provided as lambda expression.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;

namespace UADocExamples.AlarmsAndConditions
{
    partial class SubscribeEvent
    {
        public static void CallbackLambda()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object
            var client = new EasyUAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the event
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                (sender, eventArgs) => Console.WriteLine(eventArgs));
            // Remark: Production code would check e.Exception before accessing
e.EventData.
```

```
            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 2 seconds...");
            System.Threading.Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to event notifications and display each incoming
event
' using a callback method that is provided as lambda expression.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard

Namespace UADocExamples.AlarmsAndConditions
    Partial Friend Class SubscribeEvent
        Public Shared Sub CallbackLambda()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the event
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                UAObjectIds.Server, _
                1000, _
                Sub(sender, eventArgs) Console.WriteLine(eventArgs))
            ' Remark: Production code would check e.Exception before accessing
e.EventData.

            Console.WriteLine("Processing event notifications for 10 seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

5.1.4.5 Setting Parameters (OPC UA Alarms & Conditions)

As the Alarms & Conditions support for OPC UA uses the same EasyUAClient object, the same properties and objects are used for setting the parameters as for OPC Data. Please refer to the "Setting Parameters" chapter under "Imperative Programming Model for OPC Data (Class and UA)" for all necessary information.

5.1.5 Imperative Programming Model for OPC UA PubSub

This chapter gives guidance in implementing common tasks when dealing with data made available over OPC UA PubSub. You achieve these tasks mainly by calling methods on the EasyUASubscriber object. Depending on what you need, see:

Subscribing to Information (OPC UA PubSub) (Section 5.1.5.1)

- **Subscribing to Information (OPC UA PubSub) (Section 5.1.5.1)**
- **Accessing OPC UA PubSub Configuration Model (Section 5.1.5.2)**
- **OPC UA Publish-Subscribe Client (Section 6.4.3)**

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

OPC UA PubSub is a large specification, and not all its parts are commonly used and have the same significance. OPC Data Client implements the parts that are relevant for its purpose. Some parts OPC UA PubSub functionality is not available in OPC Data Client. The parts that are not (yet) available include:

- Publisher and server functionality.
- JSON message mapping.
- AMQP transport protocol mapping.
- MQTT transport protocol mapping.
- Support for message signing and encryption.
- Custom data types.
- PubSub discovery.
- PubSub security key service model.
- Write access to PubSub configuration.
- Parts of PubSub configuration that relate to subscriber configuration.

> 💡 Most of the OPC UA PubSub examples work with the **OPC UA Demo Publisher (Section 11.5.5)** that is installed with the product, or can be downloaded separately. It supports both Windows and Linux.

5.1.5.1 Subscribing to Information (OPC UA PubSub)

Subscribing to dataset messages

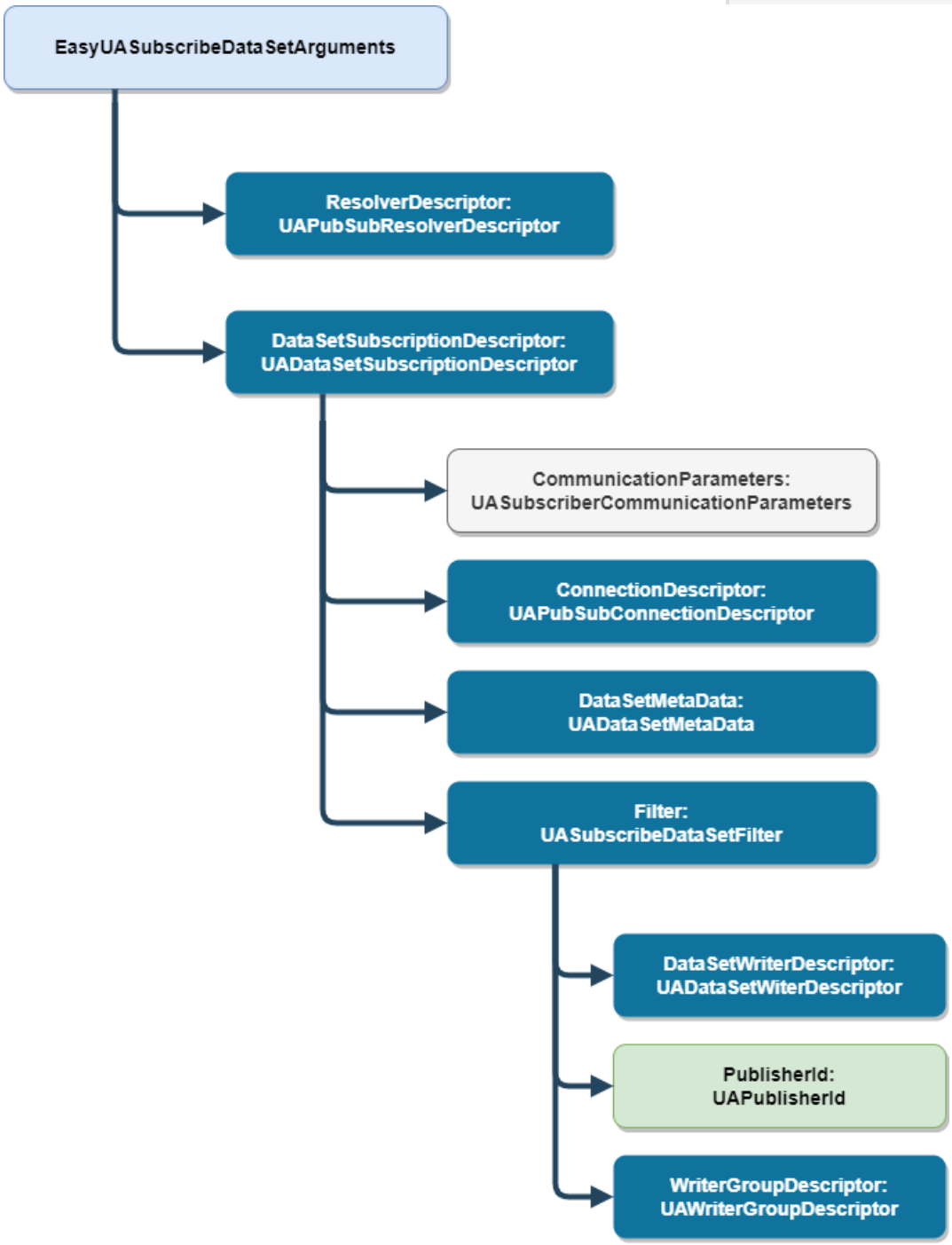
> 💡 For quick tests and experiments with OPC UA PubSub, and especially if you do not want to lose time coding when you need to figure out the proper settings, you can use the **OpcCmd Utility (Section 11.4)** - see Using OpcCmd Utility as OPC UA PubSub Subscriber. The utility allows to specify all parameters of the dataset subscriptions that are accessible from code, but simply by entering them on the command line. It also has a built-in support for displaying the incoming datasets and their fields.

A subscription is initiated by calling the SubscribeDataSet Method on the EasyUASubscriber Class. In its

In This Topic

basic form, the method takes a single argument which is an instance of the EasyUASubscribeDataSetArguments Class, and this argument contains all the information necessary to perform the subscription. The object and all the sub-objects it contains are rather complex.

The picture below is meant to give you at least a general overview about what this object contains. There are more "smaller" properties to the individual objects on the picture, but we do not care about them for now.



You can see that the two main parts of the EasyUASubscribeDataSetArguments are:

- the ResolverDescriptor: Is used for **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** only, and
- the DataSetSubscriptionDescriptor: describes the dataset(s) you want to subscribe to.

Inside the dataset subscription descriptor, you always need to specify the PubSub connection, using the ConnectionDescriptor Property. You will typically also need to fill in data in the Filter Property (see **Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)**), and sometimes provide the DataSetMetaData Property (see **Dataset Metadata (OPC UA PubSub) (Section 5.1.5.1.3)**). The CommunicationParameters Property is used to specify details about the communication, message receive timeout, and parameters specific to message or transport protocol mapping.

> ⚠ When broker transport protocol is in use (such as MQTT or AMQP), you will typically need to specify the broker queue name inside the CommunicationParameters Property.

There are many (extension method) overloads of the SubscribeDataSet Method, with various combinations of arguments. These overloads allow you to bypass the construction of the whole EasyUASubscribeDataSetArguments hierarchy, and pass in simply the parts you need.

The example below subscribes to all dataset messages on an OPC-UA PubSub connection with UDP UADP mapping. The connection is specified by its physical parameters, using the scheme name "opc.udp" and the IP address of the multicast group to listen on.

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Main1()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
```

```
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields:
4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields:
100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //...
    }
}
```

C++

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include "SubscribeDataSet.h"

namespace PubSub
{
    namespace _EasyUASubscriber
    {
        // CEasyUASubscriberEvents

        class CEasyUASubscriberEvents : public IDispEventImpl<1, CEasyUASubscriberEvents>
        {
        public:
            BEGIN_SINK_MAP(CEasyUASubscriberEvents)
                // Event handlers must have the __stdcall calling convention
                SINK_ENTRY(1, DISPID_EASYUASUBSCRIBEREVENTS_DATASETMESSAGE,
&CEasyUASubscriberEvents::DataSetMessage)
            END_SINK_MAP()

        public:
            // The handler for EasyUAClient.DataChangeNotification event
            STDMETHOD(DataSetMessage)(VARIANT varSender, _EasyUADataSetMessageEventArgs* pEventArgs)
            {
```

```
                // Display the dataset.
                if (pEventArgs->Succeeded)
                {
                    _UADataSetDataPtr DataSetDataPtr = pEventArgs->DataSetData;
                    // An event with null DataSetData just indicates a successful connection.
                    if (DataSetDataPtr != NULL)
                    {
                        _tprintf(_T("\n"));
                        _tprintf(_T("Dataset data: %s\n"), (LPCTSTR)CW2CT(DataSetDataPtr->ToString));

                        IEnumVARIANTPtr EnumDictionaryEntry2Ptr = DataSetDataPtr->FieldDataDictionary-
>GetEnumerator();
                        _variant_t vDictionaryEntry2;
                        while (EnumDictionaryEntry2Ptr->Next(1, &vDictionaryEntry2, NULL) == S_OK)
                        {
                            _DictionaryEntry2Ptr DictionaryEntry2Ptr(vDictionaryEntry2);
                            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(DictionaryEntry2Ptr->ToString));
                            vDictionaryEntry2.Clear();
                        }
                    }
                }
                else
                {
                    _tprintf(_T("\n"));
                    _tprintf(_T("*** Failure: %s\n"), (LPCTSTR)CW2CT(pEventArgs->ErrorMessageBrief));
                }
                return S_OK;
            }
        };


        void SubscribeDataSet::Main1()
        {
            // Initialize the COM library
            CoInitializeEx(NULL, COINIT_MULTITHREADED);
            {
                // Prepare arguments for a subscription to the dataset.
                _EasyUASubscribeDataSetArgumentsPtr
SubscribeDataSetArgumentsPtr(__uuidof(EasyUASubscribeDataSetArguments));

                // Define the PubSub connection we will work with.
                _UAPubSubConnectionDescriptorPtr PubSubConnectionDescriptorPtr =
                    SubscribeDataSetArgumentsPtr->DataSetSubscriptionDescriptor->ConnectionDescriptor;
                PubSubConnectionDescriptorPtr->ResourceAddress->ResourceDescriptor->UrlString =
L"opc.udp://239.0.0.1";
                // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
                // the statement below. Your actual interface name may differ, of course.
                //PubSubConnectionDescriptorPtr->ResourceAddress->InterfaceName = L"Ethernet";

                // Instantiate the subscriber object.
                _EasyUASubscriberPtr SubscriberPtr(__uuidof(EasyUASubscriber));

                // Hook events.
                CEasyUASubscriberEvents* pSubscriberEvents = new CEasyUASubscriberEvents();
                AtlGetObjectSourceInterface(SubscriberPtr, &pSubscriberEvents->m_libid, &pSubscriberEvents-
>m_iid,
                    &pSubscriberEvents->m_wMajorVerNum, &pSubscriberEvents->m_wMinorVerNum);
                pSubscriberEvents->m_iid = _uuidof(DEasyUASubscriberEvents);
                pSubscriberEvents->DispEventAdvise(SubscriberPtr, &pSubscriberEvents->m_iid);

                _tprintf(_T("Subscribing...\n"));
                _variant_t vSubscribeDataSetArguments(SubscribeDataSetArgumentsPtr.GetInterfacePtr());
                SubscriberPtr->SubscribeDataSet(vSubscribeDataSetArguments);

                _tprintf(_T("Processing dataset message events for 20 seconds...\n"));
                Sleep(20 * 1000);

                _tprintf(_T("Unsubscribing...\n"));
                SubscriberPtr->UnsubscribeAllDataSets();

                _tprintf(_T("Waiting for 1 second...\n"));
```

```
                // Unsubscribe operation is asynchronous, messages may still come for a short while.
                Sleep(1 * 1000);

                // Unhook events
                pSubscriberEvents->DispEventUnadvise(SubscriberPtr, &pSubscriberEvents->m_iid);

                _tprintf(_T("Finished.\n"));
            }
            // Release all interface pointers BEFORE calling CoUninitialize()
            CoUninitialize();
        }



        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        ////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908,
fields: 4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields:
100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers77 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Main1;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers77;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
```

```
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers77.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers77.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
```

```
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

PHP

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

class SubscriberEvents {
    function DataSetMessage($Sender, $E)
    {
        // Display the dataset.
        if ($E->Succeeded) {
            // An event with null DataSetData just indicates a successful connection.
            printf("\n");
            printf("Dataset data: %s\n", $E->DataSetData);
            foreach ($E->DataSetData->FieldDataDictionary as $Pair)
                printf("%s\n", $Pair);
        }
        else {
            printf("\n");
            printf("*** Failure: %s\n", $E->ErrorMessageBrief);
        }
    }
}



// Define the PubSub connection we will work with.
$SubscribeDataSetArguments = new COM("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments");
$ConnectionDescriptor = $SubscribeDataSetArguments->DataSetSubscriptionDescriptor->ConnectionDescriptor;
$ConnectionDescriptor->ResourceAddress->ResourceDescriptor->UrlString = "opc.udp://239.0.0.1";
// In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
// the statement below. Your actual interface name may differ, of course.
/$ConnectionDescriptor->ResourceAddress->InterfaceName = "Ethernet";

// Instantiate the subscriber object and hook events.
$Subscriber = new COM("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber");
$SubscriberEvents = new SubscriberEvents();
com_event_sink($Subscriber, $SubscriberEvents, "DEasyUASubscriberEvents");

printf("Subscribing...\n");
$Subscriber->SubscribeDataSet($SubscribeDataSetArguments);

printf("Processing dataset message events for 20 seconds...");
$startTime = time();
do {
    com_message_pump(1000);
} while (time() < $startTime + 20);

printf("Unsubscribing...\n");
$Subscriber->UnsubscribeAllDataSets;

printf("Waiting for 1 second...");
// Unsubscribe operation is asynchronous, messages may still come for a short while.
$startTime = time();
do {
    com_message_pump(1000);
} while (time() < $startTime + 1);

printf("Finished.\n");
```

```
// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

Python

```
# This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection, and pull events,
and display
# the incoming datasets.
#
# In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
# http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

import time
import win32com.client

# Define the PubSub connection we will work with.
subscribeDataSetArguments =
win32com.client.Dispatch('OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments')
connectionDescriptor = subscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
connectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.udp://239.0.0.1'
# In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
# the statement below. Your actual interface name may differ, of course.
#connectionDescriptor.ResourceAddress.InterfaceName = 'Ethernet'

# Instantiate the subscriber object.
subscriber = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber')
# In order to use event pull, you must set a non-zero queue capacity upfront.
subscriber.PullDataSetMessageQueueCapacity = 1000

print('Subscribing...')
subscriber.SubscribeDataSet(subscribeDataSetArguments)

print('Processing dataset message events for 20 seconds...')
endTime = time.time() + 20
while time.time() < endTime:
    eventArgs = subscriber.PullDataSetMessage(2*1000)
    if eventArgs is not None:
        # Display the dataset.
        if eventArgs.Succeeded:
            # An event with null DataSetData just indicates a successful connection.
            if eventArgs.DataSetdata is not None:
                print('')
                print('Dataset data: ', eventArgs.DataSetData)
                for pair in eventArgs.DataSetData.FieldDataDictionary:
                    print(pair)
        else:
            print('')
```

```
            print('*** Failure: ', eventArgs.ErrorMessageBrief)

print('Unsubscribing...')
subscriber.UnsubscribeAllDataSets

print('Waiting for 1 second...')
# Unsubscribe operation is asynchronous, messages may still come for a short while.
endTime = time.time() + 1
while time.time() < endTime:
    pass

print('Finished.')



# Example output:
#
#Subscribing...
#Processing dataset message events for 20 seconds...
#
##Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
#[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#
#Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields: 100
#[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#...
```

VB.NET

```
' This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class SubscribeDataSet
        Public Shared Sub Main1()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)
```

```
            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    ''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
    '[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '
    'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields: 100
    '[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '...

End Namespace
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber1 As EasyUASubscriber

Private Sub SubscribeDataSet_Main1_Command_Click()
```

```
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments = New EasyUASubscribeDataSetArguments
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    'ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Instantiate the subscriber object.
    Set Subscriber1 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber1.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber1.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber1 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber1_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

'

VBScript

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with UDP UADP
mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
```

```
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

The example specifies just the PubSub connection, and receives all PubSub messages it encounters. In .NET, it illustrates the use of a SubscribeDataSet Method overload that takes just the PubSub connection descriptor as an input.

Resolved or not?

The information you pass in to the SubscribeDataSet Method may either be used "as is", or it may be subject to **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** first. As explained in **OPC UA PubSub Descriptors (Section 4.12.3)**, you can pass in the physical identifiers, the logical identifiers, or both. The **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** mechanism will be invoked if the dataset subscription descriptor requires resolution (this term is explained in **OPC UA PubSub Descriptors (Section 4.12.3)**).

We can reformulate it so: If you have specified physical identifiers everywhere, they will be directly used, and no logical resolution will take place - even if there are logical identifies filled in as well. If, however, you have a PubSub object somewhere in the descriptors that is only specified by its logical identifier but no physical identifier, the logical resolution mechanism will be used.

Event handlers vs. callback methods

There are two ways your program can be informed about dataset messages that you have subscribed to:

- The EasyUASubscriber invokes event handlers for the DataSetMessage Event. You can hook one or more event handlers to this event.
- You can specify a callback method in the parameters passed to the SubscribeDataSet Method. The callback method is given either by the Callback Property in the EasyUASubscribeDataSetArguments you pass in, or there are extension method overloads to the SubscribeDataSet Method that take a callback parameter directly. The callback method has the same signature as the event handler for the DataSetMessage Event.

It is also possible to specify a non-null callback parameter, and hook an event handler as well. In this case, the callback method will be invoked in addition to the event handlers.

The DataSetMessage event or callback

The event arguments of the DataSetMessage Event are of type EasyUADataSetMessageEventArgs. Because it is indirectly derived from the OperationEventArgs, it contains the Exception Property which is null for success notifications, and non-null in case of failure. You can use the Succeeded Property to make the corresponding test. The actual data of the dataset, if available, is in the DataSetData Property.

The EasyUADataSetMessageEventArgs also contain, in their Arguments Property, a copy of the arguments (EasyUASubscribeDataSetArguments Class) you have used when making the dataset subscription that caused this event or callback. You can use this property to identify the dataset subscription, in case you are using event handlers and have made more subscriptions on the same object, or if you have used the same callback methods with multiple subscriptions. Specifically, the State Property in the EasyUASubscribeDataSetArguments can be used for any information you need to pass from the code that makes the subscription to the code that handles the dataset messages.

The EasyUASubscriber invokes the DataSetMessage event or callback when a new dataset message (you have subscribed to) arrives, but also in some other cases. Below is a list of possible property value combinations and their meaning.

- The Exception Property is null, and the DataSetData Property is also null: Indicates that the PubSub connection (to the message-oriented middleware, depending on the transport protocol) has been successfully established or re-established.
- The Exception Property is null, and the DataSetData Property is not null. A successfully received dataset.
- The Exception Property is not null: Indicates a failure. In this case, the DataSetData Property is always null.

The main causes for failures are

- problems establishing, maintaining or re-establishing the PubSub connection,
- message receive timeouts, and
- inability to decode the message (e.g. due to missing metadata with UADP and RawData encoding).

If the **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** took place, the ResolvedDataSetSubscriptionDescriptor Property is filled in with a dataset subscription descriptor that is derived from the one you have specified when calling the SubscribeDataSet Method, but with the logical identifiers resolved to physical ones.

A typical task when processing the dataset message is to extract one or more fields from the dataset and process them further. The field is identified by its name (or an index in its string form, such "#0", if field names are not available) in the FieldDataDictionary. The example below shows how it can be done.

C#

```
// This example shows how to subscribe to dataset messages and extract data of a specific field.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void ExtractField()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1);

            // Define the metadata, with the use of collection initializer for its fields. For UADP, the order of
field
            // metadata must correspond to the order of fields in the dataset message. If the field names were
contained
            // in the dataset message (such as in JSON), or if we knew the metadata from some other source, this
step would
            // not be needed.
            // Since the encoding is not RawData, we do not have to specify the type information for the fields.
            var metaData = new UADataSetMetaData
            {
                new UAFieldMetaData("BoolToggle"),
                new UAFieldMetaData("Int32"),
                new UAFieldMetaData("Int32Fast"),
                new UAFieldMetaData("DateTime")
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_ExtractField;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
```

```
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_ExtractField(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    // Extract field data, looking up the field by its name.
                    UADataSetFieldData int32FastValueData = e.DataSetData.FieldDataDictionary["Int32Fast"];
                    Console.WriteLine(int32FastValueData);
                }
            }
            else
            {
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
        //6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
        //6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
        //6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
        //6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
        //6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
        //6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
        //6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
        //7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
        //7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
        //7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
        //7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
        //7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
        //7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
        //7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and extract data of a specific field.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub ExtractField()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) Is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
```

```
UAWriterGroupDescriptor.Null, 1)

            ' Define the metadata, with the use of collection initializer for its fields. For UADP, the order of
field
            ' metadata must correspond to the order of fields in the dataset message. If the field names were
contained
            ' in the dataset message (such as in JSON), Or if we knew the metadata from some other source, this
step would
            ' Not be needed.
            ' Since the encoding Is Not RawData, we do Not have to specify the type information for the fields.
            Dim metaData = New UADataSetMetaData From {
                New UAFieldMetaData("BoolToggle"),
                New UAFieldMetaData("Int32"),
                New UAFieldMetaData("Int32Fast"),
                New UAFieldMetaData("DateTime")
            }

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_ExtractField

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_ExtractField(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    ' Extract field data, looking up the field by its name.
                    Dim int32FastValueData = e.DataSetData.FieldDataDictionary("Int32Fast")
                    Console.WriteLine(int32FastValueData)
                End If
            Else
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
    '6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
    '6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
    '6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
    '6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
    '6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
    '6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
    '6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
    '7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
    '7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
    '7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
    '7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
    '7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
    '7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
    '7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
    '...
```

```
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and extract data of a specific field.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers74 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.ExtractField;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Field1, Field2, Field3, Field4: _UAFieldMetaData;
  MetaData: _UADataSetMetaData;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers74;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
  // If the field names were contained in the dataset message (such as in JSON), or if we knew the metadata from
some other
  // source, this step would not be needed.
  // Since the encoding is not RawData, we do not have to specify the type information for the fields.
  MetaData := CoUADataSetMetaData.Create;
  //
  Field1 := CoUAFieldMetaData.Create;
  Field1.Name := 'BoolToggle';
  MetaData.Add(Field1);
  //
  Field2 := CoUAFieldMetaData.Create;
  Field2.Name := 'Int32';
  MetaData.Add(Field2);
  //
  Field3 := CoUAFieldMetaData.Create;
  Field3.Name := 'Int32Fast';
  MetaData.Add(Field3);
  //
  Field4 := CoUAFieldMetaData.Create;
  Field4.Name := 'DateTime';
  MetaData.Add(Field4);
  //
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData := MetaData;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers74.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;
```

```
  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers74.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Dnt32FastValueData: _UADataSetFieldData;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      // Extract field data, looking up the field by its name.
      Dnt32FastValueData := eventArgs.DataSetData.FieldDataDictionary.Item['Int32Fast'];
      WriteLn(Dnt32FastValueData.ToString);
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
//6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
//6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
//6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
//6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
//6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
//6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
//6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
//7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
//7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
//7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
//7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
//7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
//7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
//7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and extract data of a specific field.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit
```

```
Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
' If the field names were contained in the dataset message (such as in JSON), or if we knew the metadata from
some other
' source, this step would not be needed.
' Since the encoding is not RawData, we do not have to specify the type information for the fields.
Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
'
Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field1.Name = "BoolToggle"
MetaData.Add(Field1)
'
Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field2.Name = "Int32"
MetaData.Add(Field2)
'
Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field3.Name = "Int32Fast"
MetaData.Add(Field3)
'
Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field4.Name = "DateTime"
MetaData.Add(Field4)
'
Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            ' Extract field data, looking up the field by its name.
            Dim Int32FastValueData: Set Int32FastValueData = e.DataSetData.FieldDataDictionary.Item("Int32Fast")
            WScript.Echo Int32FastValueData
        End If
```

```
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
'6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
'6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
'6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
'6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
'6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
'6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
'6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
'7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
'7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
'7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
'7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
'7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
'7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
'7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages extract data of a specific field.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber4 As EasyUASubscriber

Private Sub SubscribeDataSet_ExtractField_Command_Click()
    OutputText = ""

    Const UAPublisherIdType_UInt64 = 4

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
    ' If the field names were contained in the dataset message (such as in JSON), or if we knew the metadata from
some other
    ' source, this step would not be needed.
    ' Since the encoding is not RawData, we do not have to specify the type information for the fields.
    Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
    '
    Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field1.Name = "BoolToggle"
    MetaData.Add (Field1)
    '
```

```
    Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field2.Name = "Int32"
    MetaData.Add (Field2)
    '
    Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field3.Name = "Int32Fast"
    MetaData.Add (Field3)
    '
    Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field4.Name = "DateTime"
    MetaData.Add (Field4)
    '
    Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

    ' Instantiate the subscriber object and hook events.
    Set Subscriber4 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber4.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber4.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber4 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber4_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            ' Extract field data, looking up the field by its name.
            Dim Int32FastValueData: Set Int32FastValueData =
EventArgs.DataSetData.FieldDataDictionary.Item("Int32Fast")
            OutputText = OutputText & Int32FastValueData & vbCrLf
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
'6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
'6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
'6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
'6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
'6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
'6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
'6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
'7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
'7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
'7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
'7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
'7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
'7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
'7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
'...
```

Key Frames and Delta Frames

Depending how OPC UA PubSub is configured, the publisher might be always sending all fields of the dataset with every dataset message, or it may use a combination of *key frames* and *delta frames*. While the key frame dataset message includes values for all fields of the dataset, the delta frame only contains values of the fields that have changed since the previous dataset message.

Key frames and delta frames are a concept that exists purely to spare the network bandwidth. When you subscribe with OPC Data Client, the delta frames are interpreted internally, and a full dataset is always shipped to you in the dataset message event or callback. This way, you do not have to put in any additional code to handle the distinction between key frames and delta frames.

> ⚠ When the publisher uses delta frames, there may be a delay before you start receiving the messages (after the connection is established). This is because OPC Data Client needs to wait for the first key frame until it can start delivering the dataset data to you.

Changing existing subscription

It is not necessary to unsubscribe and subscribe again, if you want to change some parameters of existing subscription, such as its filter (**Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)**) or the associated metadata (**Dataset Metadata (OPC UA PubSub) (Section 5.1.5.1.3)**). You change the parameters by calling the ChangeDataSetSubscription Method, passing it the dataset subscription handle, and the new parameters in form of an instance of the EasyUADataSetSubscriptionChangeArguments Class.

There is also an extension method (an overload - with the same name) that allows you to easily pass in just the dataset subscription handle, and a new filter.

Unsubscribing

If you no longer want to receive dataset messages, you need to unsubscribe. This is achieved by calling the UnsubscribeDataSet Method, passing it the dataset subscription handle obtained when calling the SubscribeDataSet Method.

Example:

C#

```
// This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
// dataset.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    class UnsubscribeDataSet
    {
        public static void Main1()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1);

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage;

            Console.WriteLine("Subscribing...");
```

```
            int dataSetHandle = subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeDataSet(dataSetHandle);

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#0, True {System.Boolean}; Good]
        //[#1, 7134 {System.Int32}; Good]
        //[#2, 7364 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#1, 7135 {System.Int32}; Good]
        //[#2, 7429 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#2, 7495 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //[#1, 7135 {System.Int32}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#1, 7136 {System.Int32}; Good]
        //[#2, 7560 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#2, 7626 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //[#1, 7136 {System.Int32}; Good]
        //
        //...
    }
```

```
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
' dataset.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class UnsubscribeDataSet
        Public Shared Sub Main1()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1)

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class



    ' Example output
```

```
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, True {System.Boolean}; Good]
    '[#1, 7134 {System.Int32}; Good]
    '[#2, 7364 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7135 {System.Int32}; Good]
    '[#2, 7429 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7495 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7135 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7136 {System.Int32}; Good]
    '[#2, 7560 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7626 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7136 {System.Int32}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
// dataset.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers82 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure UnsubscribeDataSet.Main1;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers82;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
```

```
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers82.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers82.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, True {System.Boolean}; Good]
//[#1, 7134 {System.Int32}; Good]
//[#2, 7364 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#1, 7135 {System.Int32}; Good]
//[#2, 7429 {System.Int32}; Good]
```

```
//[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 7495 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//[#1, 7135 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#1, 7136 {System.Int32}; Good]
//[#2, 7560 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 7626 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//[#1, 7136 {System.Int32}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
Rem dataset.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Dim dataSetHandle: dataSetHandle = Subscriber.SubscribeDataSet(SubscribeDataSetArguments)

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeDataSet dataSetHandle

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."
```

```
Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
Rem dataset.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber11 As EasyUASubscriber

Private Sub UnsubscribeDataSet_Main1_Command_Click()
    OutputText = ""
```

```
    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Instantiate the subscriber object and hook events.
    Set Subscriber11 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber11.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber11.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber11 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber11_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As
EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
```

```
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
'
```

You can also unsubscribe from all dataset messages you have previously subscribed to (on the same instance of the EasyUASubscriber Class) by calling the UnsubscribeAllDataSets Method.

Subscriptions are optimized

OPC Data Client merges together requests to the same PubSub objects. If, for example, you make multiple dataset subscriptions and they all use the same PubSub connection, OPC Data Client will only make the connection once.

This merging is performed among all subscriptions on all EasyUASubscriber objects that have their Isolated Property set to 'false' (the default). If you set the Isolated Property on some EasyUASubscriber object to 'true', the merging is then performed on the subscriptions made on that object only.

5.1.5.1.1 Subscriber Communication Parameters (OPC UA PubSub)

The subscriber communication parameters specify details of how the OPC UA PubSub communication is performed. As opposed to the PubSub connection descriptor, which contains parameters needed to establish and maintain the connection to the message-oriented middleware, the subscriber communication parameters are more concerned with the actual communication that takes place after the PubSub connection is established.

The subscriber communication parameters are represented by the UASubscriberCommunicationParameters Class.

There are various communication parameters that can be set, and whether dome of them are used also depends on whether broker-based middleware, or broker-less model is in use.

With broker-based middleware, you will typically need to set the broker queue name in the subscriber communication parameters (unless the logical resolution process provides this information for you). This is done inside the BrokerDataSetReaderTransportParameters Property, which contains sub-parameters used with broker-based middleware. The QueueName Property could be the name of a queue or topic defined in the broker (depending on the terminology used by the protocol in use).

The setting of the broker queue name is illustrated in the example below.

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
MQTT UADP mapping using
// TCP.
//
// The OpcLabs.MqttNet assembly needs to be referenced in your project (or otherwise made available,
together with its
// dependencies) for the MQTT transport to work. Refer to the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
```

```
using OpcLabs.EasyOpc.UA.PubSub.Engine;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void MqttUadpTcp()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a
string.
            // Default port with MQTT is 1883.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "mqtt://opcua-pubsub.demo-
this.com";
            // Specify the transport protocol mapping.
            pubSubConnectionDescriptor.TransportProfileUriString =
UAPubSubTransportProfileUriStrings.MqttUadp;

            // Define the arguments for subscribing to the dataset, specifying the MQTT topic name.
            var subscribeDataSetArguments = new
UASubscribeDataSetArguments(pubSubConnectionDescriptor)
            {
                DataSetSubscriptionDescriptor = {CommunicationParameters =
{BrokerDataSetReaderTransportParameters =
                {
                    QueueName = "opcuademo/uadp/none"
                }}}
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_MqttUadpTcp;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_MqttUadpTcp(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
```

```
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessage}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
        //[#0, False {System.Boolean}; Good]
        //[#1, 6685 {System.Int32}; Good]
        //[#2, 1444 {System.Int32}; Good]
        //[#3, 1/4/2020 6:06:20 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
        //[#0, 85 {System.Int64}; Good]
        //[#1, 185 {System.Int64}; Good]
        //[#2, 285 {System.Int64}; Good]
        //[#3, 385 {System.Int64}; Good]
        //[#4, 485 {System.Int64}; Good]
        //[#5, 585 {System.Int64}; Good]
        //[#6, 685 {System.Int64}; Good]
        //[#7, 785 {System.Int64}; Good]
        //[#8, 885 {System.Int64}; Good]
        //[#9, 985 {System.Int64}; Good]
        //[#10, 1085 {System.Int64}; Good]
        //...
    }
}
```

In some situations, more communication parameters need to be set - for example, when JSON message mapping is used, and the automatic format recognition provided by OPC Data Client is turned off, such as in the example below.

C#

```
// This example shows how to set parameters specific to JSON message mapping.
//
// The following assemblies need to be referenced in your project (or otherwise made available,
together with their
// dependencies) for the MQTT transport and JSON message mapping to work.
// - OpcLabs.MqttNet
// - OpcLabs.UAPubSubJson
// - Newtonsoft.Json
// Refer to the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.Engine;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void MappingParameters()
        {
```

```
            // Define the PubSub connection we will work with. Uses implicit conversion from a
string.
            // Default port with MQTT is 1883.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "mqtt://opcua-pubsub.demo-
this.com";
            // Specify the transport protocol mapping.
            pubSubConnectionDescriptor.TransportProfileUriString =
UAPubSubTransportProfileUriStrings.MqttJson;

            // Set a custom property on the PubSub connection that influences how the JSON parsing
works.
            // We are instructing the message parser to turn off the automatic recognition of
message format.
            // For more details, see http://kb.opclabs.com/OPC_UA_PubSub_JSON_mapping_package .
            pubSubConnectionDescriptor.CustomPropertyValueDictionary[new
UAQualifiedName("http://opclabs.com/OpcUA/PubSub",

"OpcLabs.UAPubSubJson.JsonReceiveMessageMapping.MessageParsingParameters.AutoRecognizeMessageFormat")]
=
                false;

            // Define the arguments for subscribing to the dataset.
            var subscribeDataSetArguments = new
UASubscribeDataSetArguments(pubSubConnectionDescriptor)
            {
                DataSetSubscriptionDescriptor =
                {
                    CommunicationParameters =
                    {
                        BrokerDataSetReaderTransportParameters = {QueueName = "opcuademo/json"},
                        // We must set the DataSetFieldContentMask when the format auto-recognition
is turned off.
                        DataSetFieldContentMask = UADataSetFieldContentMask.RawData,
                        JsonDataSetReaderMessageParameters =
                        {
                            // We must set the DataSetMessageContentMask when the format auto-
recognition is turned off.
                            DataSetMessageContentMask =
                                UAJsonDataSetMessageContentMask.DataSetWriterId |
                                UAJsonDataSetMessageContentMask.SequenceNumber |
                                UAJsonDataSetMessageContentMask.Status,
                            // We must set the NetworkMessageContentMask when the format auto-
recognition is turned off.
                            NetworkMessageContentMask =
                                UAJsonNetworkMessageContentMask.NetworkMessageHeader |
                                UAJsonNetworkMessageContentMask.DataSetMessageHeader |
                                UAJsonNetworkMessageContentMask.PublisherId
                        }
                    },
                    Filter =
                    {
                        DataSetWriterDescriptor = 1,
                    }
                }
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_MappingParameters;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);
```

```
            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_MappingParameters(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessage}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=[String]30, writer=1, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 9034 {System.Int64}; Good]
        //[Int32Fast, 1751 {System.Int64}; Good]
        //[DateTime, 1/30/2020 5:23:11 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=[String]30, writer=1, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 9036 {System.Int64}; Good]
        //[Int32Fast, 2526 {System.Int64}; Good]
        //[DateTime, 1/30/2020 5:23:13 PM {System.DateTime}; Good]
        //...
    }
}
```

5.1.5.1.2 Message Filtering (OPC UA PubSub)

By message filtering in OPC UA PubSub, you specify which dataset message should be delivered to your application's code. Parameters that influence the message filtering are part of the dataset subscription descriptor (see **Subscribing to Information (OPC UA PubSub) (Section 5.1.5.1)**).

The various filtering parameters always have a default value that means "no filtering" (on the corresponding piece of message of data). By leaving them all at their defaults, you effectively specify "no filtering whatsoever". You can also set just some of these parameters to the values you are interested in, or you can set all of them.

You need to make a conscious choice about how broad or narrow your filtering will be. In most cases, it is best to define the filter as narrow as possible: Specify the precise publisher, writer group and/or dataset writer etc. that provides the dataset your are interested in. You can then write code (event or callback handlers) that already knows precisely which dataset is coming in, and interpret its contents reliably and easily. Sometimes, however, it makes sense to use a broader filter, and write an event or callback handler that deals with multiple incoming datasets in a generic way. The proper choice of the filter depends on the nature of your project.

In This Topic

> For quick tests and experiments with OPC UA PubSub, and especially if you do not want to lose time coding when you need to figure out the proper settings, you can use the **OpcCmd Utility (Section 11.4)** - see Using OpcCmd Utility as OPC UA PubSub Subscriber. The utility allows to specify all parameters of the dataset subscriptions that are accessible from code, but simply by entering them on the command line. It also has a built-in support for displaying the incoming datasets and their fields.

Subscribe Dataset Filter

In its basic form, the subscribe dataset filter resides in the Filter Property of the UADataSetSubscriptionDescriptor Class, and the dataset subscription descriptor is part of the EasyUASubscribeDataSetArguments that are passed to the SubscribeDataSet Method. There are, however, several extension methods that allow you to make dataset subscription and specify the subscribe dataset filter in a different, easier way, depending on which parts you want to pass in.

Leaving any of the properties of the subscribe dataset filter at its default (initial) value means that no filtering is performed on that piece of information.

The most important parts of the subscribe dataset filter are below.

- PublisherId Property. Specifies the (optional) publisher Id filtering.
- WriterGroupDescriptor Property. Specifies the (optional) writer group filtering.
- DataSetWriterDescriptor Property. Specifies the (optional) dataset writer filtering.

You can also select whether you are interested in data, events, or both (the default), with the properties below.

- AllowDataMessages Property. Determines whether the filter allows (variable) data messages through.
- AllowEventMessages Property. Determines whether the filter allows event messages through.

It is also possibly to specify that you only want to receive dataset messages in which selected fields have certain value. To do, you set the required value under the key (that is a filed name or string form of a filed index) into the dictionary contained in the RequiredFieldValueDictionary Property. The component may, under some circumstances, you the mechanism of so-called promoted fields to achieve the filtering soon in the processing pipeline. This is, however, an implementation detail, and when specifying required field values, you do not have to be concerned about whether the field is promoted or not.

Other Filters

Depending on the message and transport protocol mapping, there may be additional filtering applied to incoming PubSub messages.

For UADP message mapping, several parameters in UAUadpDataSetReaderMessageParameters Class (UadpDataSetReaderMessageParameters in CommunicationParameters Property of the dataset subscription descriptor) acts as filters as well. They are:

- DataSetClassId Property. Defines a dataset class related filter.
- GroupVersion Property. Defines the expected value in the group version field in the header of the network message.
- NetworkMessageNumber Property. The number of the network message inside a publishing interval in which the dataset message is published.

As with the properties on the subscribe dataset filter, leaving these properties at their default (initial) value denotes "no filtering" on that particular piece of information.

There is also, implicitly, a filtering mechanism based on security level of the PubSub message. The SecurityMode Property in the CommunicationParameters of the dataset subscription descriptor describes the minimum security mode of messages that you want to subscribe to. The default settings allows messages with any security mode through.

Data not in the message

You should not specify filtering on data that is not actually transferred in the PubSub messages. If you do, such messages will be filtered out, and you will not receive any useful data.

What does it actually mean? The OPC UA PubSub specification is very flexible, and it allows various pieces of information associated with the message be included or excluded. This may be because such piece of information is not relevant for the task at hand, and/or because (especially in UADP message mapping which is meant to be very terse) for efficiency reasons, the designer of the systems want to eliminate every unnecessary byte to be transferred.

Which information pieces are included or excluded is usually controlled by various flags in the publisher configuration. Some of them (such as optional timestamps) do not relate to filtering. But for those that appear in the filter, you must follow the rules below:

- If publisher Id is not included in the message, the PublisherId Property of the subscribe dataset filter must be kept at UAPublisherId.Null.
- If writer group Id is not included in the message, the WriterGroupDescriptor Property of the subscribe dataset filter must be UAWriterGroupDescriptor.Null.
- If dataset writer Id is not included in the message, the DataSetWriterDescriptor Property of the subscribe dataset filter must be UAWriterGroupDescriptor.Null.
- If dataset class Id is not included in the message, and UADP message mapping is used, the DataSetClassId Property in the UadpDataSetReaderMessageParameters Property must be kept at Guid.Empty.
- If group version is not included in the message, and UADP message mapping is used, the GroupVersion Property in the UadpDataSetReaderMessageParameters Property must be 0.
- If network message number is not included in the message, and UADP message mapping is used, the NetworkMessageNumber Property in the UadpDataSetReaderMessageParameters Property must be 0.
- If a specific field is not included in the message, it should not appear in the RequiredFieldValueDictionary Property of the subscribe dataset filter (although this is not strictly required).

A typical example: You may have access to the publisher configuration, and you will see the concrete, non-zero dataset writer Id of the dataset you are interested in. If the publisher uses the UADP fixed layout (a commonly used profile), the dataset writer id, even though it appears in the publisher's configuration, is not transmitted in the PubSub messages ("on the wire"). Its purpose is served by knowing a concrete data offset (in the message) where the corresponding data starts. In this case, you should not put the dataset writer Id into the subscribe dataset filter.

If you are using **OPC UA PubSub logical resolution (Section 5.1.5.1.4)**, OPC Data Client obtains the information about information included and excluded from the PubSub configuration (the various flags in the publisher configuration). Based on the publisher's configuration, the logical resolution process automatically leaves out the information not included in the PubSub messages from the filters. You therefore need not to do anything special on your side in this case.

Examples

The example below uses a filter that limits the dataset messages received to messages coming from a specific publisher Id, and for a specific dataset writer Id.

C#

```
// This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
// UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Filter()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(
                UAPublisherId.CreateUInt64(31),
                UAWriterGroupDescriptor.Null,
```

```
            dataSetWriterDescriptor: 1);

        // Instantiate the subscriber object and hook events.
        var subscriber = new EasyUASubscriber();
        subscriber.DataSetMessage += subscriber_DataSetMessage_Filter;

        Console.WriteLine("Subscribing...");
        subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter);

        Console.WriteLine("Processing dataset message events for 20 seconds...");
        Thread.Sleep(20 * 1000);

        Console.WriteLine("Unsubscribing...");
        subscriber.UnsubscribeAllDataSets();

        Console.WriteLine("Waiting for 1 second...");
        // Unsubscribe operation is asynchronous, messages may still come for a short while.
        Thread.Sleep(1 * 1000);

        Console.WriteLine("Finished.");
    }

    static void subscriber_DataSetMessage_Filter(object sender, EasyUADataSetMessageEventArgs e)
    {
        // Display the dataset.
        if (e.Succeeded)
        {
            // An event with null DataSetData just indicates a successful connection.
            if (!(e.DataSetData is null))
            {
                Console.WriteLine();
                Console.WriteLine($"Dataset data: {e.DataSetData}");
                foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                    Console.WriteLine(pair);
            }
        }
        else
        {
            Console.WriteLine();
            Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
        }
    }

    // Example output:
    //
    //Subscribing...
    //Processing dataset message events for 20 seconds...
    //
    //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    //[#0, True {System.Boolean}; Good]
    //[#1, 7134 {System.Int32}; Good]
    //[#2, 7364 {System.Int32}; Good]
    //[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
    //
    //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    //[#1, 7135 {System.Int32}; Good]
    //[#2, 7429 {System.Int32}; Good]
    //[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    //[#0, True {System.Boolean}; Good]
    //
    //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    //[#2, 7495 {System.Int32}; Good]
    //[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    //[#0, True {System.Boolean}; Good]
    //[#1, 7135 {System.Int32}; Good]
    //
    //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    //[#1, 7136 {System.Int32}; Good]
    //[#2, 7560 {System.Int32}; Good]
    //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    //[#0, True {System.Boolean}; Good]
    //
    //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
```

```
        //[#2, 7626 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //[#1, 7136 {System.Int32}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
' UDP UADP mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub Filter()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) Is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1)

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_Filter

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_Filter(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
```

```
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, True {System.Boolean}; Good]
    '[#1, 7134 {System.Int32}; Good]
    '[#2, 7364 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7135 {System.Int32}; Good]
    '[#2, 7429 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7495 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7135 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7136 {System.Int32}; Good]
    '[#2, 7560 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7626 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7136 {System.Int32}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
// UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers76 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Filter;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers76;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
```

```
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers76.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers76.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 25 {System.Int32}; Good]
//[Int32Fast, 928 {System.Int32}; Good]
```

```
//[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32, 26 {System.Int32}; Good]
//[Int32Fast, 1007 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1113 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 26 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//[Int32Fast, 1201 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1260 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
Rem UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000
```

```
WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
Rem UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
```

```
'Public WithEvents Subscriber6 As EasyUASubscriber

Private Sub SubscribeDataSet_Filter_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Instantiate the subscriber object and hook events.
    Set Subscriber6 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber6.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber6.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber6 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber6_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
```

```
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
'
```

The filter in the example below only limits the messages received to those coming from a specific publisher Id.

C#

```
// This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
// with UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void PublisherId()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit)
publisher Id 31.
            var subscribeDataSetArguments = new UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31));

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_PublisherId;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();
```

```
    Console.WriteLine("Waiting for 1 second...");
    // Unsubscribe operation is asynchronous, messages may still come for a short while.
    Thread.Sleep(1 * 1000);

    Console.WriteLine("Finished.");
}

static void subscriber_DataSetMessage_PublisherId(object sender, EasyUADataSetMessageEventArgs e)
{
    // Display the dataset.
    if (e.Succeeded)
    {
        // An event with null DataSetData just indicates a successful connection.
        if (!(e.DataSetData is null))
        {
            Console.WriteLine();
            Console.WriteLine($"Dataset data: {e.DataSetData}");
            foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                Console.WriteLine(pair);
        }
    }
    else
    {
        Console.WriteLine();
        Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
    }
}

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
//[#0, True {System.Boolean}; Good]
//[#1, 1237 {System.Int32}; Good]
//[#2, 2514 {System.Int32}; Good]
//[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, False {System.Boolean}; Good]
//[#1, 1239 {System.Int32}; Good]
//[#2, 2703 {System.Int32}; Good]
//[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[#0, False {System.Boolean}; Good]
//[#1, 215 {System.Byte}; Good]
//[#2, 1239 {System.Int16}; Good]
//[#3, 1239 {System.Int32}; Good]
//[#4, 1239 {System.Int64}; Good]
//[#5, 87 {System.Int16}; Good]
//[#6, 1239 {System.Int32}; Good]
//[#7, 1239 {System.Int64}; Good]
//[#8, 1239 {System.Decimal}; Good]
//[#9, 1239 {System.Single}; Good]
//[#10, 1239 {System.Double}; Good]
//[#11, Romeo {System.String}; Good]
//[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
//[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
//[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 2722 {System.Int32}; Good]
//[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//[#0, False {System.Boolean}; Good]
//[#1, 1239 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
//[#0, 39 {System.Int64}; Good]
//[#1, 139 {System.Int64}; Good]
```

```
        //[#2, 239 {System.Int64}; Good]
        //[#3, 339 {System.Int64}; Good]
        //[#4, 439 {System.Int64}; Good]
        //[#5, 539 {System.Int64}; Good]
        //[#6, 639 {System.Int64}; Good]
        //[#7, 739 {System.Int64}; Good]
        //[#8, 839 {System.Int64}; Good]
        //[#9, 939 {System.Int64}; Good]
        //[#10, 1039 {System.Int64}; Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
' with UDP UADP mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub PublisherId()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the arguments for subscribing to the dataset, where the filter Is (unsigned 64-bit)
publisher Id 31.
            Dim subscribeDataSetArguments = New UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31))

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_PublisherId

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(subscribeDataSetArguments)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_PublisherId(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
```

```
                Next
            End If
        Else
            Console.WriteLine()
            Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
        End If
    End Sub
End Class



' Example output
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 1237 {System.Int32}; Good]
'[#2, 2514 {System.Int32}; Good]
'[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'[#2, 2703 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[#0, False {System.Boolean}; Good]
'[#1, 215 {System.Byte}; Good]
'[#2, 1239 {System.Int16}; Good]
'[#3, 1239 {System.Int32}; Good]
'[#4, 1239 {System.Int64}; Good]
'[#5, 87 {System.Int16}; Good]
'[#6, 1239 {System.Int32}; Good]
'[#7, 1239 {System.Int64}; Good]
'[#8, 1239 {System.Decimal}; Good]
'[#9, 1239 {System.Single}; Good]
'[#10, 1239 {System.Double}; Good]
'[#11, Romeo {System.String}; Good]
'[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
'[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
'[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 2722 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 39 {System.Int64}; Good]
'[#1, 139 {System.Int64}; Good]
'[#2, 239 {System.Int64}; Good]
'[#3, 339 {System.Int64}; Good]
'[#4, 439 {System.Int64}; Good]
'[#5, 539 {System.Int64}; Good]
'[#6, 639 {System.Int64}; Good]
'[#7, 739 {System.Int64}; Good]
'[#8, 839 {System.Int64}; Good]
'[#9, 939 {System.Int64}; Good]
'[#10, 1039 {System.Int64}; Good]
'...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
// with UDP UADP mapping.
```

```
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers80 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.PublisherId;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers80;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers80.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers80.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
```

```
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
//[#0, True {System.Boolean}; Good]
//[#1, 1237 {System.Int32}; Good]
//[#2, 2514 {System.Int32}; Good]
//[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, False {System.Boolean}; Good]
//[#1, 1239 {System.Int32}; Good]
//[#2, 2703 {System.Int32}; Good]
//[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[#0, False {System.Boolean}; Good]
//[#1, 215 {System.Byte}; Good]
//[#2, 1239 {System.Int16}; Good]
//[#3, 1239 {System.Int32}; Good]
//[#4, 1239 {System.Int64}; Good]
//[#5, 87 {System.Int16}; Good]
//[#6, 1239 {System.Int32}; Good]
//[#7, 1239 {System.Int64}; Good]
//[#8, 1239 {System.Decimal}; Good]
//[#9, 1239 {System.Single}; Good]
//[#10, 1239 {System.Double}; Good]
//[#11, Romeo {System.String}; Good]
//[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
//[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
//[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 2722 {System.Int32}; Good]
//[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//[#0, False {System.Boolean}; Good]
//[#1, 1239 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
//[#0, 39 {System.Int64}; Good]
//[#1, 139 {System.Int64}; Good]
//[#2, 239 {System.Int64}; Good]
//[#3, 339 {System.Int64}; Good]
//[#4, 439 {System.Int64}; Good]
//[#5, 539 {System.Int64}; Good]
//[#6, 639 {System.Int64}; Good]
//[#7, 739 {System.Int64}; Good]
//[#8, 839 {System.Int64}; Good]
//[#9, 939 {System.Int64}; Good]
//[#10, 1039 {System.Int64}; Good]
//...
```

VBScript

```
Rem This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
```

```
connection
Rem with UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 1237 {System.Int32}; Good]
```

```
'[#2, 2514 {System.Int32}; Good]
'[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'[#2, 2703 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[#0, False {System.Boolean}; Good]
'[#1, 215 {System.Byte}; Good]
'[#2, 1239 {System.Int16}; Good]
'[#3, 1239 {System.Int32}; Good]
'[#4, 1239 {System.Int64}; Good]
'[#5, 87 {System.Int16}; Good]
'[#6, 1239 {System.Int32}; Good]
'[#7, 1239 {System.Int64}; Good]
'[#8, 1239 {System.Decimal}; Good]
'[#9, 1239 {System.Single}; Good]
'[#10, 1239 {System.Double}; Good]
'[#11, Romeo {System.String}; Good]
'[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
'[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
'[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 2722 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 39 {System.Int64}; Good]
'[#1, 139 {System.Int64}; Good]
'[#2, 239 {System.Int64}; Good]
'[#3, 339 {System.Int64}; Good]
'[#4, 439 {System.Int64}; Good]
'[#5, 539 {System.Int64}; Good]
'[#6, 639 {System.Int64}; Good]
'[#7, 739 {System.Int64}; Good]
'[#8, 839 {System.Int64}; Good]
'[#9, 939 {System.Int64}; Good]
'[#10, 1039 {System.Int64}; Good]
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
Rem with UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber9 As EasyUASubscriber

Private Sub SubscribeDataSet_PublisherId_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"
```

```
    ' Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31

    ' Instantiate the subscriber object and hook events.
    Set Subscriber9 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber9.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber9.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber9 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber9_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 1237 {System.Int32}; Good]
'[#2, 2514 {System.Int32}; Good]
'[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'[#2, 2703 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[#0, False {System.Boolean}; Good]
'[#1, 215 {System.Byte}; Good]
'[#2, 1239 {System.Int16}; Good]
'[#3, 1239 {System.Int32}; Good]
'[#4, 1239 {System.Int64}; Good]
'[#5, 87 {System.Int16}; Good]
'[#6, 1239 {System.Int32}; Good]
'[#7, 1239 {System.Int64}; Good]
'[#8, 1239 {System.Decimal}; Good]
'[#9, 1239 {System.Single}; Good]
'[#10, 1239 {System.Double}; Good]
'[#11, Romeo {System.String}; Good]
```

```
'[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
'[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
'[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 2722 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 39 {System.Int64}; Good]
'[#1, 139 {System.Int64}; Good]
'[#2, 239 {System.Int64}; Good]
'[#3, 339 {System.Int64}; Good]
'[#4, 439 {System.Int64}; Good]
'[#5, 539 {System.Int64}; Good]
'[#6, 639 {System.Int64}; Good]
'[#7, 739 {System.Int64}; Good]
'[#8, 839 {System.Int64}; Good]
'[#9, 939 {System.Int64}; Good]
'[#10, 1039 {System.Int64}; Good]
'...
'
```

5.1.5.1.3 Dataset Metadata (OPC UA PubSub)

In This Topic

The dataset metadata describes the content and semantics of a dataset in OPC UA PubSub. In OPC Data Client, it is represented by an instance of the UADataSetMetaData Class. When you are subscribing to a dataset, metadata is an optional part of the dataset subscription descriptor (UADataSetSubscriptionDescriptor Class), in its DataSetMetaData Property.

The main part of the dataset metadata are field definitions - a metadata for each of the fields contained in the dataset. This is a collection of UAFieldMetaData Class instances, contained in the collection stored in the Fields Property. Each field metadata contains following data members:

- Name Property. The name of the field.
- Description Property. Description of the field.
- DataSetFieldId Property. The unique ID for the field in the dataset, expressed as a string.
- BuiltInType Property. The built-in data type of the field.
- DataTypeId Property. The node Id of the data type of this field.
- MaximumStringLength Property. Specifies the maximum supported length of a String or ByteString.
- ValueRank Property. Indicates whether the data type is an array and how many dimensions the array has.
- ArrayDimensions Property. Specifies the maximum supported length of each dimension.
- DataSetFieldFlags Property. Flags for the field.

Most of the data members have reasonable defaults, and some are not necessary at all times. The Description Property, the DataSetFieldId Property and the DataSetFieldFlags Property are optional. In the most typical case of scalar fields of built-in types, you can leave the DataTypeId Property, the MaximumStringLength Property and the ValueRank Property at their defaults.

Besides the field metadata, the dataset metadata contains some data members that apply to the dataset as a whole. They are:

- Name Property. Name of the dataset.
- Description Property. Description of the dataset.
- ConfigurationVersion Property. The configuration version for the current configuration of the dataset.
- DataSetClassId Property. Provides the globally unique identifier of the class of dataset if the dataset is based on a dataset class.

Do I need the metadata?

The answer is: It depends. Some message mappings (such as JSON) as largely self-descriptive, meaning that even without the metadata, it is possible to decode the dataset messages and event obtain metadata-like information, such as field names, from the dataset message itself. Other message mappings (such as UADP) are more compact, and not fully (or not at all) self-descriptive. The UADP message mapping does not transfer field names in the messages. Consequently, without metadata, OPC Data Client will deliver to you the datasets with field indexes (such as "#0", "#1", ...) instead of field names - but in general, things will work. If you use the UADP message mapping and the dataset is encoded with the RawData encoding (check with the publisher), you **must** have metadata in order to be able to decode the dataset messages.

Where do I get the metadata from?

There are several (or rather, many) ways of obtaining the metadata, and applying them. Below are some of them.

- With **OPC UA PubSub logical resolution (Section 5.1.5.1.4)**, you let OPC Data Client get the metadata from PubSub configuration residing in a standard OPC UA server, or a PubSub configuration file. No additional coding is needed.
- You may access the PubSub configuration from your code, and then retrieve the published dataset element by calling the FindPublishedDataSetElement Method or the GetPublishedDataSetElement Method, and get the dataset metadata from the DataSetMetaData Property of the UAPublishedDataSetElement Class instance.
- You obtain the metadata information in other form (maybe somebody described it in the document, or you read it in your program from some other file format), and you will write code that constructs the instance of the UADataSetMetaData Class. This is described in more details further below.

Defining metadata in code

In order to define the metadata in your code, create an instance of the UADataSetMetaData Class, and fill in its properties and the Fields collection as necessary. Then, assign this dataset metadata instance to the DataSetMetaData Property of the UADataSetSubscriptionDescriptor Class instance that you will use to make your dataset subscription. There are also some extension method overloads of the SubscribeDataSet Method that take the dataset metadata directly as one of their arguments.

In languages like C# or VB.NET, you can take advantage of the fact that the UADataSetMetaData Class support collection initializer syntax for its field, making the code even shorter. This approach is also shown in the example below.

C#

```
// This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
// for their decoding directly in the code.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Metadata()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
            // The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt16(30), writerGroupDescriptor: 101);

            // Define the metadata, with the use of collection initializer for its fields.
            var metaData = new UADataSetMetaData
            {
                new UAFieldMetaData("BoolToggle", UABuiltInType.Boolean),
                new UAFieldMetaData("Int32", UABuiltInType.Int32),
                new UAFieldMetaData("Int32Fast", UABuiltInType.Int32),
                new UAFieldMetaData("DateTime", UABuiltInType.DateTime)
            };

            // Define the dataset subscription, with specific communication parameters.
            // The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
            // encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
            // which is also the case here, but we are specifying the dataset offset anyway, for illustration.
            var dataSetSubscriptionDescriptor = new UADataSetSubscriptionDescriptor(
                pubSubConnectionDescriptor, filter, metaData)
            {
                CommunicationParameters = { UadpDataSetReaderMessageParameters = { DataSetOffset = 15 }}
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_Metadata;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(dataSetSubscriptionDescriptor);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();
```

```
            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_Metadata(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3072 {System.Int32}; Good]
        //[Int32Fast, 894 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3072 {System.Int32}; Good]
        //[Int32Fast, 920 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3073 {System.Int32}; Good]
        //[Int32Fast, 1003 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3073 {System.Int32}; Good]
        //[Int32Fast, 1074 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 3074 {System.Int32}; Good]
        //[Int32Fast, 1140 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
' for their decoding directly in the code.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
```

```
    Partial Friend Class SubscribeDataSet
        Public Shared Sub Metadata()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
            ' The dataset writer Id (1) must Not be specified in the filter, because it does Not appear in the message.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt16(30))
            filter.WriterGroupDescriptor.WriterGroupId = 101

            ' Define the metadata, with the use of collection initializer for its fields.
            Dim metaData = New UADataSetMetaData From {
                New UAFieldMetaData("BoolToggle", UABuiltInType.Boolean),
                New UAFieldMetaData("Int32", UABuiltInType.Int32),
                New UAFieldMetaData("Int32Fast", UABuiltInType.Int32),
                New UAFieldMetaData("DateTime", UABuiltInType.DateTime)
            }

            ' Define the dataset subscription, with specific communication parameters.
            ' The dataset offset is needed with messages that do Not contain dataset writer Ids And use RawData field
            ' encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
            ' which is also the case here, but we are specifying the dataset offset anyway, for illustration.
            Dim dataSetSubscriptionDescriptor = New UADataSetSubscriptionDescriptor(
                pubSubConnectionDescriptor, filter, metaData)
            dataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset = 15

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_Metadata

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(dataSetSubscriptionDescriptor)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_Metadata(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 894 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
```

```
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 920 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1003 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1074 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 3074 {System.Int32}; Good]
    '[Int32Fast, 1140 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
// for their decoding directly in the code.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

type
  TSubscriberEventHandlers78 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Metadata;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Field1, Field2, Field3, Field4: _UAFieldMetaData;
  MetaData: _UADataSetMetaData;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers78;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
  // The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetUint16Identifier(30);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.WriterGroupId := 101;

  // Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset message.
  MetaData := CoUADataSetMetaData.Create;
  //
  Field1 := CoUAFieldMetaData.Create;
  Field1.BuiltInType := UABuiltInType_Boolean;
  Field1.Name := 'BoolToggle';
  MetaData.Add(Field1);
  //
  Field2 := CoUAFieldMetaData.Create;
  Field2.BuiltInType := UABuiltInType_Int32;
  Field2.Name := 'Int32';
  MetaData.Add(Field2);
  //
  Field3 := CoUAFieldMetaData.Create;
  Field3.BuiltInType := UABuiltInType_Int32;
  Field3.Name := 'Int32Fast';
  MetaData.Add(Field3);
  //
```

```
  Field4 := CoUAFieldMetaData.Create;
  Field4.BuiltInType := UABuiltInType_DateTime;
  Field4.Name := 'DateTime';
  MetaData.Add(Field4);
  //
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData := MetaData;

  // Define the specific communication parameters for the dataset subscription.
  // The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
  // encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
  // which is also the case here, but we are specifying the dataset offset anyway, for illustration.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset
:= 15;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers78.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers78.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 894 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
```

```
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 920 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1003 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1074 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 3074 {System.Int32}; Good]
//[Int32Fast, 1140 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
Rem for their decoding directly in the code.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

Option Explicit

Const UABuiltInType_Boolean = 1
Const UABuiltInType_Int32 = 6
Const UABuiltInType_DateTime = 13

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
' The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetUInt16Identifier 30
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.WriterGroupId = 101

' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset message.
Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
'
Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field1.BuiltInType = UABuiltInType_Boolean
Field1.Name = "BoolToggle"
MetaData.Add(Field1)
'
Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field2.BuiltInType = UABuiltInType_Int32
Field2.Name = "Int32"
MetaData.Add(Field2)
'
Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field3.BuiltInType = UABuiltInType_Int32
Field3.Name = "Int32Fast"
MetaData.Add(Field3)
'
Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field4.BuiltInType = UABuiltInType_DateTime
Field4.Name = "DateTime"
MetaData.Add(Field4)
'
Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

' Define the specific communication parameters for the dataset subscription.
' The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
' encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
```

```
' which is also the case here, but we are specifying the dataset offset anyway, for illustration.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset
= 15

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 894 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 920 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1003 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1074 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 3074 {System.Int32}; Good]
'[Int32Fast, 1140 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
Rem for their decoding directly in the code.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

' The subscriber object, with events.
'Public WithEvents Subscriber7 As EasyUASubscriber

Private Sub SubscribeDataSet_Metadata_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
    ' The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetUInt16Identifier 30
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.WriterGroupId = 101

    ' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset
message.
    Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
    '
    Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field1.BuiltInType = UABuiltInType_Boolean
    Field1.Name = "BoolToggle"
    MetaData.Add (Field1)
    '
    Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field2.BuiltInType = UABuiltInType_Int32
    Field2.Name = "Int32"
    MetaData.Add (Field2)
    '
    Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field3.BuiltInType = UABuiltInType_Int32
    Field3.Name = "Int32Fast"
    MetaData.Add (Field3)
    '
    Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field4.BuiltInType = UABuiltInType_DateTime
    Field4.Name = "DateTime"
    MetaData.Add (Field4)
    '
    Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

    ' Define the specific communication parameters for the dataset subscription.
    ' The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
    ' encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
    ' which is also the case here, but we are specifying the dataset offset anyway, for illustration.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset
= 15

    ' Instantiate the subscriber object and hook events.
    Set Subscriber7 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber7.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber7.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber7 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
```

```
End Sub

Private Sub Subscriber7_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 894 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 920 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1003 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1074 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 3074 {System.Int32}; Good]
'[Int32Fast, 1140 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
'
'...
'
```

Partial metadata

In some cases, partial metadata is sufficient to achieve the purpose. For example, your intent might be to identify dataset fields by their names, but the communication uses UADP mapping, which does not transfer the field names in the messages, and therefore some metadata is required for that. The full metadata, which includes types information for individual fields, is however only needed if RawData encoding is used. If the encoding is Variant or DataValue, type information is not strictly necessary, because dataset data can be decoded based simply on the contents of the dataset message.

You can simplify your code by only specifying field names in the metadata in such cases. This is illustrated in the example below.

C#

```
// This example shows how to subscribe to dataset messages and specify field names, without having the full metadata.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
```

```
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void FieldNames()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31), UAWriterGroupDescriptor.Null, 1);

            // Define the metadata, with the use of collection initializer for its fields. For UADP, the order of field
            // metadata must correspond to the order of fields in the dataset message.
            // Since the encoding is not RawData, we do not have to specify the type information for the fields.
            var metaData = new UADataSetMetaData
            {
                new UAFieldMetaData("BoolToggle"),
                new UAFieldMetaData("Int32"),
                new UAFieldMetaData("Int32Fast"),
                new UAFieldMetaData("DateTime")
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_FieldNames;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_FieldNames(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 25 {System.Int32}; Good]
        //[Int32Fast, 928 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[Int32, 26 {System.Int32}; Good]
```

```
        //[Int32Fast, 1007 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
        //[BoolToggle, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[Int32Fast, 1113 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 26 {System.Int32}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 27 {System.Int32}; Good]
        //[Int32Fast, 1201 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[Int32Fast, 1260 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 27 {System.Int32}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and specify field names, without having the full metadata.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub FieldNames()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) Is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31), UAWriterGroupDescriptor.Null, 1)

            ' Define the metadata, with the use of collection initializer for its fields. For UADP, the order of field
            ' metadata must correspond to the order of fields in the dataset message.
            ' Since the encoding Is Not RawData, we do Not have to specify the type information for the fields.
            Dim metaData = New UADataSetMetaData From {
                New UAFieldMetaData("BoolToggle"),
                New UAFieldMetaData("Int32"),
                New UAFieldMetaData("Int32Fast"),
                New UAFieldMetaData("DateTime")
            }

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_FieldNames

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_FieldNames(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
```

```
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 25 {System.Int32}; Good]
    '[Int32Fast, 928 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[Int32, 26 {System.Int32}; Good]
    '[Int32Fast, 1007 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
    '[BoolToggle, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[Int32Fast, 1113 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 26 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 27 {System.Int32}; Good]
    '[Int32Fast, 1201 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[Int32Fast, 1260 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 27 {System.Int32}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and specify field names, without having the full metadata.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

type
  TSubscriberEventHandlers75 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.FieldNames;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Field1, Field2, Field3, Field4: _UAFieldMetaData;
  MetaData: _UADataSetMetaData;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers75;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
```

```
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64, 31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Define the metadata, with the use of collection initializer for its fields. For UADP, the order of field
  // metadata must correspond to the order of fields in the dataset message.
  // Since the encoding is not RawData, we do not have to specify the type information for the fields.
  MetaData := CoUADataSetMetaData.Create;
  //
  Field1 := CoUAFieldMetaData.Create;
  Field1.Name := 'BoolToggle';
  MetaData.Add(Field1);
  //
  Field2 := CoUAFieldMetaData.Create;
  Field2.Name := 'Int32';
  MetaData.Add(Field2);
  //
  Field3 := CoUAFieldMetaData.Create;
  Field3.Name := 'Int32Fast';
  MetaData.Add(Field3);
  //
  Field4 := CoUAFieldMetaData.Create;
  Field4.Name := 'DateTime';
  MetaData.Add(Field4);
  //
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData := MetaData;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers75.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers75.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
```

```
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 25 {System.Int32}; Good]
//[Int32Fast, 928 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32, 26 {System.Int32}; Good]
//[Int32Fast, 1007 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1113 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 26 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//[Int32Fast, 1201 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1260 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and specify field names, without having the full metadata.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset message.
' Since the encoding is not RawData, we do not have to specify the type information for the fields.
Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
'
Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field1.Name = "BoolToggle"
MetaData.Add(Field1)
'
Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field2.Name = "Int32"
MetaData.Add(Field2)
'
Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field3.Name = "Int32Fast"
```

```
MetaData.Add(Field3)
'
Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field4.Name = "DateTime"
MetaData.Add(Field4)
'
Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub

WScript.Echo "Finished."



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 25 {System.Int32}; Good]
'[Int32Fast, 928 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32, 26 {System.Int32}; Good]
'[Int32Fast, 1007 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1113 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 26 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'[Int32Fast, 1201 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1260 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
```

```
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages and specify field names, without having the full metadata.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

' The subscriber object, with events.
'Public WithEvents Subscriber5 As EasyUASubscriber

Private Sub SubscribeDataSet_FieldNames_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset
message.
    ' Since the encoding is not RawData, we do not have to specify the type information for the fields.
    Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
    '
    Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field1.Name = "BoolToggle"
    MetaData.Add (Field1)
    '
    Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field2.Name = "Int32"
    MetaData.Add (Field2)
    '
    Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field3.Name = "Int32Fast"
    MetaData.Add (Field3)
    '
    Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field4.Name = "DateTime"
    MetaData.Add (Field4)
    '
    Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

    ' Instantiate the subscriber object and hook events.
    Set Subscriber5 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber5.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber5.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber5 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber5_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
```

```
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 25 {System.Int32}; Good]
'[Int32Fast, 928 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32, 26 {System.Int32}; Good]
'[Int32Fast, 1007 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1113 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 26 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'[Int32Fast, 1201 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1260 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'
'...
'
```

5.1.5.1.4 OPC UA PubSub logical resolution

The OPC UA PubSub logical resolution is a mechanism that allows you to write code that specifies the information you work in logical terms (symbolic names of PubSub objects), and not in physical terms (such as the various identifier numbers that appear "on the wire").

The logical resolution is automatically used by OPC Data Client if, in the various descriptors you pass into the UnsubscribeDataSet Method, you use a logical identifier (PubSub object name, usually), and do not specify a physical identifier at the same time.

> For quick tests and experiments with OPC UA PubSub, and especially if you do not want to lose time coding when you need to figure out the proper settings, you can use the **OpcCmd Utility (Section 11.4)** - see Using OpcCmd Utility as OPC UA PubSub Subscriber. The utility allows to specify all parameters of the dataset subscriptions that are accessible from code, but simply by entering them on the command line. It also has a built-in support for displaying the incoming datasets and their fields.

In This Topic

Resolver kinds

There are kinds of resolvers, depending on where the data needed for the logical resolution come from.

When the ResolverKind Property in the UAPubSubResolverDescriptor is set to UAPubSubResolverKind.None, the logical resolution cannot be

performed.

Publisher Endpoint Resolver

This resolver is used when the ResolverKind Property in the UAPubSubResolverDescriptor is set to UAPubSubResolverKind.PublisherEndpoint. The resolver connects to an OPC UA server which has the PubSub configuration model, and contains a configuration of the publisher whose messages you want to subscribe to. Based on the publisher configuration and the parameters and logical identifiers and you have specified, the resolver derives the parameters and physical identifiers needed for making the required subscription to the publisher's data.

The OPC UA server that provides the publisher configuration may be integrated with the publisher in the same hardware or software component, but it can also be separate.

When selecting this resolver kind, you need to specify the URL (and possibly other characteristics) of the OPC UA server with the publisher configuration using the PublisherEndpointDescriptor Property in the UAPubSubResolverDescriptor. Instead of creating the PubSub resolver descriptor yourself by setting its kind and the URL, you can call the static Publisher Method and create it in one step.

Publisher File Resolver

This resolver is used when the ResolverKind Property in the UAPubSubResolverDescriptor is set to UAPubSubResolverKind.PublisherFile. The resolver opens a file with PubSub configuration model that contains configuration of the publisher whose messages you want to subscribe to. Based on the publisher configuration and the parameters logical identifiers and you have specified, the resolver derives the parameters and physical identifiers needed for making the required subscription to the publisher's data.

When selecting this resolver kind, you need to specify the PubSub configuration file path and name using the PublisherFileResourceDescriptor Property in the UAPubSubResolverDescriptor. Instead of creating the PubSub resolver descriptor yourself by setting its kind and file name, you can call the static File Method and create it one step.

The configuration file must be in the UABinary format as described in the OPC UA specification (UABinaryFileDataType in Annex A "Common Types" of Part 14).

The use of the publisher file resolver is illustrated in the example below.

C#

```
// This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
// from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode
// fixed layout messages with RawData field encoding.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void ResolveFromFile()
        {
            // Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
            // we have. The file itself is at the root of the project, and we have specified that it has
to be copied to the
            // project's output directory.
            var pubSubResolverDescriptor = UAPubSubResolverDescriptor.File("UADemoPublisher-
Default.uabinary");

            // Define the PubSub connection we will work with, using its logical name in the PubSub
configuration.
```

```
            var pubSubConnectionDescriptor = new UAPubSubConnectionDescriptor { Name =
"FixedLayoutConnection" };
            // In some cases you may have to set the interface (network adapter) name that needs to be
used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. The writer group and the dataset writer are specified using their
logical names in the
            // PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub
connection.
            var filter = new UASubscribeDataSetFilter("FixedLayoutGroup", "SimpleWriter");

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_ResolveFromFile;
            subscriber.ResolverAccess += subscriber_ResolverAccess_ResolveFromFile;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, pubSubConnectionDescriptor, filter);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_ResolveFromFile(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        private static void subscriber_ResolverAccess_ResolveFromFile(object sender,
EasyUAResolverAccessEventArgs e)
        {
            // Display resolution information.
            Console.WriteLine(e);
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
```

```
        //[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3072 {System.Int32}; Good]
        //[Int32Fast, 894 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3072 {System.Int32}; Good]
        //[Int32Fast, 920 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3073 {System.Int32}; Good]
        //[Int32Fast, 1003 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3073 {System.Int32}; Good]
        //[Int32Fast, 1074 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 3074 {System.Int32}; Good]
        //[Int32Fast, 1140 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
' from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the resolution
is used to decode
' fixed layout messages with RawData field encoding.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation,
see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub ResolveFromFile()

            ' Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
            ' we have. The file itself is at the root of the project, and we have specified that it has
to be copied to the
            ' project's output directory.
            Dim pubSubResolverDescriptor = UAPubSubResolverDescriptor.File("UADemoPublisher-
Default.uabinary")

            ' Define the PubSub connection we will work with, using its logical name in the PubSub
configuration.
            Dim pubSubConnectionDescriptor = New UAPubSubConnectionDescriptor
            pubSubConnectionDescriptor.Name = "FixedLayoutConnection"
```

```
            ' In some cases you may have to set the interface (network adapter) name that needs to be
used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. The writer group And the dataset writer are specified using their
logical names in the
            ' PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub
connection.
            Dim filter = New UASubscribeDataSetFilter("FixedLayoutGroup", "SimpleWriter")

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_ResolveFromFile
            AddHandler subscriber.ResolverAccess, AddressOf subscriber_ResolverAccess_ResolveFromFile

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, pubSubConnectionDescriptor, filter)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_ResolveFromFile(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub

        Private Shared Sub subscriber_ResolverAccess_ResolveFromFile(ByVal sender As Object, ByVal e As
EasyUAResolverAccessEventArgs)
            ' Display resolution information.
            Console.WriteLine(e)
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    '[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
```

```
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 894 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 920 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1003 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1074 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 3074 {System.Int32}; Good]
    '[Int32Fast, 1140 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
// from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode
// fixed layout messages with RawData field encoding.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

type
  TSubscriberEventHandlers81 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
    procedure OnResolverAccess(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAResolverAccessEventArgs);
  end;

class procedure SubscribeDataSet.ResolveFromFile;
var
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers81;
begin
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;

  // Define the PubSub resolver. We want the information be resolved from a PubSub binary configuration
file that
  // we have. The file itself is included alongside the script.
```

```
  SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString :=
'UADemoPublisher-Default.uabinary';
  SubscribeDataSetArguments.ResolverDescriptor.ResolverKind := UAPubSubResolverKind_PublisherFile;

  // Define the PubSub connection we will work with, using its logical name in the PubSub configuration.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.Name :=
'FixedLayoutConnection';
  // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
  // the statement below. Your actual interface name may differ, of course.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.ResourceAddress.InterfaceName
:= 'Ethernet';

  // Define the filter. The writer group and the dataset writer are specified using their logical names
in the
  // PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub
connection.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.Name :=
'FixedLayoutGroup';
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.Name :=
'SimpleWriter';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers81.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;
  Subscriber.OnResolverAccess := SubscriberEventHandlers.OnResolverAccess;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers81.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
```

```
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

procedure TSubscriberEventHandlers81.OnResolverAccess(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAResolverAccessEventArgs);
begin
  // Display resolution information.
  WriteLn(eventArgs.ToString);
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 894 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 920 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1003 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1074 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 3074 {System.Int32}; Good]
//[Int32Fast, 1140 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
Rem from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode
Rem fixed layout messages with RawData field encoding.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
```

```
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

Option Explicit

Const UAPubSubResolverKind_PublisherFile = 3

Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")

' Define the PubSub resolver. We want the information be resolved from a PubSub binary configuration file
that
' we have. The file itself is included alongside the script.
SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString =
"UADemoPublisher-Default.uabinary"
SubscribeDataSetArguments.ResolverDescriptor.ResolverKind = UAPubSubResolverKind_PublisherFile

' Define the PubSub connection we will work with, using its logical name in the PubSub configuration.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.Name =
"FixedLayoutConnection"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly
to
' the statement below. Your actual interface name may differ, of course.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.ResourceAddress.InterfaceName
= "Ethernet"

' Define the filter. The writer group and the dataset writer are specified using their logical names in
the
' PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub connection.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.Name =
"FixedLayoutGroup"
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.Name =
"SimpleWriter"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
```

```
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub

Sub Subscriber_ResolverAccess(Sender, e)
    ' Display resolution information.
    WScript.Echo e
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 894 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 920 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1003 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1074 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 3074 {System.Int32}; Good]
'[Int32Fast, 1140 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
'
'...
```

Resolution rules

When specifying the identifiers for logical resolution, the conditions below must be fulfilled.

- The PubSub objects that you specify by name must exist in the configuration.
- When you specify logical information (such as writer group name or dataset writer name) in the subscribe dataset filter, the PubSub connection must also be described with its logical name.
- When you specify dataset writer name, you must also specify writer group name.
- If you specify both the published dataset name and the dataset writer, the dataset writer must be configured to publish the same dataset.
- If you specify published dataset name, there must be a dataset writer in the configuration that can be used to provide data for the subscription.

In general, there are two approaches that you can take when using the logical resolution (although you can also combine them, and also use

physical identifiers in some places):

- Specify the names of the PubSub connection, the writer group, and the dataset writer.
- Specify the name of the published dataset.

In the first case, the resolution provider finds the specified objects in the configuration, and uses their parameters to determine the proper subscription settings (such the PubSub connection descriptor, communication parameters, and subscribe dataset filter).

In the second case, the resolution provider finds the published dataset in the structure of the dataset folders in the PubSub configuration. It then attempts to find a dataset writer that is configured with this published dataset. Since the same dataset may be published by multiple dataset writers, the algorithm only considers as eligible the dataset writers that reside on PubSub connections with transport profile URI that the component supports.

The example below shows the approach with specifying published dataset name.

C#

```
// This example shows how to subscribe to dataset messages, specifying just the published dataset name,
and resolving all
// the dataset subscription arguments from an OPC-UA PubSub configuration file.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void PublishedDataSet()
        {
            // Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
            // we have. The file itself is at the root of the project, and we have specified that it has
to be copied to the
            // project's output directory.
            var pubSubResolverDescriptor = UAPubSubResolverDescriptor.File("UADemoPublisher-
Default.uabinary");

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            Console.WriteLine("Subscribing...");
            // Specify the published dataset name, and let all other subscription arguments be resolved
automatically.
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, "AllTypes-Dynamic", (sender, args) =>
            {
                // Display the dataset.
                if (args.Succeeded)
                {
                    // An event with null DataSetData just indicates a successful connection.
                    if (!(args.DataSetData is null))
                    {
                        Console.WriteLine();
                        Console.WriteLine($"Dataset data: {args.DataSetData}");
                        foreach (KeyValuePair<string, UADataSetFieldData> pair in
args.DataSetData.FieldDataDictionary)
                            Console.WriteLine(pair);
                    }
                }
                else
                {
```

```
            Console.WriteLine();
            Console.WriteLine($"*** Failure: {args.ErrorMessageBrief}");
        }
    });

    Console.WriteLine("Processing dataset message events for 20 seconds...");
    Thread.Sleep(20 * 1000);

    Console.WriteLine("Unsubscribing...");
    subscriber.UnsubscribeAllDataSets();

    Console.WriteLine("Waiting for 1 second...");
    // Unsubscribe operation is asynchronous, messages may still come for a short while.
    Thread.Sleep(1 * 1000);

    Console.WriteLine("Finished.");
}

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[BoolToggle, False {System.Boolean}; Good]
//[Byte, 137 {System.Byte}; Good]
//[Int16, 10377 {System.Int16}; Good]
//[Int32, 43145 {System.Int32}; Good]
//[Int64, 43145 {System.Int64}; Good]
//[SByte, 9 {System.Int16}; Good]
//[UInt16, 43145 {System.Int32}; Good]
//[UInt32, 43145 {System.Int64}; Good]
//[UInt64, 43145 {System.Decimal}; Good]
//[Float, 43145 {System.Single}; Good]
//[Double, 43145 {System.Double}; Good]
//[String, Lima {System.String}; Good]
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
//[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
//[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[BoolToggle, True {System.Boolean}; Good]
//[Byte, 138 {System.Byte}; Good]
//[Int16, 10378 {System.Int16}; Good]
//[Int32, 43146 {System.Int32}; Good]
//[Int64, 43146 {System.Int64}; Good]
//[SByte, 10 {System.Int16}; Good]
//[UInt16, 43146 {System.Int32}; Good]
//[UInt32, 43146 {System.Int64}; Good]
//[UInt64, 43146 {System.Decimal}; Good]
//[Float, 43146 {System.Single}; Good]
//[Double, 43146 {System.Double}; Good]
//[String, Mike {System.String}; Good]
//[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
//[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
//[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Byte, 138 {System.Byte}; Good]
//[Int16, 10378 {System.Int16}; Good]
//[Int32, 43146 {System.Int32}; Good]
//[Int64, 43146 {System.Int64}; Good]
//[SByte, 10 {System.Int16}; Good]
//[UInt16, 43146 {System.Int32}; Good]
```

```
        //[UInt32, 43146 {System.Int64}; Good]
        //[UInt64, 43146 {System.Decimal}; Good]
        //[Float, 43146 {System.Single}; Good]
        //[Double, 43146 {System.Double}; Good]
        //[String, Mike {System.String}; Good]
        //[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
        //[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
        //[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages, specifying just the published dataset name,
and resolving all
' the dataset subscription arguments from an OPC-UA PubSub configuration file.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation,
see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub PublishedDataSet()

            ' Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
            ' we have. The file itself is at the root of the project, and we have specified that it has
to be copied to the
            ' project's output directory.
            Dim pubSubResolverDescriptor = UAPubSubResolverDescriptor.File("UADemoPublisher-
Default.uabinary")

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            Console.WriteLine("Subscribing...")
            ' Specify the published dataset name, And let all other subscription arguments be resolved
automatically.
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, "AllTypes-Dynamic",
                Sub(sender, eventArgs)
                    '  Display the dataset.
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}", eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief)
                    End If
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()
```

```
        Console.WriteLine("Waiting for 1 second...")
        ' Unsubscribe operation is asynchronous, messages may still come for a short while.
        Threading.Thread.Sleep(1 * 1000)

        Console.WriteLine("Finished...")
    End Sub
End Class

' Example output
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[BoolToggle, False {System.Boolean}; Good]
'[Byte, 137 {System.Byte}; Good]
'[Int16, 10377 {System.Int16}; Good]
'[Int32, 43145 {System.Int32}; Good]
'[Int64, 43145 {System.Int64}; Good]
'[SByte, 9 {System.Int16}; Good]
'[UInt16, 43145 {System.Int32}; Good]
'[UInt32, 43145 {System.Int64}; Good]
'[UInt64, 43145 {System.Decimal}; Good]
'[Float, 43145 {System.Single}; Good]
'[Double, 43145 {System.Double}; Good]
'[String, Lima {System.String}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
'[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
'[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[BoolToggle, True {System.Boolean}; Good]
'[Byte, 138 {System.Byte}; Good]
'[Int16, 10378 {System.Int16}; Good]
'[Int32, 43146 {System.Int32}; Good]
'[Int64, 43146 {System.Int64}; Good]
'[SByte, 10 {System.Int16}; Good]
'[UInt16, 43146 {System.Int32}; Good]
'[UInt32, 43146 {System.Int64}; Good]
'[UInt64, 43146 {System.Decimal}; Good]
'[Float, 43146 {System.Single}; Good]
'[Double, 43146 {System.Double}; Good]
'[String, Mike {System.String}; Good]
'[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
'[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
'[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Byte, 138 {System.Byte}; Good]
'[Int16, 10378 {System.Int16}; Good]
'[Int32, 43146 {System.Int32}; Good]
'[Int64, 43146 {System.Int64}; Good]
'[SByte, 10 {System.Int16}; Good]
'[UInt16, 43146 {System.Int32}; Good]
'[UInt32, 43146 {System.Int64}; Good]
'[UInt64, 43146 {System.Decimal}; Good]
'[Float, 43146 {System.Single}; Good]
'[Double, 43146 {System.Double}; Good]
'[String, Mike {System.String}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
'[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
'...
```

```
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages, specifying just the published dataset name,
and resolving all
// the dataset subscription arguments from an OPC-UA PubSub configuration file.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

type
  TSubscriberEventHandlers79 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.PublishedDataSet;
var
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers79;
begin
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;

  // Define the PubSub resolver. We want the information be resolved from a PubSub binary configuration
file that
  // we have. The file itself is included alongside the script.

  SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString :=
'UADemoPublisher-Default.uabinary';
  SubscribeDataSetArguments.ResolverDescriptor.ResolverKind := UAPubSubResolverKind_PublisherFile;

  // Specify the published dataset name, and let all other subscription arguments be resolved
automatically.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.PublishedDataSetName := 'AllTypes-Dynamic';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers79.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers79.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
```

```
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[BoolToggle, False {System.Boolean}; Good]
//[Byte, 137 {System.Byte}; Good]
//[Int16, 10377 {System.Int16}; Good]
//[Int32, 43145 {System.Int32}; Good]
//[Int64, 43145 {System.Int64}; Good]
//[SByte, 9 {System.Int16}; Good]
//[UInt16, 43145 {System.Int32}; Good]
//[UInt32, 43145 {System.Int64}; Good]
//[UInt64, 43145 {System.Decimal}; Good]
//[Float, 43145 {System.Single}; Good]
//[Double, 43145 {System.Double}; Good]
//[String, Lima {System.String}; Good]
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
//[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
//[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[BoolToggle, True {System.Boolean}; Good]
//[Byte, 138 {System.Byte}; Good]
//[Int16, 10378 {System.Int16}; Good]
//[Int32, 43146 {System.Int32}; Good]
//[Int64, 43146 {System.Int64}; Good]
//[SByte, 10 {System.Int16}; Good]
//[UInt16, 43146 {System.Int32}; Good]
//[UInt32, 43146 {System.Int64}; Good]
//[UInt64, 43146 {System.Decimal}; Good]
//[Float, 43146 {System.Single}; Good]
//[Double, 43146 {System.Double}; Good]
//[String, Mike {System.String}; Good]
//[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
//[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
//[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
```

```
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Byte, 138 {System.Byte}; Good]
//[Int16, 10378 {System.Int16}; Good]
//[Int32, 43146 {System.Int32}; Good]
//[Int64, 43146 {System.Int64}; Good]
//[SByte, 10 {System.Int16}; Good]
//[UInt16, 43146 {System.Int32}; Good]
//[UInt32, 43146 {System.Int64}; Good]
//[UInt64, 43146 {System.Decimal}; Good]
//[Float, 43146 {System.Single}; Good]
//[Double, 43146 {System.Double}; Good]
//[String, Mike {System.String}; Good]
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
//[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages, specifying just the published dataset name,
and resolving all
Rem the dataset subscription arguments from an OPC-UA PubSub configuration file.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

Option Explicit

Const UAPubSubResolverKind_PublisherFile = 3

Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")

' Define the PubSub resolver. We want the information be resolved from a PubSub binary configuration file
that
' we have. The file itself is included alongside the script.
SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString =
"UADemoPublisher-Default.uabinary"
SubscribeDataSetArguments.ResolverDescriptor.ResolverKind = UAPubSubResolverKind_PublisherFile

' Specify the published dataset name, and let all other subscription arguments be resolved automatically.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.PublishedDataSetName = "AllTypes-Dynamic"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."
```

```
Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[BoolToggle, False {System.Boolean}; Good]
'[Byte, 137 {System.Byte}; Good]
'[Int16, 10377 {System.Int16}; Good]
'[Int32, 43145 {System.Int32}; Good]
'[Int64, 43145 {System.Int64}; Good]
'[SByte, 9 {System.Int16}; Good]
'[UInt16, 43145 {System.Int32}; Good]
'[UInt32, 43145 {System.Int64}; Good]
'[UInt64, 43145 {System.Decimal}; Good]
'[Float, 43145 {System.Single}; Good]
'[Double, 43145 {System.Double}; Good]
'[String, Lima {System.String}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
'[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
'[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[BoolToggle, True {System.Boolean}; Good]
'[Byte, 138 {System.Byte}; Good]
'[Int16, 10378 {System.Int16}; Good]
'[Int32, 43146 {System.Int32}; Good]
'[Int64, 43146 {System.Int64}; Good]
'[SByte, 10 {System.Int16}; Good]
'[UInt16, 43146 {System.Int32}; Good]
'[UInt32, 43146 {System.Int64}; Good]
'[UInt64, 43146 {System.Decimal}; Good]
'[Float, 43146 {System.Single}; Good]
'[Double, 43146 {System.Double}; Good]
'[String, Mike {System.String}; Good]
'[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
'[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
'[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Byte, 138 {System.Byte}; Good]
'[Int16, 10378 {System.Int16}; Good]
```

```
'[Int32, 43146 {System.Int32}; Good]
'[Int64, 43146 {System.Int64}; Good]
'[SByte, 10 {System.Int16}; Good]
'[UInt16, 43146 {System.Int32}; Good]
'[UInt32, 43146 {System.Int64}; Good]
'[UInt64, 43146 {System.Decimal}; Good]
'[Float, 43146 {System.Single}; Good]
'[Double, 43146 {System.Double}; Good]
'[String, Mike {System.String}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
'[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
'
'...
```

Message Receive Timeout Computation

When resolving from the publisher configuration, the resolution process is sometimes required to provide information that is somewhat difficult to obtain. The most prominent example of this is the MessageReceiveTimeout Property in the UADataSetSubscriptionDescriptor.CommunicationParameters. If you do not specify a non-zero value yourself, what should the resolution process fill in? The only hint it has from the provider configuration is the KeepAliveTime Property in the writer group (UAWriterGroupElement Class). But the keep alive time cannot directly be used as the message receive timeout: There needs to be some tolerance allowed, for computing and network delays. How big this tolerance should be is a big question, and depends on multiple parameters of your system.

In order to provide working results in most cases, the resolution provider uses a linear function to compute the message receive timeout from the keep alive time. The value of the keep alive time is multiple by *message receive timeout factor* (MessageReceiveTimeoutFactor Property), and then a *message receive timeout increase* (MessageReceiveTimeoutIncrease Property) is added to it.

The default message receive timeout factor is 1.05 (i.e. 5% over the keep alive time), and the default message receive timeout increase is 250 milliseconds. Example: For a keep-alive time of 1000 milliseconds, with the default parameters, the resulting message timeout will be 1300 milliseconds. If the computation described here does not work well for you, you can either set the message receive timeout directly to a value you have determined as proper, or you can modify the computation parameters so that they accommodate the keep alive times better to your system.

Data not in the message

When the resolver constructs the subscribe dataset filter (see **Message Filtering (OPC UA PubSub) (Section 5.1.5.1.2)**), it has knowledge about which pieces of data are included in the PubSub messages, and which are excluded. As explained with the subscribe dataset filter, filtering must not be specified on data pieces that are not included in the PubSub messages. The resolver sets the filter properties in such a way that no filtering is specified for information that is excluded from the PubSub messages, and guarantees that the filter will actually work correctly.

Resiliency

The publisher endpoint resolver and the publisher file resolver are *periodic resolvers*. This means that the resolution with them is not normally a one-time occurrence. This allows for long-running programs to achieve resiliency without further programming needed on your part.

If a periodic resolver encounters a failure obtaining the data it needs to perform the resolution, it will retry after a period given by the FailureRetrialDelay Property (in milliseconds). An inaccessibility of the data does not therefore result in permanent loss of the subscription. Note that, however, if there are no problems accessing the data needed to perform the resolution, the resolution will proceed, and if the data is flawed (e.g. in case of an incorrect configuration of the publisher), it may yield subscription parameters that won't work at all, or won't work properly. This kind of error cannot be easily detected.

If a periodic resolver succeeds in obtaining the data it needs to perform the resolution, it will still retry the resolution - but this time, after a period given by the SuccessRetrialDelay Property (in milliseconds). This behavior allows for improper configurations be fixed "on the fly": If there was a configuration error and the resolution yielded dysfunctional result, and the configuration error was later fixed, eventually, the resolver will pick up the new configuration, and make a new resolution, this time with a result that will be functional.

OPC Data Client does not necessarily create separate resolver objects for each dataset subscription you make. Instead, resolvers that use the same data source (e.g. access the same OPC UA server) are shared. For optimization reasons in this scenario, the delays that control the repeated resolution attempts described above are actually further modified by an additional MaximumWaitDelay Property (in milliseconds) in the parameters. This is kind of maximum allowed age of the resolution result. If a retrial has been scheduled for an earlier time, its resolution result will be used (sparing an unnecessary resolution attempt).

The resolver parameters mentioned above are all part of the PeriodicResolverParameters Class instance, stored in static EasyUASubscriber.AdaptableParameters.ResolverParameters.PeriodicResolverParameters Property (if the Isolated Property is set to 'false', which is the default), or in EasyUASubscriber.InstanceParameters.ResolverParameters.PeriodicResolverParameters Property (if you have set the Isolated Property to 'true').

The ResolverAccess Event

For advanced scenarios, your code is informed about certain aspects of the logical resolution process. This is done using the ResolverAccess Event on the EasyUASubscriber Class. With this event, you will receive event arguments that are an instance of EasyUAResolverAccessEventArgs Class. The event arguments contain the ResolverDescriptor Property which tell you the kind of the resolver involved, and its parameters.

The event is raised every time after the resolver accesses its information source. There are two possible outcomes:

- The access has been successful. That it, the resolver could obtain the data it needs to perform the resolution. In this case, the Exception Property in the event arguments is null. Note that the success in obtaining the data needed for resolution does not yet indicate the success of the resolution itself - there can still be errors then, and they are reported through the normal DataSetMessage Event.
- The access has failed. The resolver could not obtain the data it needs to perform the resolution. In this case, the Exception Property in the event arguments is non-null, and it contains information about the failure.

The SubscriptionResolved Event

When the dataset subscription descriptor is successfully resolved, the component raises the SubscriptionResolved Event. This event contains the automatically resolved dataset subscription descriptor in its EasyUASubscriptionResolvedEventArgs.DataSetSubscriptionDescriptor Property. This allows your code to inspect it.

In addition, the event handlers for the SubscriptionResolved Event are allowed to modify the dataset subscription descriptor, and the modifications will be taken over by the component and use d for the actual subscription. This is useful in cases when the automated resolution is not complete, or fully correct.

For example, the PubSub configuration only contains endpoint URLs of Security Key Services (SKS), but no user authentication data with them. The automated resolution fill is in the URLs in the SecurityKeyServices Property of the UAPubSubCommunicationParameters Class, and you can write code inside the SubscriptionResolved Event handled to provide the use authentication information to authenticate the user to the SKS.

The resolved dataset subscription descriptor

In the DataSetMessage Event, after a successful logical resolution, the event arguments contain the final resolved dataset subscription descriptor in the ResolvedDataSetSubscriptionDescriptor Property. The resolved descriptor is derived from the one you passed to the SubscribeDataSet Method, with all the logical identifiers resolved to physical ones, and possible modifications made in the handlers for the SubscriptionResolved Event.

5.1.5.1.5 External OPC UA PubSub packages

In many cases, all code needed to perform OPC UA PubSub functionality is contained directly in core OPC Data Client assemblies. There are some situations, however, when one or more additional pieces of software ("package") is needed.

For more information about the use of communication packages in general, see http://kb.opclabs.com/Using_communication_packages .

In This Topic

OpcLabs.Pcap communication package

OpcLabs.Pcap is an external assembly/package of OPC Data Client that is only needed for:

- OPC UA PubSub Ethernet transport (the "opc.eth" scheme).
- OPC UA PubSub reading or writing Wireshark capture files.

OPC Data Client switches to the OpcLabs.Pcap communication package whenever it detects that any of the above is required. Although OpcLabs.Pcap use does not require anything different from the coding point of view, it requires special handling in several respects - e.g. with regard to licensing, installation, and configuration.

For more information about OpcLabs.Pcap, see http://kb.opclabs.com/OpcLabs.Pcap_communication_package .

MQTT communication packages

MQTT communication is not built-into core OPC Data Client assemblies, but is rather always provided externally, by means of a package that uses some specific MQTT communication library. OPC Data Client automatically uses a default MQTT communication package unless you make a different choice. In case you are fine with the default MQTT communication package selection, you do not need to do anything special in your code, but you still need to make sure that the package is installed and referenced, and you may also want to configure it in some specific way.

For more information about the packages available, see http://kb.opclabs.com/MQTT_communication_packages .

OpcLabs.UAPubSubJson mapping package

This package provides JSON mapping for OPC UA PubSub. It is needed whenever JSON encoding is used in OPC UA PubSub, that is:

- With transport profiles that use JSON message mapping, such as when MQTT or AMQP is used with JSON.
- With transport profiles that use JSON in the message "envelope", such as MQTT version 5.0 (even when the message itself might not be encoded in JSON).

For more information about UAPubSubJson, see http://kb.opclabs.com/OPC_UA_PubSub_JSON_mapping_package .

5.1.5.1.6 Subscribing to a specific dataset field (OPC UA PubSub)

The "native" granular unit of OPC UA PubSub data delivery is a dataset message. The dataset message can contain values of multiple fields. All these field values belong together - they represent a snapshot of some state of the system at a common time point. Individual fields can then be extracted from the dataset message as needed.

There are, however, situations in which your code is only interested in a specific field (or specific fields) of the dataset. For such cases, OPC Data Client offers you a mechanism that allows you to subscribe to individual dataset fields, and the field extraction will be made for you by the component.

In This Topic

This functionality is not available under (or the text does not apply to) COM development platform.

For quick tests and experiments with OPC UA PubSub, and especially if you do not want to lose time coding when you need to figure out the proper settings, you can use the **OpcCmd Utility (Section 11.4)** - see Using OpcCmd Utility as OPC UA PubSub Subscriber. The utility allows to specify all parameters of the dataset subscriptions that are accessible from code, but simply by entering them on the command line. It also has a built-in support for displaying the incoming datasets and their fields.

How-To

Subscribing to specific dataset fields is very similar to subscribing to the whole datasets. A subscription to a single dataset field is initiated by a call to SubscribeDataSetField Method. This is an extension method, with multiple overloads. The primary form takes an argument that is an instance of EasyUASubscribeDataSetFieldArguments Class. This class is very similar to the EasyUASubscribeDataSetArguments Class, but it has an additional FieldName Property, which specifies the name of the dataset field to be subscribed to.

The are various overloads of the SubscribeDataSetField Method, and also various overloads of the EasyUASubscribeDataSetFieldArguments Constructor that allow you to make your subscription code shorter, depending on which pieces of information needed for the subscription you have at hand.

The SubscribeDataSetField Method(s) return an integer (dataset) field subscription handle. Call the UnsubscribeDataSetField Method, passing it this field subscription handle, in order to unsubscribe from the dataset field.

Note that as usual, both the SubscribeDataSetField Method a UnsubscribeDataSetField Method operate asynchronously. That is, you may start receiving callbacks before the SubscribeDataSetField Method returns, and there may be some final callbacks made even after the UnsubscribeDataSetField Method returns.

The callback that you pass in is of type EasyUADataSetFieldMessageEventHandler Delegate, and carries with it event arguments of type EasyUADataSetFieldMessageEventArgs Class. This class is very similar to the EasyUADataSetMessageEventArgs Class, except that instead of the data for the whole dataset message, it contains only data for the particular dataset field that you have subscribed to, in its FieldData Property. Successful establishment of a PubSub connection is indicated by a callback event arguments in which both the Exception Property and FieldData Property contain a null reference.

Pros and Cons

If you just need a specific dataset field, the code becomes simpler with the use of single-field subscriptions. Also, there is no big performance penalty: Even if you subscribe to multiple dataset fields using this mechanism, internally, OPC Data Client makes just one PubSub connection to the message oriented middleware, and also all the message parsing, filtering and most of the processing is made just once. It is only the final message delivery steps that are split, and different.

The disadvantages related to the use of subscriptions to specific dataset fields are:

- You lose the association between the dataset field changes - it is no longer apparent which field values come from the same dataset message.
- You have to use callbacks (not events).
- Does not work in COM.
- They are slightly less efficient with regard to processing speed and memory consumption.

Examples

C#

```
// This example shows how to subscribe to a single dataset field, resolving logical
parameters to physical from an OPC-UA
// PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode fixed layout
// messages with RawData field encoding.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

using System;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    class SubscribeDataSetField
    {
        public static void Main1()
        {
            // Define the PubSub resolver. We want the information be resolved from a
PubSub binary configuration file that
            // we have. The file itself at the root of the project, and we have
specified that it has to be copied to the
            // project's output directory.
            var pubSubResolverDescriptor =
UAPubSubResolverDescriptor.File("UADemoPublisher-Default.uabinary");

            // Define the PubSub connection we will work with, using its logical name
in the PubSub configuration.
            var pubSubConnectionDescriptor = new UAPubSubConnectionDescriptor { Name =
"FixedLayoutConnection" };
            // In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. The writer group and the dataset writer are specified
using their logical names in the
            // PubSub configuration. The publisher Id in the filter will be taken from
the logical PubSub connection.
            var filter = new UASubscribeDataSetFilter("FixedLayoutGroup",
"SimpleWriter");

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            Console.WriteLine("Subscribing...");
            int fieldHandle = subscriber.SubscribeDataSetField(
                pubSubResolverDescriptor,
                pubSubConnectionDescriptor,
```

```
                filter,
                "Int32Fast",
                (sender, args) => { Console.WriteLine(args); });

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeDataSetField(fieldHandle);

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //Success
        //Success; 1626 {System.Int32}; Good
        //Success; 1711 {System.Int32}; Good
        //Success; 1741 {System.Int32}; Good
        //Success; 1837 {System.Int32}; Good
        //Success; 1897 {System.Int32}; Good
        //Success; 1993 {System.Int32}; Good
        //Success; 2082 {System.Int32}; Good
        //Success; 2135 {System.Int32}; Good
        //Success; 2185 {System.Int32}; Good
        //Success; 2241 {System.Int32}; Good
        //Success; 2324 {System.Int32}; Good
        //Success; 2368 {System.Int32}; Good
        //Success; 2423 {System.Int32}; Good
        //Success; 2445 {System.Int32}; Good
        //Success; 2497 {System.Int32}; Good
        //Success; 2584 {System.Int32}; Good
        //Success; 2608 {System.Int32}; Good
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to a single dataset field, resolving logical
parameters to physical from an OPC-UA
' PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode fixed layout
' messages with RawData field encoding.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
the interface name to be used.
```

```
Imports OpcLabs.EasyOpc.UA.PubSub

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class SubscribeDataSetField
        Public Shared Sub Main1()

            ' Define the PubSub resolver. We want the information be resolved from a
PubSub binary configuration file that
            ' we have. The file itself is at the root of the project, and we have
specified that it has to be copied to the
            ' project's output directory.
            Dim pubSubResolverDescriptor =
UAPubSubResolverDescriptor.File("UADemoPublisher-Default.uabinary")

            ' Define the PubSub connection we will work with, using its logical name in
the PubSub configuration.
            Dim pubSubConnectionDescriptor = New UAPubSubConnectionDescriptor
            pubSubConnectionDescriptor.Name = "FixedLayoutConnection"
            ' In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. The writer group And the dataset writer are specified
using their logical names in the
            ' PubSub configuration. The publisher Id in the filter will be taken from
the logical PubSub connection.
            Dim filter = New UASubscribeDataSetFilter("FixedLayoutGroup",
"SimpleWriter")

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            Console.WriteLine("Subscribing...")
            Dim fieldHandle = subscriber.SubscribeDataSetField(
                pubSubResolverDescriptor,
                pubSubConnectionDescriptor,
                filter,
                "Int32Fast",
                Sub(sender, eventArgs)
                    Console.WriteLine(eventArgs)
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class
```

```
    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    'Success
    'Success; 1626 {System.Int32}; Good
    'Success; 1711 {System.Int32}; Good
    'Success; 1741 {System.Int32}; Good
    'Success; 1837 {System.Int32}; Good
    'Success; 1897 {System.Int32}; Good
    'Success; 1993 {System.Int32}; Good
    'Success; 2082 {System.Int32}; Good
    'Success; 2135 {System.Int32}; Good
    'Success; 2185 {System.Int32}; Good
    'Success; 2241 {System.Int32}; Good
    'Success; 2324 {System.Int32}; Good
    'Success; 2368 {System.Int32}; Good
    'Success; 2423 {System.Int32}; Good
    'Success; 2445 {System.Int32}; Good
    'Success; 2497 {System.Int32}; Good
    'Success; 2584 {System.Int32}; Good
    'Success; 2608 {System.Int32}; Good
    '...

End Namespace
```

5.1.5.2 Accessing OPC UA PubSub Configuration Model

Introduction

In This Topic

PubSub configuration describes the parameters of an OPC UA publisher and/or subscriber. It can be stored in a file, or it can be accessed through an information model in a "normal" OPC UA server.

Note that the use of PubSub configuration model is entirely optional. The model is standardized in OPC UA specifications, and when used, it must be used "as is". But, OPC UA publishers and subscribers may not use the configuration model at all, or they may choose to expose their configuration is some different way.

> If your intent is to use the PubSub configuration to find data that would allow you to set up a PubSub subscription, you can use the built-in **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** mechanism instead. This will save you lots of work.

> Currently, OPC Data Client only provides a read-only view of the PubSub configuration. That is, you cannot use OPC Data Client to change anything in the PubSub configuration.

You can obtain the IUAReadOnlyPubSubConfiguration Interface by calling appropriate methods on the **OPC UA Publish-Subscribe Client (Section 6.4.3)**.

Concepts

Most PubSub objects in the PubSub configuration are identified by their names. The name is usually a string, but some PubSub object are identified by an instance of UAQualifiedName Class.

Depending on the type of PubSub object and the structure of other arguments, sometimes a simple name of the PubSub object is enough to unambiguously locate it in the PubSub configuration. In other cases, the name is not sufficient, and a locator has to be used.

In this context, locator is a structure that consists of two or more names of PubSub objects. The names together unambiguously determine the location of the PubSub object in the PubSub configuration. For example, a dataset writer locator (represented by the UADataSetWriterLocator Class) contains

- a PubSub connection name,
- a writer group name, and
- a dataset writer name.

Published datasets are organized in a tree of dataset folders. Each published dataset has a name that must be unique not only in its dataset folder, but across the whole PubSub configuration. If a location of a published dataset in the PubSub configuration needs to be specified, a dataset folder path is used. The dataset folder path is a sequence of dataset folder names, starting from the root of the dataset folder tree.

Because published dataset names are unambiguous across the whole PubSub configuration, some methods allow you to omit the dataset folder path. In such case, you can pass a null reference in place of the dataset folder path, and the method will work on the whole tree. Note that a null reference in place of a dataset folder path is different from an empty path (a non-null, empty sequence): An empty dataset folder path denotes a "root" folder.

Some more information related to OPC UA PubSub configuration can be found in **OPC UA PubSub logical identifiers (Section 4.12.2)**.

Functionality

The methods available on the PubSub configuration allow you to look up PubSub objects using various criteria, and retrieve their data. They are grouped into several categories.

Looking up and searching for PubSub objects

PubSub objects can be looked up using their names. Methods that allow you to look up certain type of object in the PubSub configuration (sometimes given the location of their logical parent) are consistently called FindXXXX, where XXXX determines the type of PubSub object being looked up.

Attempt to look up an object that does not exist is not considered an error. When any of the FindXXXX does not find the PubSub object being looked up, it does not throw an exception. Instead, it simply returns a null reference. If the method call, however, contains argument that give location where the lookup is to be performed (e.g. the name of the PubSub connection when looking for a writer group), this location must be valid, otherwise an error occurs (UALogicalException Class).

List of the FindXXXX methods is below.

- FindConnectionElement Method
- FindDataSetWriterElement Method
- FindPublishedDataSetElement Method
- FindWriterGroupElement Method

If you need to look for a published dataset (by its name) in a given dataset folder and its subfolders, use the SearchPublishedDataSet Method.

Determining PubSub object presence

In most cases, you can use one of the FindXXXX methods to determine whether a certain PubSub object is present in the PubSub configuration. The PubSub object with the given name is present if the result of the FindXXXX method is not null. Alternatively, you can use one of the ListXXXX methods, and check whether the PubSub object name is among those returned.

You can use the above described approaches, except:

- when you need to determine whether a certain dataset folder exists, there is no FindDataSetFolder method (because there is no additional information for the dataset folder besides its name). In this case, use the HasDataSetFolder Method.
- when you need to determine whether a certain extension field exists in a specified published dataset. In this case, use the HasExtensionField Method.

The methods to determine the PubSub object presence are consistently named HasXXXX, where XXXX determines the type of PubSub object being tested for existence.

Getting PubSub object names and locators

If you do not know the names of certain PubSub objects, you can obtain the names from the PubSub configuration, usually given the location of some kind of logical parent. The methods that return the list of such names (or locators) are consistently called ListXXXX, where XXXX determines the type of PubSub object.

The methods below return names or locators of PubSub objects residing under a given parent.

- ListDataSetFolderNames Method
- ListDataSetWriterLocators Method
- ListDataSetWriterNames Method
- ListExtensionFieldQualifiedNames Method
- ListPublishedDataSetNames Method
- ListPubSubConnectionNames Method
- ListWriterGroupNames Method

There are also utility methods, built on top of others, that provide data about PubSub objects combined from multiple locations. They are briefly described below.

- ListAllConnectionDataSetWriterLocators Method. Gets locators of all dataset writers on a specified PubSub connection.
- ListAllDataSetWriterLocators Method. Gets locators of all dataset writers in the whole PubSub configuration.
- ListAllPublishedDataSetNames Method. Gets names of all published datasets in the PubSub configuration.

Retrieving PubSub objects

Methods that retrieve PubSub objects from the configuration are consistently named using the GetXXXX pattern. In most cases, a PubSub object can be retrieved by looking it up using one of the FindXXXX methods described above. For the remaining information in the PubSub configuration that cannot be accessed in this way, use one of the methods below.

- GetMultipleExtensionFieldValues Method (or, if you need just one extension field, GetExtensionFieldValue Method).
- GetPublishedDataSetSourceElement Method

- GetPubSubConfigurationElement Method

The published dataset can optionally be associated with a published dataset source. The information about the published dataset source can be retrieved using the GetPublishedDataSetSourceElement Method. Because the published dataset source is optional, this method (and the methods that are derived from it - see further) returns null when the published dataset source is not configured - this is a difference from all other GetXXXX methods which never return null. The result of the GetPublishedDataSetSourceElement Method is polymorphic - its type depends on the type of the published dataset source, and it can be an instance of either UAPublishedDataItemsElement Class or UAPublishedEventsElement Class. If you know upfront which type of published dataset source you expect, you can use one of the extension methods, the GetPublishedDataItemsElement Method or GetPublishedEventsElement Method, to directly retrieve the desired type; they give an error if the actual type is not what you have coded for.

The methods below provide basically a functionality of the corresponding FindXXXX methods, with the difference that the PubSub object being looked up must exist; if not, the method gives an error. Consequently, in case of successful completion of any of these methods, the return value is never null.

- GetConnectionElement Method
- GetDataSetWriterElement Method
- GetPublishedDataSetElement Method
- GetWriterGroupElement Method

Sometimes you need to retrieve information for all child objects under certain parent. This is, in fact, a combination of a ListXXXXNames method and one of the GetXXXXElement methods above. For convenience, OPC Data Client provides (extension) methods for this functionality already. The return type is always a keyed collection of elements, where the key is the name. The methods in this group are given below.

- GetConnectionElements Method
- GetDataSetWriterElements Method
- GetPublishedDataSetElements Method
- GetWriterGroupElements Method

Examples

The example below starts at the "root" of the PubSub configuration, and first obtains names of all PubSub connections available (they are at the 1st level). For each PubSub connection, given its name, it obtains names of writer groups configured on that PubSub connection (they are at the 2nd level). And, for each such writer group, given its name, it obtains names of all dataset writers configured on that writer group (they are at the 3rd level). Besides the PubSub object names, the commented parts also show how to obtain more detailed information about each PubSub object.

C#

```
// This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
// OPC UA PubSub configuration.

using System;
using OpcLabs.BaseLib.Collections.Specialized;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions;
```

```
namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
{
    partial class GetMethods
    {
        public static void PubSubComponents()
        {
            // Instantiate the publish-subscribe client object.
            var publishSubscribeClient = new EasyUAPublishSubscribeClient();

            try
            {
                Console.WriteLine("Loading the configuration...");
                // Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                // specified that it has to be copied to the project's output
directory.
                IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary");

                // Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA Server
                // with appropriate information model.
                //IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                //
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010");

                // Get the names of PubSub connections in the configuration, regardless
of the folder they reside in.
                StringCollection pubSubConnectionNames =
pubSubConfiguration.ListPubSubConnectionNames();
                foreach (string pubSubConnectionName in pubSubConnectionNames)
                {
                    Console.WriteLine($"PubSub connection: {pubSubConnectionName}");

                    // You can use the statement below to obtain parameters of the
PubSub connection.
                    //UAPubSubConnectionElement connectionElement =
                    //
pubSubConfiguration.GetConnectionElement(pubSubConnectionName);

                    // Get names of the writer groups on this PubSub connection.
                    StringCollection writerGroupNames =
pubSubConfiguration.ListWriterGroupNames(pubSubConnectionName);
                    foreach (string writerGroupName in writerGroupNames)
                    {
                        Console.WriteLine($"  Writer group: {writerGroupName}");

                        // You can use the statement below to obtain parameters of the
writer group.
                        //UAWriterGroupElement writerGroupElement =
                        //
pubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName);

                        // Get names of the dataset writers on this writer group.
                        StringCollection dataSetWriterNames =
```

```
pubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName);
                        foreach (string dataSetWriterName in dataSetWriterNames)
                        {
                            Console.WriteLine($"    Dataset writer:
{dataSetWriterName}");

                            // You can use the statement below to obtain parameters of
the dataset writer.
                            //UADataSetWriterElement dataSetWriterElement =
pubSubConfiguration.GetDataSetWriterElement(
                            //    pubSubConnectionName, writerGroupName,
dataSetWriterName);
                        }
                    }
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine($"*** Failure:
{uaException.InnerException.Message}");
            }

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Loading the configuration...
        //PubSub connection: FixedLayoutConnection
        //  Writer group: FixedLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: AllTypesWriter
        //    Dataset writer: MassTestWriter
        //PubSub connection: DynamicLayoutConnection
        //  Writer group: DynamicLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: MassTestWriter
        //    Dataset writer: AllTypes-DynamicWriter
        //    Dataset writer: EventSimpleWriter
        //PubSub connection: FlexibleLayoutConnection
        //  Writer group: FlexibleLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: MassTestWriter
        //    Dataset writer: AllTypes-DynamicWriter
        //Finished.
    }
}
```

VB.NET

```
' This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
' OPC UA PubSub configuration.

Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
'Imports OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions
```

```
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions

Namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
    Partial Friend Class GetMethods
        Public Shared Sub PubSubComponents()

            ' Instantiate the publish-subscribe client object.
            Dim publishSubscribeClient = New EasyUAPublishSubscribeClient()

            Try
                Console.WriteLine("Loading the configuration...")
                ' Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                ' specified that it has to be copied to the project's output directory.
                Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary")

                ' Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                ' Server with appropriate information model.
                'Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                '
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010")

                ' Get the names of PubSub connections in the configuration, regardless
of the folder they reside in.
                Dim pubSubConnectionNames =
pubSubConfiguration.ListPubSubConnectionNames()
                For Each pubSubConnectionName As String In pubSubConnectionNames
                    Console.WriteLine($"PubSub connection: {pubSubConnectionName}")

                    ' You can use the statement below to obtain parameters of the
PubSub connection.
                    'Dim connectionElement As UAPubSubConnectionElement =
                    '    pubSubConfiguration.GetConnectionElement(pubSubConnectionName)

                    ' Get names of the writer groups on this PubSub connection.
                    Dim writerGroupNames =
pubSubConfiguration.ListWriterGroupNames(pubSubConnectionName)
                    For Each writerGroupName As String In writerGroupNames
                        Console.WriteLine($"  Writer group: {writerGroupName}")

                        ' You can use the statement below to obtain parameters of the
writer group.
                        'Dim writerGroupElement As UAWriterGroupElement =
                        '
pubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName)

                        ' Get names of the dataset writers on this writer group.
                        Dim dataSetWriterNames =

pubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName)
                        For Each dataSetWriterName As String In dataSetWriterNames
                            Console.WriteLine($"    Dataset writer:
{dataSetWriterName}")
```

```
                            ' You can use the statement below to obtain parameters of
the dataset writer.
                            'Dim dataSetWriterElement As UADataSetWriterElement =
pubSubConfiguration.GetDataSetWriterElement(
                            '    pubSubConnectionName, writerGroupName,
dataSetWriterName)
                        Next dataSetWriterName
                    Next writerGroupName
                Next pubSubConnectionName
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Loading the configuration...
    'PubSub connection FixedLayoutConnection
    '  Writer group: FixedLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: AllTypesWriter
    '    Dataset writer: MassTestWriter
    'PubSub connection: DynamicLayoutConnection
    '  Writer group: DynamicLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: MassTestWriter
    '    Dataset writer: AllTypes-DynamicWriter
    '    Dataset writer: EventSimpleWriter
    'PubSub connection: FlexibleLayoutConnection
    '  Writer group: FlexibleLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: MassTestWriter
    '    Dataset writer: AllTypes-DynamicWriter
    'Finished.

End Namespace
```

Object Pascal

```
// This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
// OPC UA PubSub configuration.

class procedure GetMethods.PubSubComponents;
var
  //DataSetWriterElement: _UADataSetWriterElement;
  DataSetWriterName: string;
  DataSetWriterNames: _StringCollection;
  //EndpointDescriptor: _UAEndpointDescriptor;
  I, J, K: Integer;
  //PubSubConnectionElement: _UAPubSubConnectionElement;
  PubSubConnectionName: string;
```

```
  PubSubConnectionNames: _StringCollection;
  PublishSubscribeClient: _EasyUAPublishSubscribeClient;
  PubSubConfiguration: _UAReadOnlyPubSubConfiguration;
  //WriterGroupElement: _UAWriterGroupElement;
  WriterGroupName: string;
  WriterGroupNames: _StringCollection;
begin
  // Instantiate the publish-subscribe client object.
  PublishSubscribeClient := CoEasyUAPublishSubscribeClient.Create;

  try
    WriteLn('Loading the configuration...');
    // Load the PubSub configuration from a file. The file itself is included alongside
the script.
    PubSubConfiguration :=
PublishSubscribeClient.LoadReadOnlyConfiguration('UADemoPublisher-Default.uabinary');

    // Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    // with appropriate information model.
    //EndpointDescriptor := CoUAEndpointDescriptor.Create;
    //EndpointDescriptor.UrlString := 'opc.tcp://localhost:48010';
    //PubSubConfiguration :=
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor);

    // Get the names of PubSub connections in the configuration.
    PubSubConnectionNames := PubSubConfiguration.ListPubSubConnectionNames;

    for I := 0 to PubSubConnectionNames.Count-1 do
    begin
      PubSubConnectionName := PubSubConnectionNames[I];
      WriteLn('PubSub connection: ', PubSubConnectionName);

      // You can use the statement below to obtain parameters of the PubSub connection.
      //PubSubConnectionElement :=
PubSubConfiguration.GetConnectionElement(PubSubConnectionName);

      // Get names of the writer groups on this PubSub connection.
      WriterGroupNames :=
PubSubConfiguration.ListWriterGroupNames(PubSubConnectionName);
      for J := 0 to WriterGroupNames.Count-1 do
      begin
        WriterGroupName := WriterGroupNames[J];
        WriteLn('  Writer group: ', WriterGroupName);

        // You can use the statement below to obtain parameters of the writer group.
        //WriterGroupElement :=
PubSubConfiguration.GetWriterGroupElement(PubSubConnectionName, WriterGroupName);

        // Get names of the dataset writers on this writer group.
        DataSetWriterNames :=
PubSubConfiguration.ListDataSetWriterNames(PubSubConnectionName, WriterGroupName);
        for K := 0 to DataSetWriterNames.Count-1 do
        begin
          DataSetWriterName := DataSetWriterNames[K];
          WriteLn('    Dataset writer: ', DataSetWriterName);
```

```
          // You can use the statement below to obtain parameters of the dataset
writer.
          //DataSetWriterElement :=
PubSubConfiguration.GetDataSetWriterElement(PubSubConnectionName, WriterGroupName,
DataSetWriterName);
        end;
      end;
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Finished.');
end;

// Example output:
//
//Loading the configuration...
//PubSub connection: FixedLayoutConnection
//  Writer group: FixedLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: AllTypesWriter
//    Dataset writer: MassTestWriter
//PubSub connection: DynamicLayoutConnection
//  Writer group: DynamicLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: MassTestWriter
//    Dataset writer: AllTypes-DynamicWriter
//    Dataset writer: EventSimpleWriter
//PubSub connection: FlexibleLayoutConnection
//  Writer group: FlexibleLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: MassTestWriter
//    Dataset writer: AllTypes-DynamicWriter
//Finished.
```

VBScript

```
Rem This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
Rem OPC UA PubSub configuration.

Option Explicit

' Instantiate the publish-subscribe client object.
Dim PublishSubscribeClient: Set PublishSubscribeClient =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.InformationModel.EasyUAPublishSubscribeClient")

On Error Resume Next
DumpPubSubComponents
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
```

```
On Error Goto 0

WScript.Echo "Finished."



Sub DumpPubSubComponents()
    WScript.Echo "Loading the configuration..."
    ' Load the PubSub configuration from a file. The file itself is included alongside
the script.
    Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-Default.uabinary")

    ' Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    ' with appropriate information model.
    'Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
    'EndpointDescriptor.UrlString = "opc.tcp://localhost:48010"
    'Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor)

    ' Get the names of PubSub connections in the configuration.
    Dim PubSubConnectionNames: Set PubSubConnectionNames =
PubSubConfiguration.ListPubSubConnectionNames
    Dim pubSubConnectionName: For Each pubSubConnectionName In PubSubConnectionNames
        WScript.Echo "PubSub connection: " & pubSubConnectionName

        ' You can use the statement below to obtain parameters of the PubSub
connection.
        'Dim PubSubConnectionElement: Set PubSubConnectionElement =
PubSubConfiguration.GetConnectionElement(pubSubConnectionName)

        ' Get names of the writer groups on this PubSub connection.
        Dim WriterGroupNames: Set WriterGroupNames =
PubSubConfiguration.ListWriterGroupNames(pubSubConnectionName)
        Dim writerGroupName: For Each writerGroupName In WriterGroupNames
            WScript.Echo "  Writer group: " & writerGroupName

            ' You can use the statement below to obtain parameters of the writer group.
            'Dim WriterGroupElement: Set WriterGroupElement =
PubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName)

            ' Get names of the dataset writers on this writer group.
            Dim DataSetWriterNames: Set DataSetWriterNames =
PubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName)
            Dim dataSetWriterName: For Each dataSetWriterName In DataSetWriterNames
                WScript.Echo "    Dataset writer: " & dataSetWriterName

                ' You can use the statement below to obtain parameters of the dataset
writer.
                'Dim DataSetWriterElement: Set DataSetWriterElement = _
                '    PubSubConfiguration.GetDataSetWriterElement(pubSubConnectionName,
writerGroupName, dataSetWriterName)
            Next
        Next
    Next
```

```
End Sub


' Example output:
'
'Loading the configuration...
'PubSub connection: FixedLayoutConnection
'  Writer group: FixedLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: AllTypesWriter
'    Dataset writer: MassTestWriter
'PubSub connection: DynamicLayoutConnection
'  Writer group: DynamicLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: MassTestWriter
'    Dataset writer: AllTypes-DynamicWriter
'    Dataset writer: EventSimpleWriter
'PubSub connection: FlexibleLayoutConnection
'  Writer group: FlexibleLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: MassTestWriter
'    Dataset writer: AllTypes-DynamicWriter
'Finished.
```

The example below uses the ListAllPublishedDataSetNames Method to retrieve all published dataset from the PubSub configuration. The published datasets are actually organized in the PubSub configuration using a hierarchical structure of dataset folders. The word "all" in the method name denotes that the method will truly return all published datasets in the configuration, not just those at the "root" of the dataset folder structure, or in any specified dataset folder. The method will act recursively in the dataset folder structure, if needed. There are also methods that allow you to work with contents of individual dataset folders. Note that the published dataset names are unique across the PubSub configuration as a whole (regardless of their location in the dataset folder structure), and therefore a published dataset name is sufficient to identify the published dataset, and the dataset folder path is not strictly necessary.

C#

```
// This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

using System;
using OpcLabs.BaseLib.Collections.Specialized;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions;

namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
{
    partial class GetMethods
    {
        public static void PublishedDataSets()
        {
```

```
            // Instantiate the publish-subscribe client object.
            var publishSubscribeClient = new EasyUAPublishSubscribeClient();

            try
            {
                Console.WriteLine("Loading the configuration...");
                // Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                // specified that it has to be copied to the project's output
directory.
                IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary");

                // Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                // Server with appropriate information model.
                //IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                //
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010");

                // Get the names of all published datasets in the PubSub configuration.
                StringCollection publishedDataSetNames =
pubSubConfiguration.ListAllPublishedDataSetNames();

                foreach (string publishedDataSetName in publishedDataSetNames)
                {
                    Console.WriteLine($"Published dataset: {publishedDataSetName}");

                    // You can use the statement below to obtain parameters of the
published dataset.
                    //UAPublishedDataSetElement publishedDataSetElement =
                    //
pubSubConfiguration.GetPublishedDataSetElement(publishedDataSetName);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine($"*** Failure:
{uaException.InnerException.Message}");
            }

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Loading the configuration...
        //Published dataset: Simple
        //Published dataset: AllTypes
        //Published dataset: MassTest
        //Published dataset: AllTypes-Dynamic
        //Finished.
    }
}
```

VB.NET

```
' This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions

Namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
    Partial Friend Class GetMethods
        Public Shared Sub PublishedDataSets()

            ' Instantiate the publish-subscribe client object.
            Dim publishSubscribeClient = New EasyUAPublishSubscribeClient()

            Try
                Console.WriteLine("Loading the configuration...")
                ' Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                ' specified that it has to be copied to the project's output directory.
                Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary")

                ' Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                ' Server with appropriate information model.
                'Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                '
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010")

                ' Get the names of all published datasets in the PubSub configuration.
                Dim publishedDataSetNames =
pubSubConfiguration.ListAllPublishedDataSetNames()

                For Each publishedDataSetName As String In publishedDataSetNames
                    Console.WriteLine($"Published dataset: {publishedDataSetName}")

                    ' You can use the statement below to obtain parameters of the
published dataset.
                    'Dim publishedDataSetElement As UAPublishedDataSetElement =
                    '
pubSubConfiguration.GetPublishedDataSetElement(publishedDataSetName)
                Next publishedDataSetName
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try

            Console.WriteLine("Finished...")
        End Sub
    End Class
```

```
    ' Example output
    '
    'Loading the configuration...
    'Published dataset Simple
    'Published dataset: AllTypes
    'Published dataset: MassTest
    'Published dataset: AllTypes-Dynamic
    'Finished.

End Namespace
```

Object Pascal

```
// This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

class procedure GetMethods.PublishedDataSets;
var
  //EndpointDescriptor: _UAEndpointDescriptor;
  I: Integer;
  PublishedDataSetName: string;
  PublishedDataSetNames: _StringCollection;
  //PublishedDataSetElement: _UAPublishedDataSetElement;
  PublishSubscribeClient: _EasyUAPublishSubscribeClient;
  PubSubConfiguration: _UAReadOnlyPubSubConfiguration;
begin
  // Instantiate the publish-subscribe client object.
  PublishSubscribeClient := CoEasyUAPublishSubscribeClient.Create;

  try
    WriteLn('Loading the configuration...');
    // Load the PubSub configuration from a file. The file itself is included alongside
the script.
    PubSubConfiguration :=
PublishSubscribeClient.LoadReadOnlyConfiguration('UADemoPublisher-Default.uabinary');

    // Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    // with appropriate information model.
    //EndpointDescriptor := CoUAEndpointDescriptor.Create;
    //EndpointDescriptor.UrlString := 'opc.tcp://localhost:48010';
    //PubSubConfiguration :=
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor);

    // Get the names of PubSub connections in the configuration, regardless of the
folder they reside in.
    PublishedDataSetNames := PubSubConfiguration.ListAllPublishedDataSetNames;

    for I := 0 to PublishedDataSetNames.Count-1 do
    begin
      PublishedDataSetName := PublishedDataSetNames[I];
      WriteLn('Published dataset: ', PublishedDataSetName);

      // You can use the statement below to obtain parameters of the published dataset.
      //PublishedDataSetElement :=
PubSubConfiguration.GetPublishedDataSetElement(Unassigned, PublishedDataSetName);
    end;
```

```
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Finished.');
end;

// Example output:
//
//Loading the configuration...
//Published dataset: Simple
//Published dataset: AllTypes
//Published dataset: MassTest
//Published dataset: AllTypes-Dynamic
//Finished.
```

VBScript

```
Rem This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

Option Explicit

' Instantiate the publish-subscribe client object.
Dim PublishSubscribeClient: Set PublishSubscribeClient =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.InformationModel.EasyUAPublishSubscribeClient")

On Error Resume Next
DumpPublishedDataSets
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Finished."



Sub DumpPublishedDataSets()
    WScript.Echo "Loading the configuration..."
    ' Load the PubSub configuration from a file. The file itself is included alongside
the script.
    Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-Default.uabinary")

    ' Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    ' with appropriate information model.
    'Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
    'EndpointDescriptor.UrlString = "opc.tcp://localhost:48010"
    'Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor)
```

```
    ' Get the names of PubSub connections in the configuration, regardless of the
folder they reside in.
    Dim PublishedDataSetNames: Set PublishedDataSetNames =
PubSubConfiguration.ListAllPublishedDataSetNames

    Dim publishedDataSetName: For Each publishedDataSetName In PublishedDataSetNames
        WScript.Echo "Published dataset: " & publishedDataSetName

        ' You can use the statement below to obtain parameters of the published
dataset.
        'Dim PublishedDataSetElement: Set PublishedDataSetElement =
PubSubConfiguration.GetPublishedDataSetElement(Nothing, publishedDataSetName)
    Next
End Sub



' Example output:
'
'Loading the configuration...
'Published dataset: Simple
'Published dataset: AllTypes
'Published dataset: MassTest
'Published dataset: AllTypes-Dynamic
'Finished.
```

5.1.6 Event Pull Mechanism

In This Topic

Event pull mechanism is an alternative to the traditional delivery mechanism of notifications through event handlers. Setting up event handlers, and properly handling events, is difficult or impossible in some (especially COM-based) tools (e.g. Python) or in some environments (e.g. applications running under Web server). In COM, unless taken care of already by the tool, the developer must also assure that (in an STA model) Windows messages are being pumped by the application.

Note: This feature should only be used when the traditional event delivery approach is not feasible. It is unlikely that you will need to use the event pull mechanism in C# or VB.NET (except maybe for server-side of certain Web apps and Web services).

Event pull mechanism replaces the event handlers by methods that allow the developer to obtain a next available notification from an internally provided queue. Events can then be handled by a loop that calls the “pull” method and processes the notifications. In order to maintain reasonable performance, a so-called long poll approach is used: The method only returns after a notification becomes available, or a specified timeout elapses. Note that the event pull concept is strictly confined to how the OPC Data Client exchanges notifications with the enclosing application, and is not related to how OPC Data Client communicates with OPC servers.

For each event **XXXX**, by convention, two methods for event pull mechanism are provided: A PullXXXX method, which allows to retrieve a single notification, and a PullMultipleXXXX method, which allows to retrieve multiple notifications, if they are available. Although currently is no significant performance difference between the two methods, it is recommended that your code uses the PullMultipleXXXX methods, because it may bring performance benefits in the

future.

The following table shows the traditional events, and their corresponding methods for event pull:

Type	Event	Pull method	PullMultiple method
EasyDAClient	ItemChanged	PullItemChanged	PullMultipleItemChanges
EasyAEClient	Notification	PullNotification	PullMultipleNotifications
EasyUAClient	DataChangeNotification	PullDataChangeNotification	PullMultipleDataChangeNotifications
	EventNotification	PullEventNotification	PullMultipleEventNotifications
	ServerConditionChanged	PullServerConditionChanged	PullMultipleServerConditionChanges
EasyUASubscriber Class	DataSetMessage Event	PullDataSetMessage Method	PullMultipleDataSetMessages Method
	ResolverAccess Event	PullResolverAccess Method	PullMultipleResolverAccesses Method

Each PullXXXX or PullMultipleXXXX method takes an argument that specifies a timeout (in milliseconds). When a notification is available, the methods returns it immediately, and return a non-null object containing the event arguments (or an array of such event arguments, for PullMultipleXXXX). If no notifications are available during the specified period (timeout), the methods returns after the period elapses, but the return value is a null reference.

After your code receives a non-null event arguments object, it can process it as it would do in a traditional event handler.

You will probably be calling the PullXXXX or PullMultipleXXXX method in a loop with some termination condition. Specify a timeout that will allow the loop to react to the termination condition quickly enough. You should not, however, use very low (or zero) timeouts in such a loop, because that would lead to unnecessary high CPU consumption.

In order to use the event pull mechanism, your code needs to set up a queue with a fixed capacity upfront. This is done by setting a corresponding property in the EasyXXClient object. By convention, for an event named **XXXX**, the property name is PullXXXXQueueCapacity. For example, set the EasyUAClient.PullDataChangeNotificationQueueCapacity property to allocate a queue for OPC UA data change notifications.

If you do not set the queue capacity to a positive (non-zero) value, and attempt to use the event pull mechanism, the PullXXXX or PullMultipleXXXX method will throw an InvalidOperationException.

Examples

Examples for OPC Data Access:

C#

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    class PullItemChanged
    {
```

```
        public static void Main1()
        {
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyDAClient { PullItemChangedQueueCapacity = 1000 };

            Console.WriteLine("Subscribing item changes...");
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

            Console.WriteLine("Processing item changes for 1 minute...");
            int endTick = Environment.TickCount + 60 * 1000;
            do
            {
                EasyDAItemChangedEventArgs eventArgs = client.PullItemChanged(2 *
1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);

            Console.WriteLine("Unsubscribing item changes...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to item changes and obtain the events by pulling
them.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class PullItemChanged
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            client.PullItemChangedQueueCapacity = 1000

            Console.WriteLine("Subscribing item changes...")
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000)

            Console.WriteLine("Processing item changes for 1 minute...")
            Dim endTick As Integer = Environment.TickCount + 60 * 1000
            Do
                Dim eventArgs As EasyDAItemChangedEventArgs = client.PullItemChanged(2
* 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
                End If
            Loop While Environment.TickCount < endTick
```

```
            Console.WriteLine("Unsubscribing item changes...")
            client.UnsubscribeAllItems()

            Console.WriteLine("Finished.")
        End Sub
    End Class
End Namespace
```

JScript

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullItemChangedQueueCapacity = 1000;

WScript.Echo("Subscribing item changes...");
Client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

WScript.Echo("Processing item changes for 1 minute...");
var endTime = new Date().getTime() + 60*1000
do {
    var EventArgs = Client.PullItemChanged(2*1000);
    if (EventArgs !== null) {
        //  Handle the notification event
        WScript.Echo(EventArgs);
    }
} while(new Date().getTime() < endTime);

WScript.Echo("Unsubscribing item changes...");
Client.UnsubscribeAllItems();

WScript.Echo("Finished.");
```

Object Pascal

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

class procedure PullItemChanged.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  EndTick: Cardinal;
  EventArgs: _EasyDAItemChangedEventArgs;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullItemChangedQueueCapacity := 1000;

  WriteLn('Subscribing item changes...');
  Client.SubscribeItem('', 'OPCLabs.KitServer.2', 'Simulation.Random', 1000);

  WriteLn('Processing item changes for 1 minute...');
  EndTick := Ticks + 60*1000;
```

```
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullItemChanged(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing item changes...');
  Client.UnsubscribeAllItems;

  WriteLn('Finished.');
end;
```

PHP

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
$Client->PullItemChangedQueueCapacity = 1000;

print "Subscribing item changes...\n";
$Client->SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

print "Processing item changed events for 1 minute...\n";
$endTime = time() + 60;
do {
    $EventArgs = $Client->PullItemChanged(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        print $EventArgs->ToString();
        print "\n";
    }
} while (time() < $endTime);
```

PowerScript

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
// In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullItemChangedQueueCapacity = 1000

mle_outputtext.Text = mle_outputtext.Text + "Subscribing item changes..." + "~r~n"
client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000)

mle_outputtext.Text = mle_outputtext.Text + "Processing item changes for 1 minute..." +
"~r~n"
```

```
Time endTime = RelativeTime(Now(), 60)
DO
    OLEObject eventArgs
    eventArgs = client.PullItemChanged(2*1000)
    IF NOT IsNull(eventArgs) THEN
        // Handle the notification event
        mle_outputtext.Text = mle_outputtext.Text + eventArgs.DisplayString + "~r~n"
    END IF
LOOP WHILE Now() < endTime

mle_outputtext.Text = mle_outputtext.Text + "Unsubscribing item changes..." + "~r~n"
client.UnsubscribeAllItems()

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to subscribe to item changes and obtain the events by pulling
them.

import time
import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.DataAccess.EasyDAClient')
# In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullItemChangedQueueCapacity = 1000

print('Subscribing item changes...')
client.SubscribeItem('', 'OPCLabs.KitServer.2', 'Simulation.Random', 1000)

print('Processing item changes for 1 minute...')
endTime = time.time() + 60
while time.time() < endTime:
    eventArgs = client.PullItemChanged(2*1000)
    if eventArgs is not None:
        # Handle the notification event
        print(eventArgs)

print('Unsubscribing item changes...')
client.UnsubscribeAllItems()

print('Finished.')
```

VBScript

```
Rem This example shows how to subscribe to item changes and obtain the events by
pulling them.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullItemChangedQueueCapacity = 1000

WScript.Echo "Subscribing item changes..."
```

```
Client.SubscribeItem "", "OPCLabs.KitServer.2", "Simulation.Random", 1000

WScript.Echo "Processing item changes for 1 minute..."
Dim endTime: endTime = Now() + 60*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullItemChanged(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime

WScript.Echo "Unsubscribing item changes..."
Client.UnsubscribeAllItems

WScript.Echo "Finished."
```

Examples for OPC Alarms&Events:

C#

```
// This example shows how to subscribe to events and obtain the notification events by
pulling them.

using System;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class PullNotification
    {
        public static void Main1()
        {
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            using (var client = new EasyAEClient { PullNotificationQueueCapacity = 1000
})
            {
                Console.WriteLine("Subscribing events...");
                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
1000);

                Console.WriteLine("Processing event notifications for 1 minute...");
                int endTick = Environment.TickCount + 60 * 1000;
                do
                {
                    EasyAENotificationEventArgs eventArgs = client.PullNotification(2 *
1000);
                    if (eventArgs != null)
                        // Handle the notification event
                        Console.WriteLine(eventArgs);
                } while (Environment.TickCount < endTick);

                Console.WriteLine("Unsubscribing events...");
                client.UnsubscribeEvents(handle);
```

```
                Console.WriteLine("Finished.");
            }
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events and obtain the notification events by
pulling them.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
    Partial Friend Class PullNotification
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                ' In order to use event pull, you must set a non-zero queue capacity
upfront.
                client.PullNotificationQueueCapacity = 1000

                Console.WriteLine("Subscribing events...")
                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Processing event notifications for 1 minute...")
                Dim endTick As Integer = Environment.TickCount + 60 * 1000
                Do
                    Dim eventArgs As EasyAENotificationEventArgs =
client.PullNotification(2 * 1000)
                    If Not eventArgs Is Nothing Then
                        ' Handle the notification event
                        Console.WriteLine(eventArgs)
                    End If
                Loop While Environment.TickCount < endTick

                Console.WriteLine("Unsubscribing events...")
                client.UnsubscribeEvents(handle)
            End Using

            Console.WriteLine("Finished.")
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to events and obtain the notification events by
pulling them.

class procedure PullNotification.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyAEClient;
  EndTick: Cardinal;
  EventArgs: _EasyAENotificationEventArgs;
```

```
  Handle: Integer;
  ServerDescriptor: _ServerDescriptor;
  State: OleVariant;
  SubscriptionParameters: _AESubscriptionParameters;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitEventServer.2';

  // Instantiate the client object
  Client := CoEasyAEClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullNotificationQueueCapacity := 1000;

  WriteLn('Subscribing events...');
  SubscriptionParameters := CoAESubscriptionParameters.Create;
  SubscriptionParameters.NotificationRate := 1000;
  Handle := Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters, true,
State);

  WriteLn('Processing event notifications for 1 minute...');
  EndTick := Ticks + 60*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing events...');
  Client.UnsubscribeEvents(Handle);

  WriteLn('Finished.');
end;
```

PHP

```
// This example shows how to subscribe to events and obtain the notification events by
pulling them.

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitEventServer.2";

$Client = new COM("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
$Client->PullNotificationQueueCapacity = 1000;

print "Subscribing events...\n";
$SubscriptionParameters = new
COM("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters");
$SubscriptionParameters->NotificationRate = 1000;
$handle = $Client->SubscribeEvents($ServerDescriptor, $SubscriptionParameters, TRUE,
NULL);

print "Processing event notifications for 1 minute...\n";
$endTime = time() + 60;
do {
```

```
    $EventArgs = $Client->PullNotification(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        print $EventArgs->ToString();
        print "\n";
    }
} while (time() < $endTime);

print "Unsubscribing events...\n";
$Client->UnsubscribeEvents($handle);

print "Finished.\n";
```

VBScript

```
Rem This example shows how to subscribe to events and obtain the notification events by
pulling them.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullNotificationQueueCapacity = 1000

WScript.Echo "Subscribing events..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Processing event notifications for 1 minute..."
Dim endTime: endTime = Now() + 60*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullNotification(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime

WScript.Echo "Unsubscribing events..."
Client.UnsubscribeEvents handle

WScript.Echo "Finished."
```

Examples for OPC XML-DA:

C#

```
// This example shows how to subscribe to OPC XML-DA item changes and obtain the events
by pulling them.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class PullItemChanged
    {
        public static void Main1Xml()
        {
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyDAClient { PullItemChangedQueueCapacity = 1000 };

            Console.WriteLine("Subscribing item changes...");
            client.SubscribeItem(
                "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                "Dynamic/Analog Types/Int",
                1000,
                state: null);

            Console.WriteLine("Processing item changes for 1 minute...");
            int endTick = Environment.TickCount + 60 * 1000;
            do
            {
                EasyDAItemChangedEventArgs eventArgs = client.PullItemChanged(2 *
1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);

            Console.WriteLine("Unsubscribing item changes...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to OPC XML-DA item changes and obtain the events
by pulling them.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class PullItemChanged
        Public Shared Sub Main1Xml()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            Dim client = New EasyDAClient() With {.PullItemChangedQueueCapacity = 1000}
```

```
            Console.WriteLine("Subscribing item changes...")
            client.SubscribeItem(
                    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                    "Dynamic/Analog Types/Int",
                    1000,
                    state:=Nothing)

            Console.WriteLine("Processing item changes for 1 minute...")
            Dim endTick As Integer = Environment.TickCount + 60 * 1000
            Do
                Dim eventArgs As EasyDAItemChangedEventArgs = client.PullItemChanged(2
* 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
                End If
            Loop While Environment.TickCount < endTick

            Console.WriteLine("Unsubscribing item changes...")
            client.UnsubscribeAllItems()

            Console.WriteLine("Finished.")
        End Sub
    End Class
End Namespace
```

Examples for OPC UA (Data):

C#

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class PullDataChangeNotification
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyUAClient { PullDataChangeNotificationQueueCapacity =
1000 };
```

```
            Console.WriteLine("Subscribing...");
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000);

            Console.WriteLine("Processing data change events for 1 minute...");
            int endTick = Environment.TickCount + 60 * 1000;
            do
            {
                EasyUADataChangeNotificationEventArgs eventArgs =
client.PullDataChangeNotification(2 * 1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class PullDataChangeNotification
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            client.PullDataChangeNotificationQueueCapacity = 1000

            Console.WriteLine("Subscribing...")
            client.SubscribeDataChange(
                endpointDescriptor,
                "nsu=http://test.org/UA/Data/;i=10853",
                1000)    ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

            Console.WriteLine("Processing data change events for 1 minute...")
            Dim endTick As Integer = Environment.TickCount + 60 * 1000
            Do
                Dim eventArgs As EasyUADataChangeNotificationEventArgs =
client.PullDataChangeNotification(2 * 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
```

```
                End If
            Loop While Environment.TickCount < endTick
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

class procedure PullDataChangeNotification.Main;
var
  Client: EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUADataChangeNotificationEventArgs;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullDataChangeNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing data change events for 1 minute...');
  EndTick := GetTickCount + 60*1000;
  while GetTickCount < EndTick do
  begin
    EventArgs := Client.PullDataChangeNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;
```

JScript

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

var Client = new ActiveXObject("OpcLabs.EasyOpc.UA.EasyUAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullDataChangeNotificationQueueCapacity = 1000;

WScript.Echo("Subscribing...");
Client.SubscribeDataChange("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000);
```

```
WScript.Echo("Processing data change events for 1 minute...");
var endTime = new Date().getTime() + 60*1000
do {
    var EventArgs = Client.PullDataChangeNotification(2*1000);
    if (EventArgs !== null) {
        //  Handle the notification event
        WScript.Echo(EventArgs);
    }
} while(new Date().getTime() < endTime);
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

class procedure PullDataChangeNotification.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUADataChangeNotificationEventArgs;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullDataChangeNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing data change events for 1 minute...');
  EndTick := Ticks + 60*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullDataChangeNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;
```

PHP

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
$Client->PullDataChangeNotificationQueueCapacity = 1000;
```

```
print "Subscribing...\n";
$Client->SubscribeDataChange("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000);

print "Processing data change events for 1 minute...\n";
$endTime = time() + 60;
do {
    $EventArgs = $Client->PullDataChangeNotification(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        print $EventArgs->ToString();
        print "\n";
    }
} while (time() < $endTime);
```

PowerScript

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")
// In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullDataChangeNotificationQueueCapacity = 1000

mle_outputtext.Text = mle_outputtext.Text + "Subscribing data changes..." + "~r~n"
client.SubscribeDataChange("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000)

mle_outputtext.Text = mle_outputtext.Text + "Processing data change events for 1
minute..." + "~r~n"

Time endTime = RelativeTime(Now(), 60)
DO
    OLEObject eventArgs
    eventArgs = client.PullDataChangeNotification(2*1000)
    IF NOT IsNull(eventArgs) THEN
        // Handle the notification event
        mle_outputtext.Text = mle_outputtext.Text + eventArgs.DisplayString + "~r~n"
    END IF
LOOP WHILE Now() < endTime

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

import time
import win32com.client
```

```
# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')
# In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullDataChangeNotificationQueueCapacity = 1000

print('Subscribing...')
client.SubscribeDataChange('http://opcua.demo-this.com:51211/UA/SampleServer',
'nsu=http://test.org/UA/Data/;i=10853', 1000)

print('Processing data change events for 1 minute...')
endTime = time.time() + 60
while time.time() < endTime:
    eventArgs = client.PullDataChangeNotification(2*1000)
    if eventArgs is not None:
        # Handle the notification event
        print(eventArgs)
```

VBScript

```
Rem This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullDataChangeNotificationQueueCapacity = 1000

WScript.Echo "Subscribing..."
Client.SubscribeDataChange "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000

WScript.Echo "Processing data change events for 1 minute..."
Dim endTime: endTime = Now() + 60*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullDataChangeNotification(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime
```

Examples for OPC UA (Alarms & Conditions):

C#

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples.AlarmsAndConditions
{
    class PullEventNotification
    {
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyUAClient { PullEventNotificationQueueCapacity = 1000
};

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            int endTick = Environment.TickCount + 30 * 1000;
            do
            {
                EasyUAEventNotificationEventArgs eventArgs =
client.PullEventNotification(2 * 1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);
        }

        // Example output (truncated):
        //Subscribing...
        //Processing event notifications for 30 seconds...
        //[] Success
        //[] Success; Refresh; RefreshInitiated
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was
activated" @9/10/2019 8:08:23 PM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity
has increased." @9/10/2019 8:09:07 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity
has increased." @9/10/2019 8:10:09 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity
has increased." @11/9/2019 10:29:42 AM
```

```
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:11 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:10:19 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity
has increased." @11/9/2019 10:29:42 AM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity
has increased." @9/10/2019 8:08:58 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity
has increased." @9/10/2019 8:09:48 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity
has increased." @11/9/2019 10:29:42 AM
        //[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:02 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:59 PM
        //[] Success; Refresh; RefreshComplete
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:43 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:43 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:44 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:44 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:45 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:45 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:46 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:46 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:47 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:47 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:48 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:48 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:49 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:49 AM
```

```
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:50 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:50 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:51 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:51 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:52 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:52 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:53 AM
        //[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 10:29:53 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:53 AM
        //[] Success; (10 field results) [WestTank] 500! "The alarm severity has
increased." @11/9/2019 10:29:53 AM
        //[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 10:29:53 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:54 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:54 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:55 AM
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

class procedure PullEventNotification.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUAEventNotificationEventArgs;
begin
  // Instantiate the client object and hook events
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullEventNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(
    'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer',
    'nsu=http://opcfoundation.org/UA/;i=2253',  // UAObjectIds.Server
    1000);

  WriteLn('Processing event notifications for 30 seconds...');
  EndTick := Ticks + 30*1000;
  while Ticks < EndTick do
```

```
  begin
    EventArgs := Client.PullEventNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity has
```

```
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:43
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:43 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:44
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:44 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:45
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:45 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:46
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:46 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:47
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:47 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:48
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:48 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:49
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:49 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:50
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:50 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:51
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:51 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:52
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:52 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:53
AM
//[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:53 AM
//[] Success; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:54
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:54 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:55
AM
//...
```

PHP

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.


// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$Client->PullEventNotificationQueueCapacity = 1000;

printf("Subscribing...\n");
$Client->SubscribeEvent(
    "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
    "nsu=http://opcfoundation.org/UA/;i=2253",  // UAObjectIds_Server
    1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time();
do {
    $EventArgs = $Client->PullEventNotification(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        printf("%s\n", $EventArgs);
    }
} while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity has
```

```
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:43
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:43 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:44
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:44 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:45
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:45 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:46
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:46 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:47
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:47 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:48
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:48 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:49
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:49 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:50
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:50 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:51
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:51 AM
```

```
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:52
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:52 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:53
AM
//[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:53 AM
//[] Success; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:54
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:54 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:55
AM
//...
```

VB.NET

```
' This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class PullEventNotification
        Public Shared Sub Main1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            client.PullEventNotificationQueueCapacity = 1000

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent(
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
                UAObjectIds.Server,
                1000)

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Dim endTick As Integer = Environment.TickCount + 30 * 1000
            Do
                Dim eventArgs As EasyUAEventNotificationEventArgs =
client.PullEventNotification(2 * 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
                End If
            Loop While Environment.TickCount < endTick
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

class procedure PullEventNotification.Main;
var
  Client: EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUAEventNotificationEventArgs;
begin
  // Instantiate the client object and hook events
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullEventNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(
    'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer',
    'nsu=http://opcfoundation.org/UA/;i=2253',  // UAObjectIds.Server
    1000);

  WriteLn('Processing event notifications for 30 seconds...');
  EndTick := GetTickCount + 60*1000;
  while GetTickCount < EndTick do
  begin
    EventArgs := Client.PullEventNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;
```

VBScript

```
Rem This example shows how to subscribe to event notifications, pull events, and
display each incoming event.

Option Explicit

Const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullEventNotificationQueueCapacity = 1000

WScript.Echo "Subscribing..."
Client.SubscribeEvent "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", UAObjectIds_Server, 1000
```

```
WScript.Echo "Processing event notifications for 30 seconds..."
Dim endTime: endTime = Now() + 30*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullEventNotification(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime



' Example output (truncated):
'Subscribing...
'Processing event notifications for 30 seconds...
'[] Success
'[] Success; Refresh; RefreshInitiated
'[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:13 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was acknoweledged."
@11/9/2019 9:56:23 AM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:17 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:12 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:04 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
```

```
'[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
'[] Success; Refresh; RefreshComplete
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:43 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:43 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:44 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:44 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:45 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:45 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:46 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:46 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:47 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:47 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:48 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:48 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:49 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:49 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:50 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:50 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:51 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:51 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:52 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:52 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:53 AM
'[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:53 AM
'[] Success; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
'[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:54 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:54 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:55 AM
'...
```

Examples for OPC UA PubSub:

The example below subscribes to all dataset messages on an OPC-UA PubSub connection with UDP UADP mapping. The connection is specified by its physical parameters, using the scheme name "opc.udp" and the IP address of the multicast group to listen on.

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
// the incoming datasets.
```

```
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    class PullDataSetMessage
    {
        public static void Main1()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor =
"opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Instantiate the subscriber object.
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var subscriber = new EasyUASubscriber {PullDataSetMessageQueueCapacity =
1000};

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            int endTick = Environment.TickCount + 20 * 1000;
            do
            {
                EasyUADataSetMessageEventArgs eventArgs =
subscriber.PullDataSetMessage(2 * 1000);
                if (eventArgs != null)
                {
                    // Display the dataset.
                    if (eventArgs.Succeeded)
                    {
                        // An event with null DataSetData just indicates a successful
connection.
                        if (!(eventArgs.DataSetData is null))
                        {
                            Console.WriteLine();
                            Console.WriteLine($"Dataset data:
{eventArgs.DataSetData}");
                            foreach (KeyValuePair<string, UADataSetFieldData> pair in
eventArgs.DataSetData.FieldDataDictionary)
                                Console.WriteLine(pair);
```

```
                    }
                }
                else
                {
                    Console.WriteLine();
                    Console.WriteLine($"*** Failure:
{eventArgs.ErrorMessageBrief}");
                }
            }
        } while (Environment.TickCount < endTick);

        Console.WriteLine("Unsubscribing...");
        subscriber.UnsubscribeAllDataSets();

        Console.WriteLine("Waiting for 1 second...");
        // Unsubscribe operation is asynchronous, messages may still come for a
short while.
        Thread.Sleep(1 * 1000);

        Console.WriteLine("Finished.");
    }

    // Example output:
    //
    //Subscribing...
    //Processing dataset message events for 20 seconds...
    //
    //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
    //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
    //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-
01-01T00:00:00.000; Good]
    //
    //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-
a715-0979884b55ae, fields: 100
    //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
```

```
Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
' the incoming datasets.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
the interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class PullDataSetMessage
        Public Shared Sub Main1()

            ' Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor =
"opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Instantiate the subscriber object.
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            Dim subscriber = New EasyUASubscriber()
            subscriber.PullDataSetMessageQueueCapacity = 1000

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Dim endTick As Integer = Environment.TickCount + 20 * 1000
            Do
                Dim eventArgs As EasyUADataSetMessageEventArgs =
subscriber.PullDataSetMessage(2 * 1000)
                If Not eventArgs Is Nothing Then
                    ' Display the dataset.
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful
connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
```

```
                            Console.WriteLine("Dataset data: {0}",
eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String,
UADataSetFieldData) In eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                    End If
                End If
            Loop While Environment.TickCount < endTick

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    ''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
    '[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
    '
    'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
    '[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '...
```

```
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
// the incoming datasets.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.


class procedure PullDataSetMessage.Main1;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Count: Cardinal;
  Element: OleVariant;
  EndTick: Cardinal;
  EventArgs: _EasyUADataSetMessageEventArgs;
  PairEnumerator: IEnumVARIANT;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: OpcLabs_EasyOpcUA_TLB._EasyUASubscriber;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor :=
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString :=
'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs
to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  // ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Instantiate the subscriber object.
  Subscriber := CoEasyUASubscriber.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Subscriber.PullDataSetMessageQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message events for 20 seconds...');
  EndTick := Ticks + 20*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Subscriber.PullDataSetMessage(2*1000);
    if EventArgs <> nil then
      begin
        // Display the dataset.
        if EventArgs.Succeeded then
        begin
          // An event with null DataSetData just indicates a successful connection.
          if EventArgs.DatasetData <> nil then
          begin
```

```
            WriteLn;
            WriteLn('Dataset data: ', EventArgs.DatasetData.ToString);
            PairEnumerator := EventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
            while (PairEnumerator.Next(1, Element, Count) = S_OK) do
            begin
              WriteLn(Element.ToString);
            end;
          end;
        end
        else
        begin
          WriteLn;
             WriteLn(' *** Failure: ', EventArgs.ErrorMessageBrief);
        end;
      end;
  end;

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

VBScript

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
Rem the incoming datasets.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher
tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

Option Explicit

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")

Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString =
"opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to
be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Instantiate the subscriber object.
Dim Subscriber: Set Subscriber =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Subscriber.PullDataSetMessageQueueCapacity = 1000

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
Dim endTime: endTime = Now() + 20*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Subscriber.PullDataSetMessage(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Display the dataset.
        If EventArgs.Succeeded Then
            ' An event with null DataSetData just indicates a successful connection.
            If Not (EventArgs.DataSetData Is Nothing) Then
                WScript.Echo
                WScript.Echo "Dataset data: " & EventArgs.DataSetData
                Dim Pair: For Each Pair in EventArgs.DataSetData.FieldDataDictionary
                    WScript.Echo Pair
                Next
            End If
        Else
            WScript.Echo
            WScript.Echo "*** Failure: " & EventArgs.ErrorMessageBrief
        End If
    End If
Loop While Now() < endTime

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets
```

```
WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
Rem the incoming datasets.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher
tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

Private Sub PullDataSetMessage_Main1_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments = New
EasyUASubscribeDataSetArguments
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
```

```
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString =
"opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs
to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    'ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Instantiate the subscriber object.
    Dim Subscriber: Set Subscriber = New EasyUASubscriber
    ' In order to use event pull, you must set a non-zero queue capacity upfront.
    Subscriber.PullDataSetMessageQueueCapacity = 1000

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." &
vbCrLf
    Dim endTime: endTime = Now() + 20 * (1 / 24 / 60 / 60)
    Do
        Dim EventArgs: Set EventArgs = Subscriber.PullDataSetMessage(2 * 1000)
        If Not (EventArgs Is Nothing) Then
            ' Display the dataset.
            If EventArgs.Succeeded Then
                ' An event with null DataSetData just indicates a successful
connection.
                If Not (EventArgs.DataSetData Is Nothing) Then
                    OutputText = OutputText & vbCrLf
                    OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData
& vbCrLf
                    Dim Pair: For Each Pair In
EventArgs.DataSetData.FieldDataDictionary
                        OutputText = OutputText & Pair & vbCrLf
                    Next
                End If
            Else
                OutputText = OutputText & vbCrLf
                OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief
& vbCrLf
            End If
        End If
    Loop While Now() < endTime

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
```

```
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

More information

Besides the usual ArgumentException (when your code passes in an improper argument), the InvalidOperationException described above (which also indicates a bug in the code), and system exceptions such OutOfMemoryException, the PullXXXX and PullMultipleXXXX methods can only throw one type of exception, and that is ProcedureCallException. The current implementation does not actually throw this exception type, but future implementations may do so, and we recommend that your code is written in such a way that is ready for that. The ProcedureCallException indicates a failure in the event delivery mechanism (and a likely loss of one or more notifications).

Make sure you always specify a queue capacity that will provide a "buffer" of sufficient length for a difference between the rate of incoming notifications, and the rate your code can consume them. Besides an indication through the event log output (currently in OPC UA only), there is no way to detect queue overflows.

5.1.7 Error Handling

Various kinds of errors may be returned by OPC Data Client, e.g.:

- Errors returned by system calls when performing OPC-related operations.
- In OPC Classic, errors returned by COM/DCOM infrastructure, including RPC errors.
- In OPC UA, errors returned by communication infrastructure, including Web service errors.
- Error returned by OPC-UA Stack or SDK.
- Network-related errors.
- Errors reported by the OPC server you are connecting to.
- Errors detected by the OPC Data Client library.

In general, you cannot reliably prevent all these errors from happening. Many conditions external to your code can cause

OPC failures, e.g. network problems, improper OPC server configuration, etc. For this reason, you should always expect than OPC operation can fail.

OPC Data Client "Classic" and OPC Data Client-UA each define one new type of exception, called OpcException (for OPC "Classic"), or UAException (for OPC-UA), derived from the Exception object. This exception is for all errors arising from OPC operations.

More details about the cause of the problem can be found by interrogating the InnerException property of OpcException or UAException, or by examining the ErrorId property. In most scenarios, however, your code will be handling all OpcException-s or UAException-s in the same way, independent of the inner exception or error code.

The InnerException can be:

- COMException, **especially common in OPC Classic**.
- UAStatusCodeException, when the OPC-UA status code is not as expected.
- UAEngineException, for errors that originates in the OPC-UA engine and not in the OPC service (or OPC-UA SDK).
- UAServiceException, when an OPC-UA defined error occurs.
- LicenseException, if there is a problem with component's license in OPC Data Client-UA.
- An exception from the lower level systems, such as .NET Framework, WCF, OPC-UA stack or SDK.

For more details, see **OPC Classic Errors (Section 5.1.7.3)** or **OPC UA Errors (Section 5.1.7.4)**, respectively.

Because many errors are provided by the infrastructure that OPC Data Client is using, there is no way to provide a comprehensive list of all possible errors. You can, however, be sure that all error will be reported in the way described here, and therefore they can be handled as needed.

If you need to display a meaningful error message to the user, or log it for further diagnostics, it is best to take the OpcException or UAException instance, obtain its base exception using GetBaseException method, and retrieve its Message property. The error message obtained in this way is closest to the actual cause of the problem. OPC Data Client.NET even tries to fill in the error text in cases when the system or OPC server has not provided it.

It should be noted that for OPC Data Client "Classic" and OPC Data Client-UA operations, OpcException or UAException is the ONLY exception class that your code should be explicitly catching when calling OPC Data Client methods or getting properties. This is because you cannot safely influence or predict whether the exception will or will not be thrown. Other kinds of exception, such as those related to argument checking, should NOT be caught by typical application code, because they either indicate a programming bug in your code, or an irrecoverable system problem.

In OPC Data Client-COM, the actual error handling concepts (and related terminology) depend strongly on the programming language and development tool you are using, for example:

- In C++, if you are using the raw interfaces provided by the type library, each function call will return an HRESULT value that you will test using macros such as SUCCEEDED() or FAILED().
- In C++, if you are using "Compiler COM Support" (the #import directive), errors will be converted to exceptions of _com_error type.
- In VBScript, failed function calls will either generate run-time error (with On Error Goto 0), or fill in the Err object with information about the error (with On Error Resume Next).

In COM, OPC Data Client single-operation methods return their success/failure indication using the standard HRESULT and possibly IErrorInfo approaches. This allows you to obtain the error code, and sometimes additional pieces of information such as Description, and Source. The actual type of exception thrown (as discussed above) is hidden using this mechanism. If you need the exception object, use an alternative OPC Data Client method that performs multiple operations at once. Such methods return the success/failure information through the Exception property in the OperationResult (or derived) objects they return, allowing you to obtain the full extent of information associated with the error.

5.1.7.1 Errors and Multiple-Element Operations

Some methods on the main EasyDAClient or EasyUAClient object operate on multiple elements (such as OPC items) at once, and they also return results individually for each of the input elements. Such methods cannot simply throw an exception when there is a problem with processing one of the elements, because throwing an exception would make it impossible to further process other elements that are not causing errors. In addition, exception handling is very slow, and we need some efficient mechanism for dealing with situations when there may be multiple errors occurring at once.

For this reason, methods that work on multiple elements return an array of results, and each result may have an Exception associated with it. If this exception is a null reference, then there has been no error processing the operation on the element, and other data contained in the result object is valid. When the exception is not a null reference, it contains the actual error.

For multiple-element operations, the element-level exceptions are not wrapped in OpcException or UAException, because there is no need for you to distinguish them from other exception types in the catch statement. If there is an exception inside a multiple-level operation, it is always an exception related to OPC operations. The Exception property of the result object in a multiple-element operation therefore contains what would be the InnerException of the OpcException or UAException thrown for a single-element operation.

Exceptions that are not related to OPC operations, such as argument-checking exceptions, are still thrown directly from the multiple-element operations, i.e. they are not returned in the OperationResult object.

5.1.7.2 Errors in Subscriptions

Similarly as with multiple-element operations (above), errors in subscriptions are reported to your code by means of an Exception property in the event arguments passed to your event handler or callback method. If this exception is a null reference, then there has been no error related to the event notification, and other data contained in the event arguments object is valid. When the exception is not a null reference, it contains the actual error.

In event notifications, the exceptions are not wrapped in OpcException or UAException, because there is no need for you to distinguish them from other exception types in the catch statement. If there is an exception related to the event notification, it is always an exception related to OPC operations.

5.1.7.3 OPC Classic Errors

OPC Classic is based on Microsoft COM/DCOM, and as such, the vast majority of exceptions you encounter is of type COMException (or derived from it). The HResult and ErrorId properties of the COMException contain a numeric identification of the problem, which you can use to distinguish between the specific error causes. These exceptions can be

- Errors generated by the target OPC Classic server. It can be a standard error code defined in the OPC specifications, or a custom error code defined by the server vendor.
- Errors from the COM/DCOM infrastructure, either on the computer where the OPC Classic server runs, or on the client computer.
- Errors detected on the client side and generated by OPC Data Client.

In This Topic

A comprehensive list of possible error code cannot be given. Your program must always be ready to handle any error code that it does not explicitly know about. In most cases, you will be treating all errors the same way. In some cases, you

may need to have a special handling for one or a few specific errors.

Some Common Errors

Note: The error texts are for illustration only. They are subject to change without notice. Your program should never rely on the error texts; use the ErrorId, or specific identification properties of various exception types, to distinguish between specific errors.

The addressed OPC Classic Server is not installed

In this case, you will typically receive a COMException "Invalid class string" (localized) with HResult of -2147221005 (0x800401F3), symbolically CO_E_CLASSSTRING. Of course, you will get the same exception if the intended OPC server is actually installed, but you have misspelled its ProgId.

The remote computer cannot be reached

In this case, you will typically receive a COMException "The RPC server is unavailable" (localized) with HResult of -2147023174 (0x800706BA). Of course, you will get the same exception if the intended computer is actually running and accessible, but you have misspelled its name or IP address.

The item you are trying to access does not exist

In this case, you will typically receive a COMException "The item is no longer available in the server address space." (possibly localized) with HResult of -1073479673 (0xC0040007), symbolically OPC_E_UNKNOWNITEMID. Of course, you will get the same exception if the intended item actually exists, but you have misspelled its Item ID. This error is returned by the OPC server if the syntax of the Item ID conforms to the server's requirements, but the item does not exist. If even the syntax of the Item ID is incorrect, you will receive "The item ID does not conform to the server's syntax." (possibly localized) instead, with HResult of -1073479672 (0xC0040008), symbolically OPC_E_INVALIDITEMID.

Reading an item that is not readable, or writing an item that is not writeable

In this case, you will typically receive one of the following COMException-s:

- "Access rights of the topic do not allow reading." with HResult -1073430522 (0xC004C006).
- "Access rights of the topic do not allow writing." with HResult -1073430521 (0xC004C007).

The above exceptions come from the client side, where OPC Data Client is pre-checking the access rights of the item before it actually attempts to perform the operation. If these checks succeed, but the operation is nevertheless rejected by the server, you will typically receive

- "The item's access rights do not allow the operation." (possibly localized) with HResult of -1073479674 (0xC0040006L), symbolically OPC_E_BADRIGHTS.

Bad or Uncertain Quality of the Data

⚠ Except in methods that return the value alone (and not the DAVtq object), bad or uncertain quality is considered a normal operational situation, and does not throw or return an exception. You are responsible for testing the quality (either directly using the Quality Property, or indirectly using the HasValue property) in your code before accessing the Value Property of the DAVtq Class, because in most cases when the quality is bad, the Value is not defined (and is actually a null reference). Uncertain quality usually has some value carried with the data, although its usability is

> questionable. For more information, see **Always test the HasValue property before accessing DAVtq.Value or UAAttributeData.Value (Section 10.5)**.

The methods that do not have data quality in their results, and therefore must return an error instead of the value when the value is not present, include:

- ReadMultipleItemValues
- ReadItemValue

5.1.7.4 OPC UA Errors

Exceptions returned from OPC UA operations of OPC Data Client can be

- Errors generated by the target OPC UA server. It can be a standard error (service result or status code) code defined in the OPC specifications, or a custom error defined by the server vendor.
- Errors from the communication infrastructure, either on the host where the OPC UA server runs, or on the client host, or the components in between.
- Errors detected on the client side and generated by OPC Data Client.

In This Topic

A comprehensive list of possible errors cannot be given. Your program must always be ready to handle any error that it does not explicitly know about. In most cases, you will be treating all errors the same way. In some cases, you may need to have a special handling for one or a few specific errors.

Service Results

OPC UA services indicate the outcome of the operation using *service results*. In OPC Data Client, when the OPC UA server returns a service result that denotes an error, you will receive a UAServiceException. This object contains, among other information, a ServiceResult Property. You can use this property to distinguish between the specific errors. The value it contains is actually of the UAServiceResult type, and is further structured into various fields. In most cases, however, you are only interested in the so-called code bits (an integer) of the status code associated with the service result. The various code bits defined in the OPC UA specifications are available to you as predefined constants in the UACodeBits Class. For example, there is BadNodeIdUnknown Field, BadNotReadable Field, BadNotWritable Field etc. The UAServiceResult Class has an implicit conversion operator to an integer, which returns the code bits of the service result. This means that in order to test the UAServiceException for a specific error, you can simply take its ServiceResult Property, and compare it against the desired constant from the UACodeBits Class.

Some Common Errors

Note: The error texts are for illustration only. They are subject to change without notice. Your program should never rely on the error texts; use the exception type, and possibly other properties of the specific exception types, to distinguish between the errors.

The remote host cannot be reached

The errors in this case vary a lot, depending on factors such as the communication protocol used, or the actual configuration of the target host (when accessible, whether it drops or returns incoming packates that it does not process, etc.). You may receive e.g.:

- A UAServiceException with service result "BadTcpInternalError" (when "opc.tcp" is used).
- A UAServiceException with service result "BadCommunicationError" (when HTTPS is used).
- An EndpointNotFoundException (when HTTP is used).
- A SocketException, when "opc.tcp" is used and the host name is unknown (not found in DNS etc.).

Of course, the true cause of the exception may be that the intended host is actually running and accessible, but you have misspelled its name or IP address.

The addressed OPC UA Server is not installed or not running

The symptoms are in many cases similar as when the remote host cannot be reached.

The node you are trying to access does not exist

In this case, you will typically receive a UAServiceException with service result "BadNodeIdUnknown".

Of course, you will get the same exception if the intended node actually exists, but you have made a mistake in its Node ID.

Reading a node that is not readable, or writing a node that is not writeable

In this case, you will typically receive a UAServiceException with service result "BadNotReadable" or "BadNotWritable".

Bad or Uncertain Status of the Data

> ⚠ Except in methods that return the value alone (and not the UAAttributeData object), bad or uncertain status is considered a normal operational situation, and does not throw or return an exception. You are responsible for testing the status (either directly using the StatusCode Property, or indirectly using the HasValue Property) in your code before accessing the Value Property of the UAAttributeData Class, because in most cases when the status is bad, the Value is not defined (and is actually a null reference). Uncertain status usually has some value carried with the data, although its usability is questionable. For more information, see **Always test the HasValue property before accessing DAVtq.Value or UAAttributeData.Value (Section 10.5)**.

The methods that do not have data quality in their results, and therefore must return an error instead of the value when the value is not present, include:

- ReadMultipleValues
- ReadValue

5.1.8 Callback Queuing

All callbacks (including event notifications) originating from any of the EasyXXClient or EasyXXSubscriber objects are (by default) sent in a queued manner by a dedicated internal thread, improving resiliency against errors in custom code, and decreasing the possibility of deadlocks. When there is a sudden peak in the rate of incoming events, they can accumulate temporarily in the queue, allowing the processing to proceed smoothly.

This callback queuing can be turned on or off by the QueueCallbacks property. Turning off the callback queuing is only advisable in very special circumstances. When the callback queuing is turned off, the callback and event handling is subject to much stricter rules with regard to performance, but also with regard to what the handlers are allowed to do. Most importantly, the handlers must not call any methods on the originating component, because doing so can cause

deadlocks. This rule also applies to calls that may be inflicted by the SynchronizationContext on the component. For example, the common synchronization contexts based on Windows messages (for Windows Forms and WPF) can call other "posted" methods before the action that are instructed to do, causing unpredictable behavior and a possible deadlock too.

> ⚠ For the reasons described above, do not turn the callback queuing off unless you know very well what you are doing.

The capacity of the callback queue is given by the CallbackQueueCapacity property. The memory is not preallocated for the full maximum size, however, and therefore under normal circumstances, there is no penalty for using quite high queue capacities. The callback queue capacity should be large enough to be able to hold all incoming callbacks in case of a sudden "burst", before they are all processed (consumed) by your code.

> ⚠ For errors coming from subscriptions, OPC Data Client uses the same "channel" as for success notifications. For example, in case of the EasyUAClient Class, this means that the errors come through the DataChangeNotification Event or EventNotification Event, and the component treats each monitored item separately. Consequently, in case of an error that affects the whole connection to a particular server (or more servers), callbacks (and/or events) will be generated for every monitored item affected. This is by design. If you are subscribed to larger number of monitored items, this itself can create a "burst" that is way above the normal rate of events. Your code, and the size of the callback queue, need to be prepared for this scenario as well.

If the amount of notifications that needs to be held in the queue exceeds its capacity, a *callback queue overflow* occurs, and these notifications are lost. In addition, a LogEntry Event is generated on the EasyXXClient or EasyXXSubscriber, with corresponding message. If a notification is removed from a queue while it has overflown, the overflow status is cleared, and a different LogEntry Event is generated on the component ("*overflow clearing*"). The event contains *overflow count*, which is effectively the number of notifications that have been lost due to the queue overflow.

The idle time before the internal thread executing the queued callbacks is stopped is controlled by the CallbackQueueIdleTimeToSleep property. The thread is stopped as an optimization measure, to reduce the amount of system resources consumed. When new callbacks come, the thread is automatically started.

5.1.9 Operation Monitoring and Control

The EasyUAClient object provides a ServerConditionChanged event. This event is raised for a change in the condition of an OPC server. Specifically, you receive a notification whenever the connection to the server is established, or whenever the connection is terminated. Whenever there an error (exception) associated with the server condition, it is also available in the event arguments.

The event notification passes to you an EasyUAServerConditionChangedEventArgs object, which contains data about the new server condition. The EndpointDescriptor property identifies the server the notification applies to.
The Boolean Connected property indicates whether the server is now connected. The Exception property contains a null reference in case of no error, or it contains an exception object in case of problems.

Additional properties of the event arguments distinguish the change further. For example, the ConnectionState property contains an enumerated value (ConnectionState Enumeration) with members Disconnected, Connecting, Connected, and Disconnecting. The RetrialDelay property (valid in the Disconnected state) indicates the time (in milliseconds) to the next reconnection attempt.

A corresponding event is not currently available for OPC Classic.

Note that the information provided by this event should be used for oversight, informational and diagnostics purposes only. Specifically, you should not use it to attempt to implement any error recovery mechanism, because error recovery is already built into the OPC Data Client software, and your own mechanism would inevitably cause conflicts.

5.2 Live Mapping Model

Live Mapping works with OPC Data Access ("Classic"), OPC XML-DA and OPC Unified Architecture servers (for data access). Its features include:

- You can map to any OPC item, node attribute or property.
- Different mapping kinds such as Value, Quality, Timestamps, ErrorMessage, and many more.
- Whole class can be mapped at once, or details can be specified on mapped members individually.
- Operations on mapped types can be narrowly targeted as needed using mapping tags.
- Relative OPC browse paths can be used to avoid necessity of mapping to absolute item or node IDs.
- Meta-members allow to relate the mapped object to its location in the address space, and parameters for operations.
- Pre-made node classes for easier mapping.
- Type-less mapping is also possible, for direct correspondence between the .NET members and OPC data.

The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

This functionality is not yet available for OPC UA PubSub.

The live mapping development model allows you to write objects that correspond logically to a functionality provided by whatever is “behind” the OPC data. For example, if part of your application works with a boiler, it is natural for you to create a boiler object in the code. You then describe how your objects and their members correspond to OPC data – this is done using attributes to annotate your objects. Using Live Mapping methods, you then map your objects to OPC data, and perform OPC operations on them.

For example, when you subscribe to OPC items, incoming OPC item changes can directly set corresponding properties in your objects, without any further coding. You can then focus on writing the code that handles these property changes, abstracting away the fact how they come in.

To give a concrete example, here is a piece of code of a .NET object annotated for live mapping with OPC Data Access, in C#:

Live Mapping Annotations

```
[DAType]
class BoilerInputPipe
{
    [DANode]
    public FlowTransmitter FlowTransmitter1 = new FlowTransmitter();

    [DANode]
    public Valve Valve = new Valve();

    [DANode, DAItem]
    public bool InAlarm { get; set; };
}
```

You can use the Liver Mapping model with OPC Data Access, or with OPC Unified Architecture data.

5.2.1 Live Mapping Model for OPC Data (Classic and UA)

The live mapping development model allows you to write objects that correspond logically to a functionality provided by whatever is "behind" the OPC data. For example, if part of your application works with a boiler, it is natural for you to create a boiler object in the code. You then describe how your objects and their members correspond to OPC data – this is done using attributes to annotate your objects. Using Live Mapping methods, you then map your objects to OPC data, and perform OPC operations on them.

For example, when you subscribe to OPC items (in OPC Classic) or OPC attributes (in OPC-UA), incoming OPC notifications about changes can directly set corresponding properties in your objects, without any further coding. You can then focus on writing the code that handles these property changes, abstracting away the fact how they come in.

Note: Some mapping technologies for.NET already exist, from Microsoft or otherwise. These technologies are mainly focused on mapping database information (ORM – Object-relational mapping), and they are not well suitable for automation purposes like OPC. We therefore had to implement a new mapping model. We use the term "live mapping" so that it is clearly distinguished from other mapping technologies used; it also reflects the fact that this kind of mapping is well-suited for real-time, "live" data.

Do not confuse the live mapping model with live binding, also provided by OPC Data Client. The main differences between the two are that

- live mapping is primarily non-visual (both in design and in runtime), and
- live mapping has the ability to work with object types, not just individual members.

With live mapping model, you generally need to do following steps in order:

1. Write classes that are a logical model of your application. If you are following object-oriented principles in your design, chances are that your code already has such classes anyway.
2. Annotate the classes with attributes for live mapping. The main purpose of these attributes is to describe how the members of your classes correspond to nodes in the OPC address space.
3. Create instance(s) of your classes, as usually.
4. Perform the actual mapping, which associates your objects with OPC data.
5. Call mapping operations as needed, e.g. execute OPC reads, writes, or subscribe to changes in OPC values.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

5.2.1.1 Live Mapping Example

It is best to start with Live Mapping by going through a simple example.

In This Topic

Define the mapping

Our example works with a hypothetical boiler, which is an object composed from other structured objects, such as controllers, a drum, level indicator, and pipes. Some of these objects are composed as well. The precise structure is described in the code comments below.

Note: The code snippets demonstrated here are for OPC Classic. In OPC-UA, the principles of Live Mapping are the same, but the actual attributes and pieces of code are slightly different. If you want to use Live Mapping with OPC-UA, please use the explanation here as a guidance, and refer to the UAConsoleLiveMapping example in the product instead.

The simulated boiler data are provided by the sample OPC server that ships with the OPC Data Client product.

The code below is taken from the ConsoleLiveMapping example that ships with the product (file **Boiler.cs**):

Define Live Mapping

```
using System;
using OpcLabs.BaseLib.LiveMapping;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;
using OpcLabs.EasyOpc.DataAccess.LiveMapping.Extensions;

namespace ConsoleLiveMapping
{
    // The Boiler and its constituents are described in our application
    // domain terms, the way we want to work with them.
    // Attributes are used to describe the correspondence between our
    // types and members, and OPC nodes.

    // This is how the boiler looks in OPC address space:
    //  - Boiler #1
    //      - CC1001                    (CustomController)
    //          - ControlOut
    //          - Description
    //          - Input1
    //          - Input2
    //          - Input3
    //      - Drum1001                  (BoilerDrum)
    //          - LIX001                (LevelIndicator)
    //              - Output
    //      - FC1001                    (FlowController)
    //          - ControlOut
    //          - Measurement
    //          - SetPoint
    //      - LC1001                    (LevelController)
    //          - ControlOut
    //          - Measurement
    //          - SetPoint
    //      - Pipe1001                  (BoilerInputPipe)
    //          - FTX001                (FlowTransmitter)
```

```
//              - Output
//      - Pipe1002                  (BoilerOutputPipe)
//          - FTX002                (FlowTransmitter)
//              - Output

[DAType]
class Boiler
{
    // Specifying BrowsePath-s here only because we have named the
    // class members differently from OPC node names.

    [DANode(BrowsePath = "Pipe1001")]
    public BoilerInputPipe InputPipe = new BoilerInputPipe();

    [DANode(BrowsePath = "Drum1001")]
    public BoilerDrum Drum = new BoilerDrum();

    [DANode(BrowsePath = "Pipe1002")]
    public BoilerOutputPipe OutputPipe = new BoilerOutputPipe();

    [DANode(BrowsePath = "FC1001")]
    public FlowController FlowController = new FlowController();

    [DANode(BrowsePath = "LC1001")]
    public LevelController LevelController = new LevelController();

    [DANode(BrowsePath = "CC1001")]
    public CustomController CustomController = new CustomController();
}

[DAType]
class BoilerInputPipe
{
    // Specifying BrowsePath-s here only because we have named the
    // class members differently from OPC node names.

    [DANode(BrowsePath = "FTX001")]
    public FlowTransmitter FlowTransmitter1 = new FlowTransmitter();

    [DANode(BrowsePath = "ValveX001")]
    public Valve Valve = new Valve();
}

[DAType]
class BoilerDrum
{
    // Specifying BrowsePath-s here only because we have named the
    // class members differently from OPC node names.
```

```
        [DANode(BrowsePath = "LIX001")]
        public LevelIndicator LevelIndicator = new LevelIndicator();
    }

    [DAType]
    class BoilerOutputPipe
    {
        // Specifying BrowsePath-s here only because we have named the
        // class members differently from OPC node names.

        [DANode(BrowsePath = "FTX002")]
        public FlowTransmitter FlowTransmitter2 = new FlowTransmitter();
    }

    [DAType]
    class FlowController : GenericController
    {
    }

    [DAType]
    class LevelController : GenericController
    {
    }

    [DAType]
    class CustomController
    {
        [DANode, DAItem]
        public double Input1 { get; set; }

        [DANode, DAItem]
        public double Input2 { get; set; }

        [DANode, DAItem]
        public double Input3 { get; set; }

        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double ControlOut { get; set; }

        [DANode, DAItem]
        public string Description { get; set; }
    }

    [DAType]
    class FlowTransmitter : GenericSensor
    {
```

```
    }

    [DAType]
    class Valve : GenericActuator
    {
    }

    [DAType]
    class LevelIndicator : GenericSensor
    {
    }

    [DAType]
    class GenericController
    {
        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double Measurement { get; set; }

        [DANode, DAItem]
        public double SetPoint { get; set; }

        [DANode, DAItem(Operations =
            DAItemMappingOperations.ReadAndSubscribe)] // no OPC writing
        public double ControlOut { get; set; }
    }

    [DAType]
    class GenericSensor
    {
        // Meta-members are filled in by information collected during
        // mapping, and allow access to it later from your code.
        // Alternatively, you can derive your class from DAMappedNode,
        // which will bring in many meta-members automatically.
        [MetaMember("NodeDescriptor")]
        public DANodeDescriptor NodeDescriptor { get; set; }

        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double Output
        {
            get { return _output; }
            set
            {
                _output = value;
                Console.WriteLine("Sensor \"{0}\" output is now {1}.", NodeDescriptor,
value);
            }
```

```
        }

        private double _output;
    }

    [DAType]
    class GenericActuator
    {
        [DANode, DAItem]
        public double Input { get; set; }
    }
}
```

Notice the setter for the GenericSensor.Output property in the code above. It contains your application logic for "what to do when sensor's output changes" (in our example, we simply display the sensor's identification and the new value). Similarly, you can write logic that handles other OPC item changes, or – in the opposite direction – provides values to be written to the OPC items.

Perform operations

Let us now use the above mapping definitions to actually perform some meaningful tasks.

First, we create an instance of our logical "boiler" object, and using the DAClientMapper object (for mapping to OPC Data Access), we map it to OPC. The type mapping definition for the Boiler does not contain information about which OPC server to use, and where in its address space (tree of items) can the boiler items be found. This is intentional - although such information could have been specified directly on the Boiler class, it would hamper the reusability of such class. We therefore provide that information at the moment of mapping, by specifying it inside the new DAMappingContext passed to the mapper. Similarly, we specify the requested update rate.

The code pieces below are taken from the ConsoleLiveMapping example that ships with the product (file **Program.cs**).

Perform Live Mapping

```
varboiler1 = new Boiler();
varmapper = new DAClientMapper();
mapper.Map(boiler1, new DAMappingContext
    {
        ServerDescriptor = "OPCLabs.KitServer.2",   // local OPC server
        // The NodeDescriptor below determines where in the OPC address
        // space we want to map our data to.
        NodeDescriptor = new DANodeDescriptor
            { BrowsePath = "/Boilers/Boiler #1" },
        // Requested update rate (for subscriptions):
        GroupParameters = 1000,
    });
```

With the mapping already in place, we can easily do various operations on the mapped objects. For example, we can read in all data of the boiler, and the values will appear directly in the properties of our Boiler object:

Read data with Live Mapping

```
    Console.WriteLine();
    Console.WriteLine("Reading all data of the boiler...");
    mapper.Read();
    Console.WriteLine("Drum level is: {0}",
        boiler1.Drum.LevelIndicator.Output);
```

You can also modify properties of the level controller, e.g. change its setpoint by setting the SetPoint property, and then send all writeable values to level controller through OPC:

Write data with Live Mapping

```
    Console.WriteLine();
    Console.WriteLine("Writing new setpoint value...");
    boiler1.LevelController.SetPoint = 50.0;
    mapper.WriteTarget(boiler1.LevelController, /*recurse:*/false);
```

If you want to subscribe to changes occurring in the boiler, and have the properties in your object automatically updates with new values, you can do it as below. When, e.g., an output of any sensor changes, the Output property will be set on the corresponding object, and the code that (in our example) displays the new value will run.

Subscribe to data with Live Mapping

```
    Console.WriteLine();
    Console.WriteLine("Subscribing to boiler data changes...");
    mapper.Subscribe(/*active:*/true);

    Thread.Sleep(30 * 1000);

    Console.WriteLine();
    Console.WriteLine("Unsubscribing from boiler data changes...");
    mapper.Subscribe(/*active:*/false);
```

Benefits

You can already see some of the benefits of the live mapping in this simple example. The code that you write can focus on the logic of the application, and works directly with members of your .NET objects. You do not have to clutter your code with OPC-specific information such as server and item IDs. It also gets very easy to reuse the mapped types: If you had more boilers that are represented identically in the OPC address space, you can simply create more Boiler objects and map them all, without a need to change anything in the definition of the Boiler class.

5.2.1.2 Live Mapping Overview

In live mapping model, you end up creating Mappings that relate Mapping Sources with Mapping Targets. These concepts are explained further below.

The live mapping is orchestrated by the mapper object, described in a separate chapter (see The Mapper Object).

5.2.1.2.1 Mapping Sources

The mapping source is a data provider and/or consumer external to your application code.

For OPC Data Access, there are two types of mapping sources that you can use:

- The DAClientItemSource represents an OPC item.
- The DAClientPropertySource represents an OPC property.

For OPC Unified Architecture, following type of mapping source exists:

- The UAClientDataMappingSource represents an attribute of an OPC-UA node.

One mapping source can be (and often is) shared by multiple mappings. This happens e.g. when you map multiple members to the same source, using different Mapping Kinds (for example, you may map separate members to the value, timestamp, and quality information of the OPC item).

When you use the attributes to define the live mappings, mapping source objects are created and maintained together with their Mappings, so normally you do not deal with them directly.

5.2.1.2.2 Mapping Targets

The mapping target is an element in your code from which data can be obtained or that can accept data. Most typically, it is a property of your object, but it can also be a field, method, or an event.

The actual mapping target is represented by the ObjectMemberLinkingTarget class. When you use the attributes to define the live mappings, mapping target objects are created and maintained together with their Mappings, so normally you do not deal with them directly.

5.2.1.2.3 Mappings

The mapping is a relation between a mapping source and a mapping target. Besides specifying which mapping source and mapping target belong together, it also specifies how and when the data is being transferred between the two.

For example, a mapping for OPC Data Access items (DAClientItemMapping) may specify that the data is only transferred (from the source to the target) when a “Read” operation is requested (see Mapping Operations) and that the data will be in form of the value-timestamp-quality triple, i.e. a DAVtq object (see Mapping Kinds). With OPC-UA and UAClientDataMapping, the “Read” operation exists as well, and an example form of transfer may be the UAAttributeData, which contains the attribute value, status code, and timestamps. Or, the transferred data may simple be the OPC value itself.

When you annotate your objects using attributes, and then map your objects using the mapper object, the necessary mappings are created for all members of your objects that are annotated in such way. You therefore do not normally need to create mappings individually.

For OPC Data Access, there are two types of mappings that you can use:

- The DAClientItemMapping relates a DAClientItemSource to some mapping target. It refers to an OPC item, and you can read it or write it, or subscribe to it. These mappings are created with the use of the DAItem attribute (see further below).
- The DAClientPropertyMapping relates a DAClientPropertySource to some mapping target. It refers to an OPC property, and you can get its value. These mappings are created with the use of the DAProperty attribute (see

further below).

For OPC Unified Architecture, following type of mapping exists:

- The UAClientDataMapping relates a UAClientDataMappingSource to some mapping target. It refers to an attribute of an OPC-UA node , and you can read it or write it, or subscribe to it. These mappings are created with the use of the UAData attribute (see further below).

5.2.1.3 Attributes for Live Mapping

By annotating your types and members by attributes, you provide information to the live mapping system about the correspondences between the source OPC system and the target .NET objects.

For example, using the combination of DANode and DAItem attributes on a property of your object, you denote that the property corresponds to a node in OPC-DA address space, specify the name of that node (if different from property name), and mark the node as OPC item, i.e. capable of reading, writing or subscriptions. Using other attributes, you can also specify parameters for these mapping operations (such as the requested update rate). In OPC-UA, you would use a similar combination of UANode and UAData attributes.

Attributes in .NET are classes that, by convention, are named so that they end in the word "**Attribute**". Some languages, such as C#, allow you to omit the final **Attribute** word, when the attribute is specified. For example, in C#, you can write [DAItem] instead of [DAItemAttribute]. In this document, we will mostly use the "shortened" named of attribute classes. Remember that in some languages, you may have to use the full name of the attribute class, and therefore append the word "**Attribute**" to the attribute name listed here.

The following table shows the Live Mapping attributes common for OPC Classic and OPC-UA, and the type of elements they can be applied to.

Attribute	Purpose	on Class or Struct	on Property, Method, or Field	on Event
MappingTag	Member mapping		✔	✔
MetaMember	Member mapping		✔	

The following table shows the attributes available for OPC Classic, and the type of elements they can be applied to.

Attribute	Purpose	on Class or Struct	on Property, Method, or Field	on Event
DAItem	Member mapping		✔	✔
DAItemIdTemplate	Propagated parameter	✔		
DAMember	Member mapping		✔	
DANode	Member mapping		✔	✔
DAProperty	Member mapping		✔	
DARead	Propagated parameter	✔	✔	✔
DASubscription	Propagated	✔	✔	✔

Attribute	Purpose	on Class or Struct	on Property, Method, or Field	on Event
	parameter			
DAType	Type mapping	✔		
Server	Propagated parameter		✔	✔

The following table shows the attributes available for OPC Unified Architecture, and the type of elements they can be applied to.

Attribute	Purpose	on Class or Struct	on Property, Method, or Field	on Event
UAData	Member mapping		✔	✔
UADataChangeFilter	Propagated parameter	✔	✔	✔
UAEndpoint	Propagated parameter		✔	✔
UAMember	Member mapping		✔	
UAMonitoring	Propagated parameter	✔	✔	✔
UANamespace	Propagated parameter	✔		
UANode	Member mapping		✔	✔
UANodeIdTemplate	Propagated parameter	✔		
UARead	Propagated parameter	✔	✔	✔
UASubscription	Propagated parameter	✔	✔	✔
UAType	Type mapping	✔		

The meaning of these attributes will be explained in subsequent chapters, and can be found in the reference documentation as well.

5.2.1.4 Propagated Parameters

Attributes that are marked with “propagated parameter” in the above table do not create a new type definition or member mapping. Instead, when used, they establish or change parameter values that will be used when the actual mapping is created, using attributes for Type Mapping or Member Mapping.

The parameters set in this way also propagate further down in the mapping hierarchy. E.g. if you put a propagated

parameter attribute on a type, it will apply to that type and all its mapped members, and if any such member refer to a mapped type, the parameters will propagate recursively to it. If, however, any of these down-level types or members is annotated with the same attribute, the values from this new attribute will replace the parameters that were set higher in the hierarchy.

5.2.1.5 Type Mapping

Types involved in the live mapping for OPC Data need to be annotated with the DAType attribute (OPC Classic) or the UAType attribute (OPC-UA). If you derive your types from one of the Mapped Node Classes (such as DAMappedNode or UAMappedNode), this attribute is inherited and you do not have to explicitly specify it again.

Both reference and value types are allowed in the live mapping.

In addition to the DAType (or UAType) attribute, you can use other ("propagated parameter") attributes on your type, as seen from the table in the Attributes for Live Mapping chapter. These additional attributes do not have direct effect on the type itself, but influence all members of that type. For example, if you specify the DASubscription attribute on your type, and specify the requested update rate, this update rate will be used with all members of that type, unless overridden again by the DASubscription attribute on any member. Similarly, in OPC Unified Architecture, use can specify the sampling interval using the UAMonitoring attribute, and it will be propagated further down.

5.2.1.6 Member Mapping

The member mapping has its specifics depending whether you apply the mapping attribute to a property or field (which is most common), to a method, or to an event. In the following three chapters, we will explain the mapping behavior with these different types of members.

The subsequent chapters then explain how the actual mappings are created with different attributes, and what is their meaning and usage.

5.2.1.6.1 Mapping Property and Field Members

This is the most obvious and straightforward member mapping. The property's or field's value is the actual mapped data.

The property or field type must be compatible with the type of data, as given by the mapping (see Mapping Kinds for details on types for different kinds of mappings). Note that it is generally preferred to use properties rather than fields for the mapping purposes.

Indexed properties and fields are not currently supported. I.e. you can have an array contained in the mapped property or field, but the array has to be treated as a whole.

5.2.1.6.2 Mapping Method Members

When you map a method, it has to conform to one of the following:

- A method (function) that has no arguments, and returns a value. This method can be used to obtain values from the target. It is equivalent to a property getter.
- A method that has one argument (return value is ignored and can be void). This method can be used to modify values in the target. It is equivalent to a property setter.

The method return type (in first case), or the method's argument type (in second case) must be compatible with the type

of data, as given by the mapping (see Mapping Kinds for details on types for different kinds of mappings).

5.2.1.6.3 Mapping Event Members

An event can only serve for delivering data from the source to the target. When you map an event, the event handler has to accept two arguments. The first argument is the sender – the target object itself. The second argument contains the mapped data, and is of type TargetDataEventArgs<TData> (if the target type is not known) or TargetDataEventArgs<TData,TTarget> (for target of type TTarget).

TData is the type of data, as given by the mapping (see Mapping Kinds for details on types for different kinds of mappings).

5.2.1.6.4 Map-through, OPC Nodes and OPC Data

Following attributes are used to create or describe the actual member mappings for OPC Classic (Data Access):

- DAMember: Causes the mapping to continue into the value of the mapped member (map-through).
- DANode: Marks the member that represents a node in the OPC-DA address space (no new mapping is directly created just by using this attribute), and maps through it.
- DAItem: Gives information about OPC Data Access item, and creates a mapping for it.
- DAProperty: Gives information about OPC Data Access property, and creates a mapping for it.

Following attributes are used to create or describe the actual member mappings for data with OPC Unified Architecture (OPC-UA):

- UAMember: Causes the mapping to continue into the value of the mapped member (map-through).
- UANode: Marks the member that represents a node in the OPC-UA address space (no new mapping is directly created just by using this attribute), and maps through it.
- UAData: Gives information about OPC-UA data provided by an attribute on an OPC-UA node, and creates a mapping for it.

In order to describe what precisely happens when these attributes and their combinations are used, it is important do understand that when the mapping is done, there is a "current" OPC node maintained by the mapper object, and a "current" object being mapped. When the mapping is started, the current OPC node is given to the mapper (or, if omitted, starts at the root of the address space), and the current object is the object to be mapped, also passed to the mapper.

As the mapping progresses, the current OPC node is only changed by the DANode (or UANode) attribute. That is, if you want any of your members be mapped to a node different from the parent node, you need to annotate that member with the DANode (or UANode) attribute. The current OPC node is thus changed for that member and anything below it in the mapping hierarchy.

The DAMember and DANode (or the UAMember and UANode) attributes both function as map-through. This means that when they are encountered during mapping, the value of the annotated member is obtained, and the mapping continues into the object given by the value (if the value is a null reference, it is simply ignored). You can therefore have a structure of objects where object's properties or fields contain other objects, and the whole structure can be mapped at once.

The difference between the DAMember and DANode (or UAMember and UANode) is that with DAMember (UAMember), the current OPC node is not changed. This means that members of the child object will be mapped to the same OPC node as the parent object. With DANode (UANode), however, the members of the child object will be mapped to an OPC node different from the parent. Which node precisely is used for the child object's members is described in a subsequent chapter, Browse Path and Node Id Resolution. Here, it is enough to say that if you don't specify any argument to the DANode (UANode) attribute, the current OPC node will be changed to a node with the same name as the member you

are annotating. So, if your property is named e.g. Output, it will correspond to an **Output** node in the OPC address space.

The DAMember and DANode (or UAMember and UANode) attributes do not create any OPC data mappings that can actually have some effect on the underlying OPC system. They are just helper attributes that allow you to define the structure of mappings. In order to create OPC data mappings, you need to annotate your member with either DAItem or DAProperty attribute (in OPC Classic), or with the UAData attribute (in OPC-UA).

With the DAItem attribute, you specify that the annotated member should be mapped to an OPC Data Access item, and you optionally provide additional information about how the mapping should be done. The arguments that you can use with the DAItem attribute allow you to optionally specify:

- AccessPath: An optional data path suggestion to the server.
- RequestedDataType: The data type requested (VarTypes.Empty for the server's canonical data type).
- Kind: Specifies how the item will be mapped (see Mapping Kinds).
- Operations: The item mapping operations in which the mapping will participate (see Mapping Operations).
- ItemType: The type of OPC item.

Note that with the DAItem attribute, you do not specify information about which OPC node (item) should be mapped: This information comes from the usage of DANode attribute(s).

With the DAProperty attribute, you specify that the annotated member should be mapped to an OPC Data Access property (on an item), specify the numerical ID of the property, and optionally additional information about how the mapping should be done.

With the UAData attribute, you specify that the annotated member should be mapped to an OPC-UA attribute, and you optionally provide additional information about how the mapping should be done. The arguments that you can use with the UAData attribute allow you to optionally specify:

- AttributeId: Identifies an attribute of a node.
- ValueType: The type of the OPC-UA Value attribute.
- IndexRangeList: List of index ranges for individual array dimensions.
- Kind: Specifies how the item will be mapped (see 7.1.9Mapping Kinds).
- Operations: The item mapping operations in which the mapping will participate (see 7.1.8Mapping Operations).

Note that with the UAData attribute, you do not specify information about which OPC-UA node should be mapped: This information comes from the usage of UANode attribute(s).

Typically, when the structure of your objects follows the structure of the OPC address space, you will annotate object members as follows:

- Members that correspond to OPC branches (folders): In OPC Classic, use the DANode attribute. In OPC-UA, use the UANode attribute.
- Members that correspond to OPC leaves (items, variables): In OPC Classic, use the DANode and DAItem attributes together. In OPC-UA, use the UANode and UAData attributes together.

It is, however, perfectly possibly (and indeed, practical) that your objects may have a structure that differs from a structure of the OPC address space – either just at some places, or globally. In such case, you can still make your objects "fit" the address space, by proper placing of all attributes described here.

5.2.1.6.5 Browse Path and Node Id Resolution

Items in OPC Data Access are ultimately identified by their item IDs (node IDs), which are strings defined by the server. Their syntax is not standardized, and although they commonly look like a sequence of names or numbers with some separators interspersed (such as "Boiler.Drum1001.LIX001.Output"), there is no general way to construct the item IDs

without browsing the OPC server's address space.

In OPC-UA, nodes have their Node Ids, which are a bit more complex (they are qualified by a namespace, and can be of several types, not just strings) – but effectively, they behave similarly. Their values are not generally standardized; they are completely determined by the concrete OPC-UA server you are connecting to.

When mapping, you could specify the precise node ID with the DANode (or UANode) attribute. This may work in some scenarios, but has severe limitations. The OPC address space (and your object structure) typically contains repeated patterns of nodes that are just placed differently. You want to define and map object types that match these patterns, but because the OPC node IDs will differ based on the placement of the pattern in the OPC address space, you won't be able to tell a unique item ID for each member.

This problem can be solved by using browse paths – sequences of item names that start from a known place in the OPC address space (e.g. in its root). The names in browse paths are "short" names that are used when browsing and sometimes displayed to the user, and the "short" node name typically remains the same if the node with the same meaning (a part of node pattern) is placed differently in the OPC address space. The mapping can translate the browse paths to item IDs when they are actually needed. For more on browse paths, see 4.11.7Browse Paths.

In order to specify nodes using a browse path, use the BrowsePath argument of the DANode (or UANode) attribute. It is a string that can contain an absolute or relative browse path. An absolute browse path string has similar limitations as the item ID; you may find it useful to define the starting point for the mapping (always needed in OPC-UA), or if the mapping does not start at the root of the OPC address space (in OPC Classic), though.

With a relative browse path string, you can specify that the member annotated with DANode (or UANode) attribute will belong to an OPC node that is based on the current OPC node, but appended with the relative browse path given. The relative browse path string can contain a single "short" item name, or multiple separated names forming a longer path.

For example, if the current OPC node (for the parent) in OPC Classic is given by browse path '/Boiler/Drum1001/LIX001', and your member is annotated with DANode attribute with BrowsePath = "Output", the OPC node for the member will be given by a browse path '/Boiler/Drum1001/LIX001/Output', which can then be translated to a full item ID. When the same type containing your member is used within a different context (at different place in the OPC address space), the resulting browse path will be different, but still contain "Output" node at the end. This is how repeated node patterns can be mapped without having to specify the individual item IDs.

By default, when the DANode (or UANode) attribute is used without further arguments, the name of the member being annotated is used as the relative browse path string for the DANode (or UANode) attribute. This means that if you name your members the same as the nodes in the OPC address space, you do not have to provide any additional arguments to the DANode (or UANode) attribute. This is a common practice, but if some members cannot follow it, you can still specify a different browse path just for them.

In some cases, the use of browse paths is not possible or practical, but you can define a "rule" specifying how the node IDs are constructed for the concrete OPC server and concrete part of its address space. In such case, you can use the DAItemIdTemplate (or UANodeIdTemplate) attribute to annotate your type, giving it an ItemIdTemplateString (or a NodeIdTemplateString, in OPC-UA) argument that specifies how a node ID of a node is made of the parent node Id, and browse name of the node being mapped (a member name, typically).

Following macros can be used in template strings for DAItemIdTemplate attribute:

- **$(ParentItemID)**: The OPC Item Id of the parent node.
- **$(ParentNodePath)**: The OPC node path of the parent node (only in OPC XML-DA).
- **$(BrowseName)**: The browse name of current node.

Following macros can be used in template strings for UANodeIdTemplate attribute:

- **$(ParentNodeID)**: The node Id of the parent OPC-UA node.
- **$(BrowseName)**: The browse name of current node.

Final note: In OPC Classic, it is possible to specify combinations of BrowsePath and ItemId arguments of the DANode attribute, and further combine it with the DAItemIdTemplate attribute. In OPC-UA, it is possible to specify combinations of BrowsePath and NodeId arguments of the UANode attribute, and further combine it with the UANodeIdTemplate attribute.

In such cases, the information specified at the "deepest" level prevails. If both the browse path and node ID are specified for a node, then both are passed to the underlying OPC client object for processing. Which one of them will be finally used depends on many factors, including the concrete capabilities of the server and the version(s) of OPC specifications it conforms to, so in such case, you are responsible for guaranteeing that both the browse path and the item ID represent the same node.

5.2.1.6.6 Meta- Members

Sometimes you may have a need to obtain additional information that comes from the mapping process; such as what were the resulting values of propagated parameters, or what is the precise node the current object is being mapped to (this is useful when the object is used repeatedly for patterns of nodes in the OPC address space). In order to have this information stored into members of your objects, you can use *meta-members*.

Annotating a member (field, method or property) with the MetaMember attribute causes it to be set to named information gathered during mapping (a meta-information).

The MetaMember attribute has a Name argument that denotes which meta-information will be used.

Meta-members in the following table are available for both OPC Classic and OPC-UA:

Name	Type	Description
Mappings	IEnumerable<AbstractMapping>	An enumeration of mappings defined directly on the target object.
Parent	Object	A reference to the parent target object; **null** if none.

In addition, meta-members in the following table are available for OPC Classic:

Name	Type	Description
GroupParameters	DAGroupParameters	An object containing subscription parameters, such as the requested update rate.
NodeDescriptor	DANodeDescriptor	The descriptor of the OPC node involved in the operation.
ReadParameters	DAReadParameters	The read parameters (such as data source or value age).
ServerDescriptor	ServerDescriptor	An OPC server involved in the operation.

In addition, meta-members in the following table are available for OPC-UA:

Name	Type	Description
EndpointDescriptor	UAEndpointDescriptor	An OPC-UA server involved in the operation.
MonitoringParameters	UAMonitoringParameters	An object containing monitoring parameters, such as the sampling interval.
NodeDescriptor	UANodeDescriptor	The descriptor of the OPC-UA node involved in the operation.
ReadParameters	UAReadParameters	The read parameters (such as the maximum age of the value).
SubscriptionParameters	UASubscriptionParameters	An object containing subscription parameters, such as the publishing interval.

All meta-information is constant after being established, and changing it later has no effect on the underlying mappings.

5.2.1.6.7 Mapping Tags

Mapped members can be tagged with one or more tags. When performing operations, the developer can specify that only members with certain tag, or combination of tags, will be affected.

A mapping tag is any non-empty string.

In order to place a mapping tag on any mapped member, use the MappingTag attribute with it. You can repeat the MappingTag attribute on the member, if you wish the member be tagged with multiple tags.

If you are creating library objects that can be shared with other projects or developers, you are responsible for making sure that the same tag is not accidentally used for different purposes. It is therefore recommended to design a scheme that prevents such conflicts. For example, you may use a namespace approach, and have tags such as "MyCompany.MyProduct.MyComponent.MyTag".

5.2.1.6.8 Deferred Mapping

With deferred mapping, the mapped node object does not have to exist at the time of mapping, but is determined later, when it is actually needed. This is useful e.g. if it is costly to create the full object graph upfront, either because it is very large, or because it depends on dynamic factors that are not known at the time of mapping.

In order to use deferred mapping, you need to:

1. Specify Deferred = true with the DANode or UANode attribute. This causes the mapped member be set to a function ("deferred mapping function") that returns the mapped object (normally, the mapped member contains the mapped object itself).
2. Set the DeferredMapNodeFunction of the DAClientMapper or UAClientMapper to a function that returns the deferred mapping function (see above) for a given mapping context.

The deferred mapping is used in OPC-UA Modelling to dynamically provide node objects whose types depend on the information model of the OPC-UA server.

5.2.1.7 Mapped Node Classes

The DAMappedNode class is a pre-made base class for mapping OPC Data Access nodes. The UAMappedNode class is a pre-made base class for mapping OPC-UA data nodes.

You may (but do not have to) use this class as a base class for your objects that you map to OPC Data Access (or OPC-UA data) nodes. The class is already annotated with the DAType (or UAType) attribute, and contains properties that are mapped to useful meta-members, such as the NodeDescriptor property.

Sometimes it is not possible to use these mapped node classes, usually because you already have a class hierarchy where your objects are derived form a different base that you cannot change. In other cases, however, the DAMappedNode (or UAMappedNode) class serves as a handy shortcut that can make your code shorter.

5.2.1.8 Mapping Operations

There are several things you can do with OPC Data: you can read from them, write to them, or subscribe to their changes. Not all mappings should be involved in all operations, and you therefore need to specify the operations that are of interest to you.

For DAClientItemMapping (OPC Data Access items), doing this means properly setting the Operations argument of the DAItem attribute. You can set it to one of the following values, or their combination:

- DAItemMappingOperations.Read: A Read operation.
- DAItemMappingOperations.Write: A Write operation.
- DAItemMappingOperations.Subscribe: A Subscribe operation.

If you want a combination, you can use a logical OR of the individual values, or use a predefined symbolic name such as ReadAndWrite, ReadAndSubscribe, etc. By default, the Operations argument is set to DAItemMappingOperations.All, meaning that the mapping will be involved in all operations.

There is just one mapping operation for DAClientPropertyMapping (OPC Data Access properties) – the "Get" operation that obtains the value of the OPC property. For OPC-DA property mappings, it is therefore neither necessary nor possible to specify the operation(s) that certain mapping should be involved in.

For UAClientDataMapping (in OPC-UA), you specify the operations by properly setting the Operations argument of the UAData attribute. You can set it to one of the following values, or their combination:

- UADataMappingOperations.Read: A Read operation.
- UADataMappingOperations.Write: A Write operation.
- UADataMappingOperations.Subscribe: A Subscribe operation.

The operations do not get actually executed until you write and run a code that does it. See Invoking the Operations for details.

5.2.1.9 Mapping Kinds

When you describe that certain mapping target (e.g. a property) should be mapped to certain mapping source (e.g. an OPC item), there is still a piece of information missing: What exactly is the content of that mapping target? For example, with OPC-DA item, does the mapped element contain the value of the OPC item itself, or the quality, or does it contain the value/timestamp/quality combination, or even something else?

In order to distinguish between these, the live mapping has a concept of a mapping kind. You can choose which kind of

mapping to use for certain mapping target. You can also map the same source to multiple targets, each time with a different mapping kind. Using this approach, you can e.g. map the value, timestamp, and quality, each to a separate property.

The mapping kind is specified using the Kind property of the DAItem attribute (for OPC-DA item mappings) on the target member or on the UAData attribute (OPC OPC-UA data mappings).

The following table lists all possible mapping for OPC-DA item mappings. When no mapping kind is specified, the default mapping kind of Value is used.

Kind	Type	Description
Result	OperationResult or a derived type	Map the operation result.
ErrorId	Int32	Map the error ID. Always empty with success.
Exception	Exception or a derived type	Map the exception. null if the operation has been successful.
ErrorMessage	String	Map the error message. An empty string if the operation has been successful.
StatusInfo	StatusInfo	Map the status information.
Vtq	DAVtq or DAVtq<*T*>	Map the item value/timestamp/quality combination. See Note 1.
Value	Object or *T*	Map the item value. This is the default mapping kind. See Note 1.
Timestamp	DateTime	Map the timestamp. In UTC. See Note 1.
TimestampLocal	DateTime	Map the timestamp. In local time. See Note 1.
Quality	DAQuality	Map the OPC quality. See Note 1.

For OPC-DA property mappings, the mapping kind is specified using the Kind property of the DAProperty attribute on the target member. The following table lists all possible mapping for OPC-DA property mappings. When no mapping kind is specified, the default mapping kind of Value is used.

Kind	Type	Description
Result	OperationResult or a derived type	Map the operation result.
ErrorId	Int32	Map the error ID. Always empty with success.
Exception	Exception or a derived type	Map the exception. null if the operation has been successful.
ErrorMessage	String	Map the error message. An empty string if the operation has been successful.
StatusInfo	StatusInfo	Map the status information.
Value	Object or *T*	Map the property value. This is the default mapping kind. See Note 1.

The following table lists all possible mapping for OPC-UA data mappings. When no mapping kind is specified, the default mapping kind of Value is used.

Kind	Type	Description
Result	OperationResult or a derived type	Map the operation result.
ErrorId	Int32	Map the error ID. Always empty with success.
Exception	Exception or a derived type	Map the exception. null if the operation has been successful.
ErrorMessage	String	Map the error message. An empty string if the operation has been successful.
StatusInfo	StatusInfo	Map the status information.
AttributeData	UAAttributeData or UAAttributeData <*T*>	Map the attribute value/source timestamp/server timestamp/status code combination. See Note 1.
Value	Object or *T*	Map the item value. This is the default mapping kind. See Note 1.
ServerTimestamp	DateTime	Map the server timestamp. In UTC. See Note 1.
ServerTimestampLocal	DateTime	Map the server timestamp. In local time. See Note 1.
SourceTimestamp	DateTime	Map the source timestamp. In UTC. See Note 1.
SourceTimestampLocal	DateTime	Map the source timestamp. In local time. See Note 1.
StatusCode	UAStatusCode	Map the status code. See Note 1.

Note 1: The mapping target value is not changed if the operation resulted in an exception. This means that when an error occurs, the mapping target with this mapping kind will not be reset to null or other default value; in fact it won't be influenced at all. E.g. if an item's value (a property annotated with mapping kind of Value) is successfully obtained once, and then an error occurs in a subsequent operation, the target property will remain unchanged. You can detect the error by additional mappings with different mapping kinds (e.g. Exception).

5.2.1.10 Mapping Arguments and Phases

✔p>When a mapping operation, such as Read, Write is performed, the execution is made in certain phases. First, the arguments for the operation are obtained from the targets, the actual operation then takes places, and after that, the results are stored to the targets again. For operations such as Subscribe, when the mapping source generates an asynchronous event, similar phases exist during the processing of the notification as well.

For the purpose of clarity, we will call the individual phases as follows:

- **Input**: Before the mapping operation is executed, values are obtained from the mapping target (and then send to the mapping source).
- **Output**: After the mapping operation has been executed, values (received from the mapping source) are stored to the mapping target.
- **Notification Input**: When an asynchronous notification arrives from the mapping source, the values contained in the notification are stored to the mapping target.
- **Notification Output**: When an asynchronous notification from the mapping source is finalized, it is updated with the values obtained from the mapping target. Not currently used.

You can map some of Outputs and Notification Inputs, or none, or all of them – all these approaches are valid, because it is up to you how you handle the data that comes from the mapping source. With Inputs (and Notification Outputs), you must, however, obey the rules that exist for every mapping operation, and map the proper combination of them, so that the mapping source can receive the data it expects.

The following table shows how (depending on the item mapping kind) the Read, Write and Subscribe operations treat their mapping arguments for OPC-DA item mappings.

	Mapping Operation		
Mapping Kind	**Read**	**Write**	**Subscribe**
Result	Output	Output	Notification Input
ErrorId	Output	Output	Notification Input
Exception	Output	Output	Notification Input
ErrorMessage	Output	Output	Notification Input
StatusInfo	Output	Output	Notification Input
Vtq	Output	Input	Notification Input
Value	Output	Input	Notification Input
Timestamp	Output	Input	Notification Input
TimestampLocal	Output	Input	Notification Input
Quality	Output	Input	Notification Input

The Write operation in OPC-DA can either write just a value, or a value/timestamp/quality triple. Only following combinations of inputs are therefore valid for the OPC-DA Write operation:

Vtq	Value	Timestamp or TimestampLocal	Quality	Outcome
✔				Value/timestamp/quality triple is written.
	✔	✔	✔	
	✔			Just the value is written.

For OPC-DA property mappings, there is just one mapping operation (Get), and the corresponding table is trivial:

	Mapping Operation
Mapping Kind	**Get**
Result	Output
ErrorId	Output
Exception	Output
ErrorMessage	Output
StatusInfo	Output

Value	Output

The following table shows how (depending on the item mapping kind) the Read, Write and Subscribe operations treat their mapping arguments for OPC-UA data mappings.

	Mapping Operation		
Mapping Kind	**Read**	**Write**	**Subscribe**
Result	Output	Output	Notification Input
ErrorId	Output	Output	Notification Input
Exception	Output	Output	Notification Input
ErrorMessage	Output	Output	Notification Input
StatusInfo	Output	Output	Notification Input
AttributeData	Output	Input	Notification Input
Value	Output	Input	Notification Input
ServerTimestamp	Output	Input	Notification Input
ServerTimestampLocal	Output	Input	Notification Input
SourceTimestamp	Output	Input	Notification Input
SourceTimestampLocal	Output	Input	Notification Input
StatusCode	Output	Input	Notification Input

The Write operation in OPC-UA can either write just a value, or a value/server timestamp/source timestamp/status code quadruple. Only following combinations of inputs are therefore valid for the OPC-UA Write operation:

AttributeData	**Value**	**ServerTimestamp or ServerTimestampLocal**	**SourceTimestamp or SourceTimestampLocal**	**StatusCode**	**Outcome**
✔					Value/server timestamp/source timestamp/status code quadruple is written.
	✔	✔	✔	✔	
	✔				Just the value is written.

5.2.1.11 The Mapper Object

The mapper object is at the center of the live mapping mode. You will typically use it for two main purposes:

1. Establish the correspondences between the source OPC data, and you target .NET objects: see Mapping Your Objects.
2. Perform various operations using the mappings established earlier: Invoking the Operations.

For mapping of OPC Data Access (OPC-DA) sources, the actual mapper class is DAClientMapper. You can create an instance of DAClientMapper simply by instantiating it with one of its constructor overloads. You can also use a static DAClientMapper.SharedInstance object and avoid the need of instantiation.

For mapping of OPC Unified Architecture (OPC-UA) sources, the actual mapper class is UAClientMapper. You can create an instance of UAClientMapper simply by instantiating it with one of its constructor overloads. You can also use a static UAClientMapper.SharedInstance object and avoid the need of instantiation.

5.2.1.11.1 Mapping Your Objects

In order to create the mappings, i.e. associations between mapping sources (OPC data) and mapping targets (your object members), you typically call one of the Map method overrides on the mapper object (DAClientMapper, for OPC Data Access).

For the mapping, you can either have the mapper use a copy of its mapping context template (defined by its MappingContextTemplate property), or pass in your own Mapping Context object (see further below for details of the mapping context). In both cases, you need to specify the object to be mapped. The mapper then uses its knowledge about the object type (as given by the mapping attributes) to create and add to itself a set of mappings, one mapping for each mapped member in your object structure.

It is also possible to "undo" the mapping. To do so, you typically call one of the Unmap method overrides on the mapper object (DAClientMapper, for OPC Data Access). You can either un-map all mappings that currently exist in the mapper (if you pass no arguments), or un-map specific mappings.

5.2.1.11.2 Mapping Context

The mapping context holds information about the state of the mapping process. In OPC Data Access (OPC-DA), the mapping context is represented by an instance of DAMappingContext class. In OPC Unified Architecture (OPC-UA), the mapping context is represented by an instance of UAMappingContext class.

The state of mapping changes in the mapping context, as the mapping process proceeds from your given "starting" object, to its possible constituent objects.

For example, the mapping context contains the information about the OPC server and current OPC node, and parameters for reading and subscription. As the mapping proceeds to deeper level, the properties in the mapping context are propagated to the deeper levels as well, and they change accordingly. For example, if a member is mapped to an OPC-DA node, the NodeDescriptor property of the DAMappingContext changes to reflect the information about the node. If the member represents an OPC branch and contains another object, this new NodeDescriptor will be the basis for mapping the members of the inner object.

Refer to Propagated Parameters for descriptions of attributes that are maintained in the mapping context.

You need to make sure that enough information is provided for the mapping, either through the "starting" mapping context used with the Map method, or by means of the mapping attributes. Typically, for example, the OPC server's computer and ProgID (or endpoint URL) will not be specified through the mapping attributes, but will come from your code, so that it can be modified in run-time. In OPC-DA, you will therefore create a new DAMappingContext with properly filled ServerDescriptor, and pass it to the Map method. In OPC-UA, you will create a new UAMappingContext with properly filled EndpointDescriptor, and pass it to the Map method.

5.2.1.11.3 Invoking the Operations

Once you have your objects mapped, you can invoke operations on them. The operations available depend on the type of mapper object.

For OPC Data Access (the DAClientMapper class), following methods exist for invoking operations:

- The Get method executes the “Get”, i.e. gets the values of OPC-DA properties.
- The Read method executes the “Read”, i.e. reads the data of OPC-DA items.
- The Subscribe method executes the “Subscribe”; depending on its Boolean arguments, it either subscribes to or unsubscribes from changes of OPC-DA items.
- The Write method executes the “Write”, i.e. writes data into the OPC-DA items.

For OPC Unified Architecture (the UAClientMapper class), following methods exist for invoking operations:

- The Read method executes the “Read”, i.e. reads the data of OPC-UA node attributes.
- The Subscribe method executes the “Subscribe”; depending on its Boolean arguments, it either subscribes to or unsubscribes from changes of OPC-UA node attributes (monitored items).
- The Write method executes the “Write”, i.e. writes data into the OPC-UA node attributes.

Each operation can be invoked either on all applicable mappings currently existing in the mapper, or on a specified subset of mappings. For example, the Read method on DAClientMapper has two overloads, one with no arguments (reads all item mappings that have DAItemMappingOperations.Read included in their Operations property), and one with the IEnumerable<IDAClientMapping> argument (reads just the specified mappings).

Since you typically do not have direct access to the individual mappings created for your objects (except with Type-less Mapping), you might be asking how to get access to the subset of mappings that you need. For example, you may need to perform an OPC Read on just certain mapped object, somewhere inside the whole hierarchy.

The way to obtain the set of mapping is to query the mapper itself. The Mappings property of the mapper contains an IEnumerable<> of all mappings currently maintained by the mapper. You can filter these mappings in any way you like, testing whether they fulfill your application-defined criteria, and then perform the operation just with the filtered subset.

In order to make this task easier, some extension method exist. Following methods are available for all OPC specifications:

- AbstractMappingExtension.GetTargetObject method determines the target object of the mapping. You can use it to test whether the mapping is for “your” object.
- AbstractMappingExtension.BelongsTo method determines whether a given target object belongs to this mapping, either directly, or recursively (to any of the parent mappings).

For OPC Data Access, you can use following extension methods:

- DAClientMapperExtension.GetTarget method executes the OPC-DA "Get" operation on member of given target object, directly or recursively.
- DAClientMapperExtension.ReadTarget method executes the OPC-DA "Read" operation on member of given target object, directly or recursively.
- DAClientMapperExtension.SubscribeTarget method executes the OPC-DA "Subscribe" operation on member of given target object, directly or recursively.
- DAClientMapperExtension.WriteTarget method executes the OPC-DA "Write" operation on member of given target object, directly or recursively.

For OPC Unified Architecture, you can use the following extension methods:

- UAMapperExtension.ReadTarget method executes the OPC-UA "Read" operation on member of given target object, directly or recursively.
- UAMapperExtension.SubscribeTarget method executes the OPC-UA "Subscribe" operation on member of given target object, directly or recursively.

- UAMapperExtension.WriteTarget method executes the OPC-UA "Write" operation on member of given target object, directly or recursively.

With each of the extension methods on the DAClientMapperExtension or UAMapperExtension, you can also specify that just members with given mapping tag, or with tags fulfilling certain condition (predicate) will be affected.

The DAClientMapper and UAClientMapper objects have a Boolean QueuedExecution property (defaults to false). When set to 'true', the operations on the mapper are executed outside the calling thread, but still in the order of arrival. This allows non-blocking operation calls on the mappers.

5.2.1.11.4 Type-less Mapping

While less common, you can actually using the live mapping mechanism for establishing a direct correspondence between mapping sources and mapping targets, without having to map the types. This may be beneficial e.g. if your application just needs to pick a small number of “scattered” items to be mapped from a large OPC address space, and at the same time you won’t gain much from defining the types for objects that exist in the OPC address space.

In order to use the type-less mapping, you need to call the DefineMapping method on the mapper object, passing it the three essential pieces of information: The mapping source, the mapping itself (not yet associated with the source and target), and the mapping target. The DefineMapping method associates the mapping with its source and target, and adds the mapping to the mapper.

The example below shows how to define a new type-less mapping which maps a specified OPC-DA item to the Value member of your target MyClass object. The mapper is then instructed to invoke the OPC read operation, which will in turn store the OPC item’s value to your target object.

Type-less mapping (OPC DA) in C#

```
var mapper = new DAClientMapper();
var target = new MyClass2();

// Define a type-less mapping.

mapper.DefineMapping(
     new DAClientItemSource("OPCLabs.KitServer.2", "Simulation.Register_I4",
DADataSource.Cache),
     new DAClientItemMapping(typeof(Int32)),
     new ObjectMemberLinkingTarget(target.GetType(), target, "Value"));

// Perform a read operation.
mapper.Read();
```

Type-less mapping (OPC DA) in VB.NET

```
Public Shared Sub Main1()
    Dim mapper = New DAClientMapper()
    Dim target = New MyClass2()

    ' Define a type-less mapping.

    mapper.DefineMapping( _
```

```
            New DAClientItemSource( _
                "OPCLabs.KitServer.2",
                "Simulation.Register_I4",
                New DAReadParameters(DADataSource.Cache)),
            New DAClientItemMapping(GetType(Int32)),
            New ObjectMemberLinkingTarget(target.GetType(), target, "Value"))

    ' Perform a read operation.
    mapper.Read()

    ' Display the result.
    Console.WriteLine(target.Value)
End Sub
```

The example below shows how to define a new type-less mapping which maps a specified node in OPC Unified Architecture (OPC-UA) server to the Value member of your target MyClass2 object. The mapper is then instructed to invoke the OPC read operation, which will in turn store the OPC node's value to your target object:

Type-less mapping (OPC UA) in C#

```
UAEndpointDescriptor endpointDescriptor =
    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
// or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
// or "https://opcua.demo-this.com:51212/UA/SampleServer/"

var mapper = new UAClientMapper();
var target = new MyClass2();

// Define a type-less mapping.

MemberInfo memberInfo = target.GetType().GetMember("Value").SingleOrDefault();
Debug.Assert(memberInfo != null);

mapper.DefineMapping(
    new UAClientDataMappingSource(
        endpointDescriptor,
        "nsu=http://test.org/UA/Data/;i=10389",
        UAAttributeId.Value,
        UAIndexRangeList.Empty,
        UAReadParameters.CacheMaximumAge),
    new UAClientDataMapping(typeof(Int32)),
    new ObjectMemberLinkingTarget(target.GetType(), target, memberInfo));

// Perform a read operation.
mapper.Read();
```

Type-less mapping (OPC UA) in VB.NET

```
Public Shared Sub Main1()

    ' Define which server we will work with.
    Dim endpointDescriptor As UAEndpointDescriptor =
            "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
    ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
    ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

    Dim mapper = New UAClientMapper()
    Dim target = New MyClass2()

    ' Define a type-less mapping.

    Dim memberInfo = target.GetType().GetMember("Value").SingleOrDefault()
    Debug.Assert(memberInfo IsNot Nothing)

    mapper.DefineMapping( _
        New UAClientDataMappingSource( _
            endpointDescriptor, _
            "nsu=http://test.org/UA/Data/;i=10389", _
            UAAttributeId.Value, _
            UAIndexRangeList.Empty, _
            UAReadParameters.CacheMaximumAge), _
        New UAClientDataMapping(GetType(Int32)), _
        New ObjectMemberLinkingTarget(target.GetType(), target, memberInfo))

    ' Perform a read operation.
    mapper.Read()

    ' Display results
    Console.WriteLine(target.Value)
End Sub
```

In order to remove the mapping added in this way, call the UndefineMapping method on the mapper object.

The example below shows how to subscribe and unsubscribe:

Subscribe and subscribe with type-less mapping in C#

```
// This example for OPC DA type-less mapping shows how to define a mapping and perform
subscribe and unsubscribe operations.

using System;
```

```
using System.Threading;
using OpcLabs.BaseLib.ComponentModel.Linking;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;

namespace DocExamples.DataAccess._DAClientMapper
{
    partial class DefineMapping
    {
        class MyClassSubscribe
        {
            public Double Value
            {
                set
                {
                    // Display the incoming value
                    Console.WriteLine(value);
                }
            }
        }

        public static void Subscribe()
        {
            var mapper = new DAClientMapper();
            var target = new MyClassSubscribe();

            // Define a type-less mapping.

            mapper.DefineMapping(
                 new DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp", 1000,
DADataSource.Cache),
                 new DAClientItemMapping(typeof(Double)),
                 new ObjectMemberLinkingTarget(target.GetType(), target, "Value"));

            // Perform a subscribe operation.
            mapper.Subscribe(true);

            Thread.Sleep(30 * 1000);

            // Perform an unsubscribe operation.
            mapper.Subscribe(false);
        }
    }
}
```

Subscribe and subscribe with type-less mapping in VB.NET

```
' This example for OPC DA type-less mapping shows how to define a mapping and perform
```

```
subscribe and unsubscribe operations.

Imports OpcLabs.BaseLib.ComponentModel.Linking
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping

Namespace DocExamples._DAClientMapper
    Partial Friend Class DefineMapping
        Class MyClassSubscribe
            Public WriteOnly Property Value As Double
                Set(value As Double)
                    ' Display the incoming value
                    Console.WriteLine(value)
                End Set
            End Property
        End Class

        Public Shared Sub Subscribe()
            Dim mapper = New DAClientMapper()
            Dim target = New MyClassSubscribe()

            ' Define a type-less mapping.

            mapper.DefineMapping(
                 New DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp", 1000,
DADataSource.Cache),
                 New DAClientItemMapping(GetType(Double)),
                 New ObjectMemberLinkingTarget(target.GetType(), target, "Value"))

            ' Perform a subscribe operation.
            mapper.Subscribe(True)

            Threading.Thread.Sleep(30 * 1000)

            ' Perform an unsubscribe operation.
            mapper.Subscribe(False)
        End Sub
    End Class
End Namespace
```

The example below shows how to make various kinds of mappings:

Various mapping kinds with type-less mapping in C#

```
// This example for OPC DA type-less mapping shows how to define mappings of various
kinds, and perform subscribe and
```

```
// unsubscribe operations.

using System;
using System.Threading;
using OpcLabs.BaseLib.ComponentModel.Linking;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Generic;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;

namespace DocExamples.DataAccess._DAClientMapper
{
    partial class DefineMapping
    {
        class MyClassMappingKinds
        {
            public Double CurrentValue
            {
                set
                {
                    // Display the incoming value
                    Console.WriteLine("Value: {0}", value);
                }
            }

            public DAVtq<Double> CurrentVtq
            {
                set
                {
                    // Display the incoming Vtq
                    Console.WriteLine("Vtq: {0}", value);
                }
            }

            public Exception CurrentException
            {
                set
                {
                    // Display the incoming exception
                    Console.WriteLine("Exception: {0}", value);
                }
            }

            public DAVtqResult<Double> CurrentResult
            {
                set
                {
                    // Display the incoming result
                    Console.WriteLine("Result: {0}", value);
```

```
                }
            }
        }

        public static void MappingKinds()
        {
            var mapper = new DAClientMapper();
            var target = new MyClassMappingKinds();

            // Define several type-less mappings for the same source, with different
mapping kinds.

            Type targetType = target.GetType();
            var source = new DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp",
1000, DADataSource.Cache);

            mapper.DefineMapping(
                source,
                new DAClientItemMapping(typeof(Double)),
                new ObjectMemberLinkingTarget(targetType, target, "CurrentValue"));

            mapper.DefineMapping(
                source,
                new DAClientItemMapping(typeof(Double), DAItemMappingKind.Vtq),
                new ObjectMemberLinkingTarget(targetType, target, "CurrentVtq"));

            mapper.DefineMapping(
                source,
                new DAClientItemMapping(typeof(Double), DAItemMappingKind.Exception),
                new ObjectMemberLinkingTarget(targetType, target, "CurrentException"));

            mapper.DefineMapping(
                source,
                new DAClientItemMapping(typeof(Double), DAItemMappingKind.Result),
                new ObjectMemberLinkingTarget(targetType, target, "CurrentResult"));

            // Perform a subscribe operation.
            mapper.Subscribe(true);

            Thread.Sleep(30 * 1000);

            // Perform an unsubscribe operation.
            mapper.Subscribe(false);
        }
    }
}
```

Various mapping kinds with type-less mapping in VB.NET

```
' This example for OPC DA type-less mapping shows how to define mappings of various
kinds, and perform subscribe and
' unsubscribe operations.

Imports OpcLabs.BaseLib.ComponentModel.Linking
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Generic
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping

Namespace DocExamples._DAClientMapper
    Partial Friend Class DefineMapping
        Class MyClassMappingKinds
            Public WriteOnly Property CurrentValue As Double
                Set(value As Double)
                    ' Display the incoming value
                    Console.WriteLine("Value: {0}", value)
                End Set
            End Property

            Public WriteOnly Property CurrentVtq As DAVtq(Of Double)
                Set(value As DAVtq(Of Double))
                    ' Display the incoming Vtq
                    Console.WriteLine("Vtq: {0}", value)
                End Set
            End Property

            Public WriteOnly Property CurrentException As Exception
                Set(value As Exception)
                    ' Display the incoming exception
                    Console.WriteLine("Exception: {0}", value)
                End Set
            End Property

            Public WriteOnly Property CurrentResult As DAVtqResult(Of Double)
                Set(value As DAVtqResult(Of Double))
                    ' Display the incoming result
                    Console.WriteLine("Result: {0}", value)
                End Set
            End Property
        End Class

        Public Shared Sub MappingKinds()
            Dim mapper = New DAClientMapper()
            Dim target = New MyClassMappingKinds()

            ' Define several type-less mappings for the same source, with different
```

```
mapping kinds.

            Dim targetType = target.GetType()
            Dim source = New DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp",
1000, DADataSource.Cache)

            mapper.DefineMapping(
                  source,
                  New DAClientItemMapping(GetType(Double)),
                  New ObjectMemberLinkingTarget(targetType, target, "CurrentValue"))

            mapper.DefineMapping(
                  source,
                  New DAClientItemMapping(GetType(Double), DAItemMappingKind.Vtq),
                  New ObjectMemberLinkingTarget(targetType, target, "CurrentVtq"))

            mapper.DefineMapping(
                  source,
                  New DAClientItemMapping(GetType(Double), DAItemMappingKind.Exception),
                  New ObjectMemberLinkingTarget(targetType, target, "CurrentException"))

            mapper.DefineMapping(
                  source,
                  New DAClientItemMapping(GetType(Double), DAItemMappingKind.Result),
                  New ObjectMemberLinkingTarget(targetType, target, "CurrentResult"))

            ' Perform a subscribe operation.
            mapper.Subscribe(True)

            Threading.Thread.Sleep(30 * 1000)

            ' Perform an unsubscribe operation.
            mapper.Subscribe(False)
        End Sub
    End Class
End Namespace
```

5.2.1.11.5 Error Handling in the Mapper Object

There are several types of errors that can occur in relation to live mapping, and they are treated differently. It is important to understand what happens in various situations, so that you can write code that behaves properly.

In the list below, we have tried to sort the possible types of errors by severity, starting with the most severe errors first.

A. Errors during object mapping. These are errors that are encountered when you are associating your objects

(mapping targets) with mapping sources, usually by means of calling one of the Map method overloads on the mapper object. Typically, these errors are cause by improper usage of mapping attributes, such that the mapping cannot be created at all. They are basically coding errors, and are reported by throwing a MappingException during the mapping (the Map method call).

B. Operation execution errors. These are logical errors that happen when you invoke an operation on the mapper, such as Read, Write, Get or Subscribe. Operation execution errors are similar to the errors encountered during object mapping, except that they could not have been discovered earlier. For example, you may have a conflicting combination of attributes that does not allow the OPC Write operation be performed. Operation execution errors are basically coding errors as well, and are reported by throwing an ExecutionException from the operation method invoked.

C. Update failures. The update failure means that a mapping target could not be updated with a new value. Each update failure is indicated by an UpdateFailure event raised on the mapper object. The event notification contains an Exception indicating the cause of the update failure, and a reference to the mapping source. You can hook your event handler to this event if you need some action be performed in such case.

   There may be various reasons for update failures; below are the common specific causes.

   1. Target access errors. They happen when the live mapping encounters a failure accessing your objects (mapping targets). For example, setting or getting a property on your object may throw an exception, and such exception will be caught. Target access error is represented by an instance of TargetAccessException in an update failure.
   2. Validation errors. These are errors caused by invalid values passed to or from the mapping target. For example, you may be attempting to write a null reference DAVtq object, a null timestamp, or a null quality. Validation errors are represented by an instance of ValidationException in an update failure. Note that these are validations on the client side, not any validations made by the OPC server.

D. Mapping source problems. These are errors related to the actual mapping sources, such as OPC items or OPC properties. For example, the OPC server may be shut down, or the item ID might have changed name and is no longer accessible. Mapping source problems may actually be THE important errors for you to handle, but they have the least severity as far as the live mapping is concerned, because they are treated as just another data being mapped (and sent to your application). There are mapping kinds (see Mapping Kinds) such as Exception or ErrorId that allow you to set up members on your objects that will receive error information in case of mapping source problems.

5.2.2 OPC-UA Modelling (Preliminary)

This development model automatically provides .NET objects that logically correspond to OPC-UA information model. The operative parts of this model are internally implemented using Live Mapping.

Currently, only certain standard OPC-UA variable types are supported; custom types are not yet available. Supported concrete classes include: UAPropertyNode, UABaseDataVariable; for UA Data Access: UADataItemNode, UAAnalogItemNode, UATwoStateDiscreteNode, UAMultiStateDiscreteNode.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

5.2.2.1 How It Works

Here are the steps that you need to get started with using OPC-UA modelling approach in your code:

1. Create an instance of UAModelClient class that will use the information model resident in the OPC server. You obtain this UAModelClient object by calling a static method UAModelClient.CreateWithServerModel().
2. When you want to work with a node in the OPC server, obtain the corresponding node object by calling the GetNode method on the UAModelClient object. There are also extension methods that are useful to obtain nodes with a type that you expect. For example, to obtain a node for an analog variable of type TValue, use the GetAnalogItemNode<TValue> extension method.
3. a. If you want to read data from the node, call the ReadNode method on the UAModelClient object, passing it the node object, and optionally the so-called modelling tag. Use UAModellingTags.VariableValue to operate only on the value of the variable. Use UAModellingTags.VariableProperty to operate on all variable properties. After the ReadNode call, the properties on the node object contain the values obtained. For example, the Value property contains the current value of the variable. Or, the EURange property contains the engineering units range.
   b. If you want to write data into the node, set the properties of the node objects (such as the Value property), and then call the WriteNode method on the UAModelClient object. You can optionally use the modelling tags, as with the ReadNode method.
   c. If you want to subscribe to (or unsubscribe from) changes in the node data, call the SubscribeNode method on the UAModelClient object, passing it the "active" flag (determines whether you want to subscribe or unsubscribe) and the node object. You can optionally use the modelling tags, as with the ReadNode method. The properties of the node object (such as the Value property) will be automatically be updated with changes from the OPC server.

5.2.2.2 OPC UA Node Types

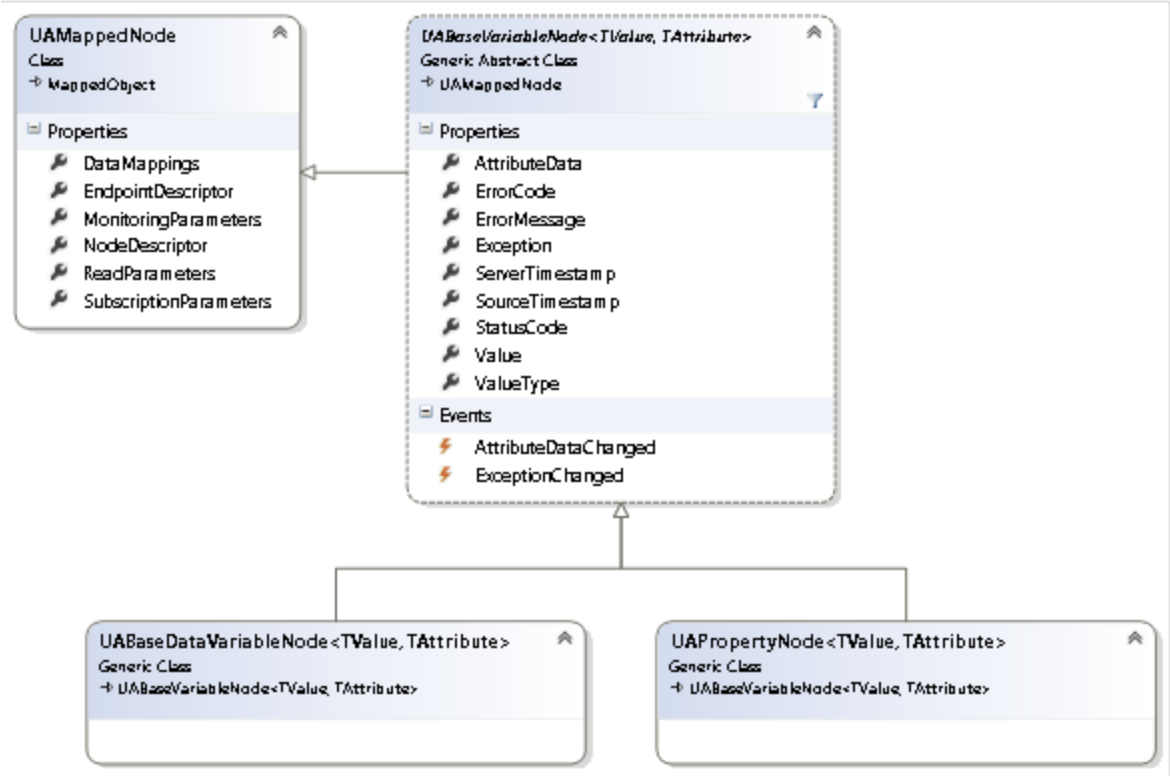
OPC Data Client contains definitions of certain OPC-UA node types that can be used with the modelling. Currently, these are the only node types supported, and the list cannot be extended. Future OPC Data Client versions will support the creation of use of custom node types with OPC-UA modelling.

The .NET classes for various OPC-UA node types use an inheritance scheme that follows the inheritance hierarchy in OPC-UA information model. The UABaseVariable is the root of this hierarchy; in itself, it derives from UAMappedNode, which means that it contains meta-information about itself, such as the EndpointDescriptor or NodeDescriptor, and also parameters that define details of operations that performed on the node – such as ReadParameters, or SubscriptionParameters and MonitoringParameters.

Each UABaseVariable contains properties that describe the state of the node either individually (Value, StatusCode, timestamps, ...), or in a summary way (AttributeData). There are also properties that give information about errors encountered during the operations (Exception, ErrorId, ErrorMessage).

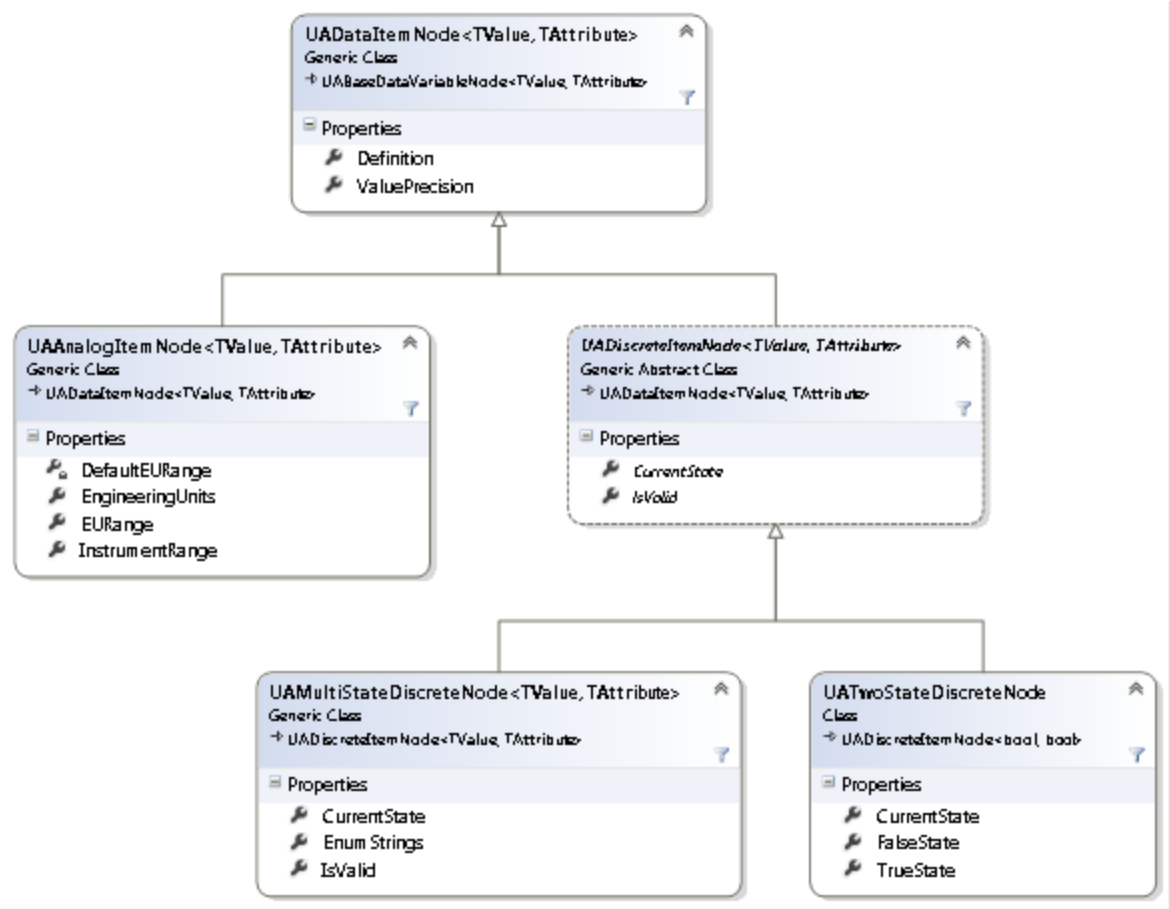
You can hook to AttributeDataChanged and ExceptionChanged events to get your code invoked whenever the relevant data of the OPC-UA variable changes.

The hierarchy of basic OPC-UA node type classes, and their most important members, are shown on the following diagram.



For use in Data Access, OPC-UA defines additional variable types, and they are reflected in OPC Data Client .NET node types as well. When working with OPC-UA Data Access modelling, you will most likely use at least one of the following classes: UAAnalogItemNode (for analog variables), UATwoStateDiscreteNode (for digital – Boolean variables), or UAMultiStateDiscreteNode (for enumeration-type variables).

The hierarchy of OPC-UA Data Access node type classes, and their most important members, are shown on the following diagram.



The classes contain members that correspond to OPC-UA properties defined for these kind of variables. For example, the Definition, ValuePrecision, EngineeringUnits, EURange etc. properties all contain information that further describes the variable. In addition, for discrete variables, the classes provide transformation between the "raw" values of the variable, and their string representation. The CurrentState property contains the current value of the variable, expressed as a string.

5.3 Live Binding Model

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

> This functionality is not yet available for OPC UA PubSub.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

In This Topic

Microsoft .net With live binding development model, no manual coding is necessary to obtain OPC connectivity. You simply use the Visual Studio Designer to configure bindings between properties of visual or non-visual components, and OPC data. All functionality for OPC reads, writes, subscriptions etc. is provided by the OPC Data Client components.

Note: Several binding technologies for.NET already exist, some directly from Microsoft. These technologies are mainly focused on binding database or similar data, and they are not well suitable for automation purposes like OPC. We therefore had to implement a new binding model. We use the term “live binding” so that it is clearly distinguished from other binding technologies used; it also reflects the fact that this kind of binding is well-suited for real-time, “live” data.

The Live Binding model is currently available for:

- Windows Forms (or UI-less) containers, and
- Windows Presentation Foundation (WPF).

You can use it with OPC Data Access, OPC XML-DA, or with OPC Unified Architecture data.

An important aspect of the live binding model is that it is not limited to or associated with any particular set of UI controls. It works by extending the features of existing controls, and providing functionality on top of what the controls already do. Live binding works well with standard Microsoft controls, but also work with controls from other vendors. This approach gives the live binding an enormous power, because you can combine the advanced features of free or commercial controls with the OPC connectivity.

In the examples installed with the product, we provide a demonstration of the integration capability described above.

There are demos that show how Live Binding can be used with several 3rd party products, some aimed directly at industrial automation area.

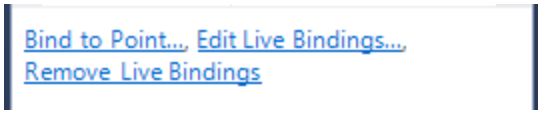
Live Binding Features

Live Binding works with OPC Data Access ("Classic") and OPC Unified Architecture servers (for data access). Its features include:

- You can bind to any OPC item, node attribute or property.
- You can bind .NET object properties, methods, and even sub-members. Members of extender components (such as ToolTip or ErrorProvider) can be bound, too!
- Works with subscriptions, but also one-time reads and writes.
- Bindings can be grouped for easier configuration and control.
- You can convert and format the values in various ways.
- Design just by dragging components from Toolbox and configuring them.
- Work with array elements using the element extraction feature, and built-in read-modify-write algorithm.
- Different binding kinds such as Value, Quality, Timestamps, ErrorMessage, and many more.

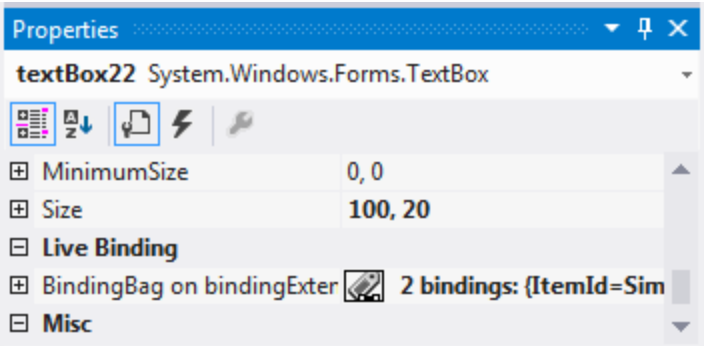
Live Binding Fundamentals

The live binding model is allowed by the BindingExtender, PointBinder and DAConnectivity (or UAConnectivity, CompositeConnectivity) components that you place from the toolbox items onto the component tray. You can then use extender commands in the Properties window, or on the context menu of each component (control), to bind the component’s properties to OPC data:

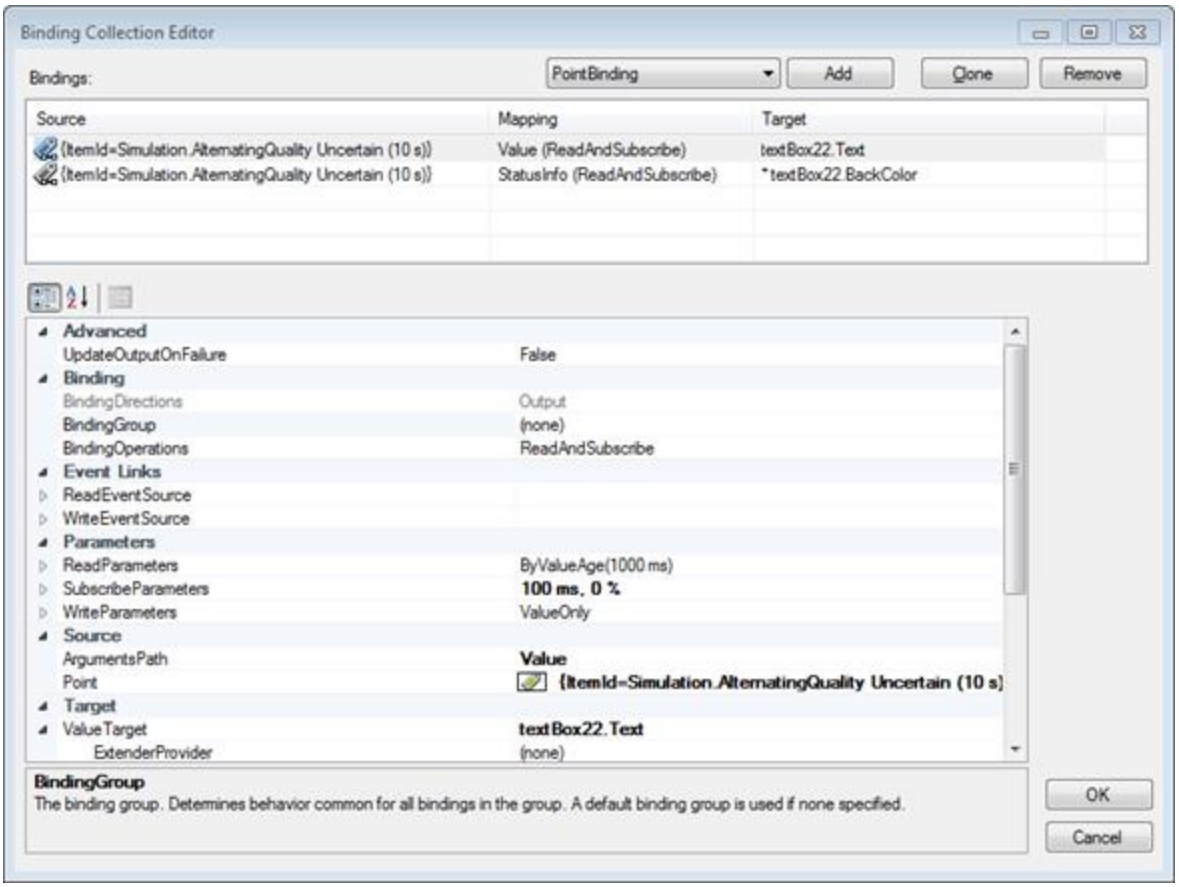


After selecting the “Bind to Point” command, you select the OPC data you want to bind to, and you are done. The

currently configured bindings appear as additional entries in the Properties window:



You can also invoke a full-featured binding editor, and configure all aspects on bindings – either on individual component (control), or for a whole container (form) at once:



5.3.1 Live Binding Model for OPC Data (Classic and UA)

Live Binding is supported to OPC data points, i.e. identified values (items, nodes) that are changing over time. Live Binding is currently not available for other types of OPC data, such as OPC alarms, events or conditions.

Do not confuse the live binding model with live mapping, also provided by OPC Data Client. The main differences between the two are that

- live binding is for UI (User Interface), and is therefore visual, both in design and in runtime, and
- live binding works with individual object members, and not on whole types.

Following resources will help you get started with Live Binding:

- The Getting Started section in this document: Contains instructions on how to create an application with live binding in a few steps.
- For OPC-DA in Windows Forms, the "Live Binding Demo" application that is installed with the product; its source code is contained in the **WindowsForms\LiveBindingDemo** project. The application contains annotated live binding configurations for typical scenarios.
- For OPC-UA in Windows Forms, the "UA Live Binding Demo" application that is installed with the product; its source code is contained in the **WindowsForms\UALiveBindingDemo** project. The application contains annotated live binding configurations for typical scenarios.
- For OPC-UA in WPF, the "WPF Live Binding Demo" application that is installed with the product; its source code is contained in the **WPF\WpfLiveBindingDemo** project. The application contains annotated live binding configurations for typical scenarios.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

5.3.1.1 Live Binding Overview

The live binding allows you to extend your controls and components with one or more *bindings*. The binding can perform various *operations*, such as read or write values, or continuously update (subscribe). The data reside in the *binding source* – e.g. in OPC Data Access or OPC Unified Architecture server, and are provided through a *connectivity* component for that binding source. The other end of the binding is the *binding target*. The usual binding targets can accept or provide values of various types, and are therefore referred to *as value targets*. When values are transferred between to or from the target, *value conversions* can be made – using *string formats*, or *converters*.

The live binding functionality is provided by a generic *binding extender*. The specific types of bindings are made available by *binding providers*, which are encapsulated in *binder* components. The binding extender keeps *binding bags*, i.e. collections of bindings for each control being extended.

Operations on bindings can be tied to *event sources* and thus performed automatically with events in your application. In order to handle multiple bindings together, it is possible to define *binding groups* and assign bindings to them. *Binding templates* can be defined to make it easier to create new bindings that share similar characteristics.

The objects and terms briefly mentioned above are described in more detail sin the subsequent chapters.

5.3.1.1.1 Value Targets

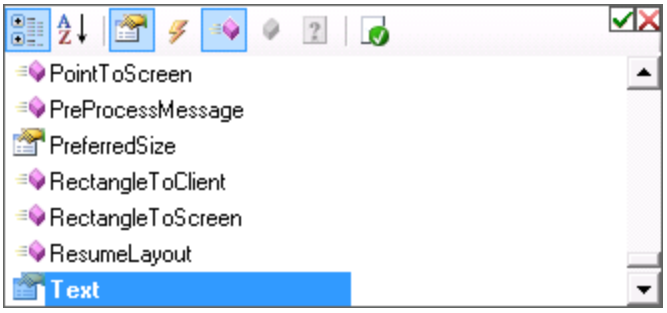
The *value target* represents a place in your application where values can be sent to or obtained from. You configure the value target by selecting the TargetComponent and TargetMember. A typical value target is a property of a visual control on your form. For example, in order to send values to the Text property of the TextBox control, you would set the TargetComponent to the text box control itself, and the TargetMember to a string "Text".

It is also possible to target the so-called extender properties. In such case, you also need to select the ExtenderProvider component that actually provides the extender property. Typical and useful extender providers are the ToolTip and ErrorProvider components for Windows Forms.

When you select the TargetComponent (and the TargetMember is not set yet), the live binding system attempts to determine the TargetMember that you want to use. For example, the most commonly used property of the TextBox control is the Text property, so it makes sense to provide that as a default. This default member name is determined using several hints, including the DefaultBindingProperty and DefaultProperty attributes on the target component, and other characteristics.

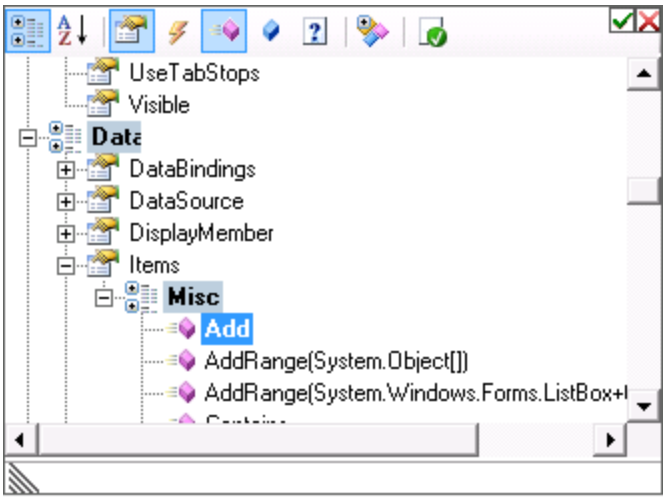
Not all controls (especially 3rd party controls) have their default members marked up (or, they may be marked up inappropriately for live binding). The live binding contains some built-in knowledge that allows proper selection of the default target member for popular controls, even in such cases. If you still get an improper target member selected for you by default, you can always change it to any control's member you like.

You can type in the name of the target member, or you can select it using the drop-down editor (picture below).



The member name editor shows different icons for different member kinds, annotates the members with number of overloads if needed, shows member information (such as type, and description) in the tooltip, allows filtering (e.g. Properties, Event, Methods, Fields), alphabetical or categorized view, and reverting to default member.

When binding .NET members, you can specify not just a member name, but a whole path to it – i.e. it is possible to bind sub-members as well. The path is a sequence of member names, separated by dots. For example, for binding on a ListBox component, it is possible to set the target path to "Items.Add", which means that we bind the Add method of the Items collection of the ListBox (in effect, this will append incoming values as new values to the ListBox). If you want to specify the path (sub-members) and not just a single member, you use the TargetPath property instead of just TargetMember. By pressing a down-arrow symbol next to the path, you are offered a tree of available members and sub-members, with various filtering and sorting capabilities:



When binding .NET members (methods) that have multiple overloads, it is possible to specify the desired method signature using a type name in parenthesis, added to the element path. For example, if you wanted to bind the Add method of the Items collection of a ListView, there are two overloads that take one input argument, and the Live Binding would not know which one to use. By using a target path "Items.Add(System.String)", you will instruct the Live Binding to use the overload that accepts string as an input.

5.3.1.1.2 Binding Operations

There are several things you can do with OPC Data: you can read from them, write to them, or subscribe to their changes. Not all bindings should be involved in all operations, and you therefore need to specify the operations that are of interest to you.

You can choose one of the following values, or their combination:

- PointBindingOperations.Read: A Read operation.
- PointBindingOperations.Write: A Write operation.
- PointBindingOperations.Subscribe: A Subscribe operation.

If you want a combination, you can use a predefined symbolic name such as ReadAndWrite, ReadAndSubscribe, etc. By default, the BindingOperations argument is set to PointBindingOperations.ReadAndSubscribe, meaning that the binding will be involved in Reads and Subscribes, but not in Writes.

5.3.1.1.3 Bindings

A binding is a correspondence between two entities that allows information be passed between them (either in one of the directions, or in both directions).

The main parts of binding are:

- **(Binding) Source**: The external data you want to bind to. Here you select e.g. the OPC item, update rate etc. You can also select which "part" of the data you want to bind to (e.g. just the value, or just the quality information, etc.).
- **(Binding) Target**: Defines element in your application that will be bound to the binding source. This is typically a value target (see Value Targets).
- **Binding Operations**: Which operations the binding is involved in.
- **Binding Groups**: Which binding group the binding belongs to.
- **Value Conversion**: Allows you to convert the values on their way to/from the binding target.
- **Event Links**: Defines events in your application that will automatically cause binding operation be performed.

The types of bindings that are available depend on the binder in use (see further below). For the (most common) PointBinder, there is just one type of binding available, a PointBinding. The point binding can be e.g. for an OPC-DA item, an OPC-DA property, or for OPC-UA data (node attribute).

There may be multiple bindings associated with a control. With this feature, you can, for example, bind a value (converted to string) to the actual text box's text, while binding the status information to its color.

5.3.1.1.4 Binding Bags

A binding bag is essentially a collection of bindings, with a reference to a common component to which the bindings belong. The live binding system keeps a binding for each component that is extended with bindings, and also allows you to access a common binding bag that represents all bindings defined in a container.

One binding bag can hold bindings of different types, and bindings that have different sources.

5.3.1.1.5 Binding Extender

The binding extender is a central piece of live binding model. It is represented by the BindingExtender component; this component must be present on the component tray (of your form or window) in order to enable it for live binding.

The main purpose of the binding extender is to keep track of bindings defined on each control in the container (form), and provide extender properties to them.

The binding extender is generic; it does not have a specific knowledge about types of bindings that can be used. The available binding types are defined on the binding extender by a binder (or binders) that register with it. For explanation of binders, see further below.

5.3.1.1.6 Binders and Binding Providers

While the binding extender is a generic coordination component without specific knowledge about possible types of bindings, the *binding provider* is a component that is specific to certain technology and defines binding types tailored to that technology.

A binding provider has the ability to create one or more types of bindings, as necessary for the technology it covers. Binding providers exist in form of *binder* components; the binder is a binding provider (can create new bindings) with certain additional capabilities, such as event linking, and ability to perform operations on bindings.

Each binding provider needs to register itself with the binding extender; this is done by properly setting the BindingExtender property of the binder. When you place the binder component from the Toolbox Items to your form's or window's component tray, this setting is already made for you.

The most commonly used binder component is the PointBinder, which is described further below in a separate chapter. The PointBinder provides a unified approach to live binding; it merges the commonalities between OPC "Classic" and OPC Unified Architecture bindings into a single component.

Note: Earlier, OPC Data Client had used separate DABinder (for OPC Data Access) UABinder (for OPC Unified Architecture) component. They are now obsolete.

5.3.1.1.7 Event Sources

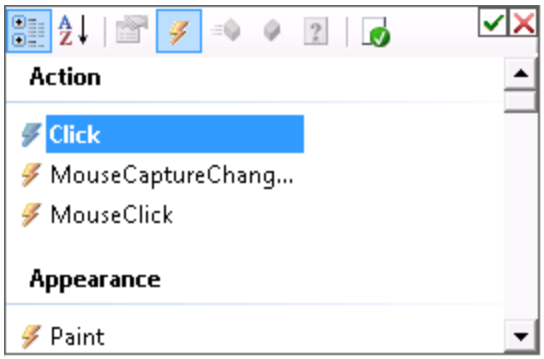
The *event source* represents a place in your application where events are generated. You configure the event source by selecting the SourceComponent and SourceMember. A typical event source is a Click event of a Button on your form.

Event sources are used in live binding to automate actions that would otherwise have to be implemented in the code by writing an event handler.

For example, if you are designing a form or window with HMI functionality, you probably want it to subscribe to value changes of the configured items when it is shown. This can be achieved by properly setting the OnlineEventSource event source of the BindingExtender component (and indeed, when you place the BindingExtender from the toolbox items onto a component tray, this event source is pre-set like this already).

Similarly, you may want to write some value into a bound OPC item when a button is clicked. Instead of writing an event handler, you can simply configure the WriteEventSource on the binding (or binding group) to the Click event of the Button, and you are done.

You can type in the name of the source event, or you can select it using the drop-down editor (picture below).



The member name editor shows member information (such as type, and description) in the tooltip, and allows alphabetical or categorized view, and reverting to default member.

You can specify not just a member name, but a whole path to it – i.e. it is possible to bind sub-members as well. The path is a sequence of member names, separated by dots. If you want to specify the path (sub-members) and not just a single member, you use the SourcePath property instead of just SourceMember. By pressing a down-arrow symbol next to the path, you are offered a tree of available members and sub-members, with various filtering and sorting capabilities.

5.3.1.1.8 Binding Groups

A *binding group* has the ability to handle multiple bindings together. For example, you can create a binding group, assign several bindings to it, and then have them all read or written by a single command.

The use of binding groups is optional. If you do not assign a binding to any binding group, it will belong to a *default binding group*, which always exists on the binder component.

5.3.1.2 Point Binder

The point binder (represented by a PointBinder component) provides live binding to generic data points (see further below). It supports just one type of binding – a PointBinding; however, the point binding type is so generic that it can easily describe OPC-DA "Classic" items, properties, OPC-UA nodes (attributes), or just about any other point-based data source.

The point binder makes use of the **Connectivity Model (Section 6.2)**, descrined in a separate section. Please refer to the relevant parts of documentation for information about the Connectivity Model.

In This Topic

Connectivities

Connectivities (Section 6.2.1) provide access to data sources.

The developer will usually start the live binding by placing one of these connectivity components from the Visual Studio **Toolbox** onto the component tray (in Windows Forms, by dragging from the **Toolbox** to the design surface of the form; in WPF, using the Component Tray context menu command, and the **Component Tray Editor** dialog then).

Notice that after placing any of the connectivity components onto an empty component tray, three components will appear below on the component tray: Besides the one for the connectivity itself, there will be one for the PointBinder, and one for BindingExtender.

If, after placing a connectivity component to a component tray, the PointBinder and the BindingExtender **components are not placed onto the component tray automatically, make sure that the target framework of the project is set to ".NET Framework 4.7" or later.**

Points

Points (Section 6.2.2) identify the pieces of data the Live Bindingg works with.

Point Editor

Point Editor (Section 6.2.3) is the dialog used to select, view or modify points.

Parameters

Parameters (Section 6.2.4) determine how the data will be accessed.

Arguments and Results

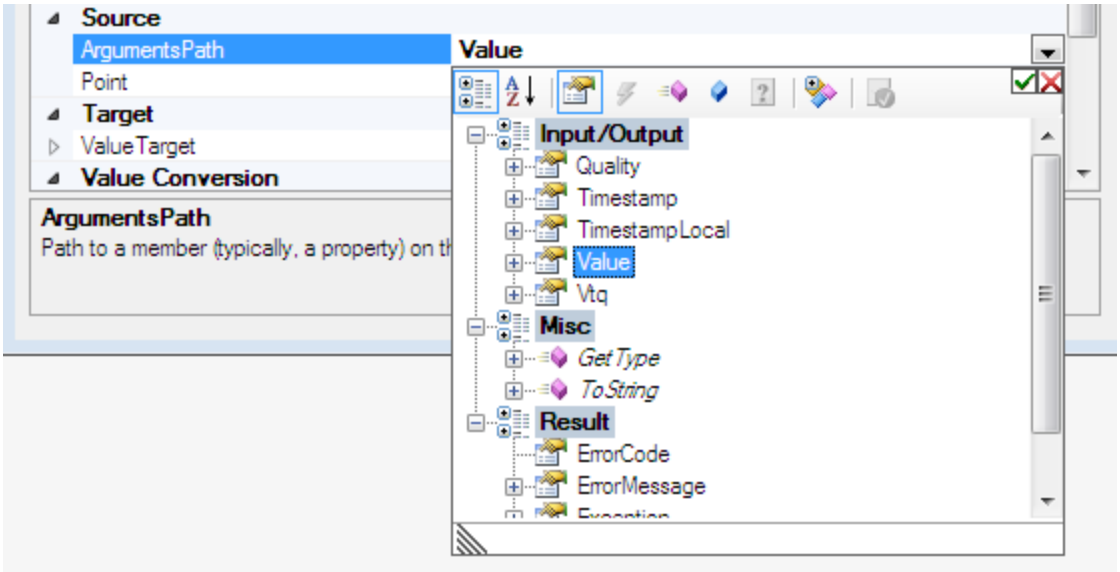
The **Arguments and Results (Section 6.2.5)** are the actual data the binding operation transfers, i.e. the "what".

The PointBinding has an UpdateOutputOnFailure property (defaults to 'false'). Normally, when an operation (such as OPC read) fails, and you are binding to an operation output, nothing happens - the binding output is not updated. For example, a text bound to an OPC data value continues to display the last known value. When the UpdateOutputOnFailure property is set to 'true', the binding output will always be updated. In our example, the text box will be cleared in case of operation failure.

5.3.1.2.1 Arguments Path

When making a binding, you are basically connecting a value target (typically, a property on a visual control) with an argument (described in the above chapter). The argument being used is specified using an *argument path*. An argument path can be simply a property name (name of the argument), such as Value, or a sequence of member names (separated by dots) that determines the path into the argument; for example, Quality.StatusBitField.

In the Visual Studio designer, the arguments path can either be typed in, or you can select using a drop-down editor, as on the following picture:



A binding can be involved in multiple binding operations (e.g. read and write), but there is just a single arguments path selection for the whole binding, and the set of arguments might be different for each operation. The arguments path should be valid for all the binding operations that are enabled on the binding. The drop-down editor offers a combined view of the possible arguments. For this reason, argument categories can change (by combining) as well: For example, in OPC DataAccess, a Value is an output in the Read operation, but it is an input in the Write operation. It is therefore shown under the Input/Output category, when both Read and Write binding operations are enabled on the binding.

5.3.1.2.2 Parameter Templates

When you create new bindings, often they share many common characteristics. For example, you may be binding to OPC items all residing in the same OPC server, or using the same parameters, such as the update rate.

In order to make it easier to create such similar bindings, parameters of newly created bindings are taken from *parameter templates*.

For OPC Data Access (OPC-DA), you can configure following templates on the DAConnectivity component:

- ItemReadParametersTemplate: OPC-DA read parameters, such as the data source (cache, device, and so on), in newly added bindings to OPC items.
- ItemSubscribeParametersTemplate: OPC-DA group parameters in newly added bindings to OPC-DA items.
- ItemWriteParametersTemplate: OPC-DA write parameters (such as whether just the value, or the whole VTQ triple should be written) in newly added bindings to OPC-DA items.

For OPC Unified Architecture (OPC-UA), you can configure following templates on the UAConnectivity component:·

- AttributeReadParametersTemplate: OPC-UA read parameters (such as the maximum value age) in newly added bindings to OPC-UA attributes.
- AttributeSubscribeParametersTemplate: OPC-UA subscription and monitoring parameters (such as the sampling interval, or data change filter) in newly added bindings to OPC-UA attributes.
- AttributeWriteParametersTemplate: OPC-UA write parameters (such as whether just the value should be written) in newly added bindings to OPC-UA attributes.

Templates have meaning only in the designer; they are not used in runtime.

5.3.1.2.3 Queued Execution

With the help of QueuedExecution property on the PointBinder, the developer can choose that the operations (such as Write-s) are performed outside the calling thread, and therefore do not block. Typical usage involves a one-time Read or Write linked to a click on a button, which would otherwise block the user interface until completed.

The Boolean QueuedExecution property defaults to 'false'. When set to 'true', the operations on the binder are executed outside the calling thread, but still in the order of arrival. This allows non-blocking operation calls on the binder.

5.3.1.3 Value Conversions, String Formats and Converters

When values are transferred to or from the target, *value conversions* can be made – using *intrinsic conversions*, *string formats*, or *data converters*.

In This Topic

Intrinsic Conversions

The intrinsic conversions do not have to be configured, and are applied automatically when needed.

The Live Binding infrastructure contains following intrinsic conversions :

- System.Drawing.Color is converted to System.Windows.Media.Color when the target is of that type. This allows to use colors in the same way in both Windows Forms and WPF.
- System.Drawing.Color is converted to System.Windows.Media.SolidColorBrush when the target is of that type. This allows to use colors in the same way in both Windows Forms and WPF.

String Formats

In order to format the bound value according to a string format, simply set the StringFormat property of the binding. An empty string means no formatting. The format string is constructed according to usual .NET rules, and can contain following placeholders:

- {0}: The value.
- {1}: The type of the value (or null if the value itself is null)

The string formatting is only applied in one direction – for values being transferred from the binding source to the binding target.

In addition to formats defined by the .NET Framework, OPC Data Client defines format strings for various its types, and especially the types used in Live Binding, such as DAQuality, DAVtq, UAAttributeData, UAStatusCode, etc. This makes it easy to format compound values for display purposes, without having to write any code. For details, please refer to Format Strings in the Reference.

Data Converters

When string formatting is not sufficient, any kind of conversion can be made using data converters, and possibly in both directions. Data converters are objects that implement the IDataConverter interface. You can place converters on the design surface of the form, and then select from them for the Converter property of any binding.

OPC Data Client ships with several data converters, described in the subsequent chapters.

Besides using the data converter(s) that ship with the product, you can create your own data converter objects or components, and assign them to your bindings.

5.3.1.3.1 The LinearConverter Component

The LinearConverter component provides a linear conversion and its inverse, and also (optionally) allows output value clamping. The linear conversion is useful for proper scaling of values when the underlying OPC source does not directly have the desired value.

You configure the parameters of the linear conversion so that the X1 value converts to Y1, and the X2 value converts to Y2. Other values are interpolated or extrapolated according to these settings.

For values that fall outside the given limits, you can choose one of the following behaviors:

- Normal. The off-limit condition is ignored, the value is converted as usual (i.e. extrapolated, in this case).
- Error. An exception is thrown.
- Clamp. The output value is clamped at the limit.

5.3.1.3.2 The StatusToColorConverter Component

This type of data converter is useful if you want to change some of the control's colors (e.g. background or foreground) according to the status of the bound source. For example, you may want the background of the text box turn to red color if there is an error related to the source.

The StatusToColorConverter can convert several types of values on its input. On its output, it provides a Color value that it chooses from one of the four colors that you can configure: UnknownColor, NormalColor, WarningColor and ErrorColor. Each color corresponds to one state, and the possible state choices are given by the StatusInfo enumeration: Unknown, Normal, Warning, and Error.

The possible input values to the StatusToColorConverter are converted as follows:

- A null reference converts to NormalColor.
- A StatusInfo.Normal converts to NormalColor, a StatusInfo.Warning converts to WarningColor, a StatusInfo.Error converts to ErrorColor, and a StatusInfo.Unknown converts to UnknownColor.
- An Exception-typed value converts to ErrorColor.
- Empty String-s convert to NormalColor, non-empty String-s convert to ErrorColor.
- Zero integers convert to NormalColor, positive integers convert to WarningColor, negative integers convert to ErrorColor.
- A true Boolean converts to NormalColor, a false Boolean converts to ErrorColor.

Several types in OPC Data Client provide built-in conversion to StatusInfo enumeration, and can therefore be used as inputs to StatusColorConverter as well.

Following types support conversion to StatusInfo in OPC Data Access:

- DAQuality
- DAVtqResult
- EasyDAItemChangedEventArgs

Following types support conversion to StatusInfo in OPC Unified Architecture:

- EasyUADataChangeNotificationEventArgs
- UAAttributeDataResult
- UAStatusCode

The rules and built-in conversion are design in such way that you can easily use various kinds of bindings with the StatusToColorConverter component. For example, if you set the ArgumentsPath property of the binding to Exception, it will convert to one of two colors. Even better, if you set the ArgumentsPath of the binding to StatusInfo, it will properly convert to one of all four possible colors.

5.3.1.4 Live Binding in the Designer

The beauty of the live binding is that it can be set up without any manual coding. This means that there has to be a tight integration with the development environment (IDE), i.e. Visual Studio.

This chapter describes the various places where the live binding in integrated with the Visual Studio and its designer features.

> ⚠ While the main principles of the Live Binding remain the same regardless of the application, some of its aspects differ in design time. The differences between designing in Windows Forms and WPF are described in the subsequent chapters, where applicable.

5.3.1.4.1 Preparing for Live Binding

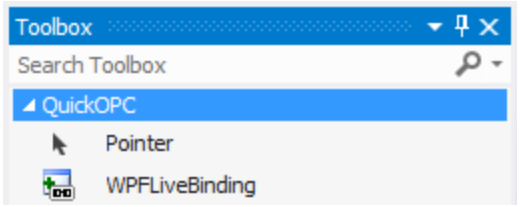
Windows Forms

No special preparation is needed in Windows Forms projects before you can use Live Binding. Specifically, the Component Tray and Toolbox Items (described in subsequent chapters) are automatically available to you in Visual Studio in Windows Forms projects.

In This Topic

WPF

In WPF, each window that you want to use with Live Binding needs to be prepared for it first, by initialize the support for Component Tray in it. To do so, drag the WPFLiveBinding component icon from the Visual Studio **Toolbox** to the design area of your WPF window.



A **Component Tray Editor** dialog appears. You can add components to it right now, or later; just make sure that you end the dialog by pressing the OK button this first time, in order for the Live Binding support be properly initialized in the designed window.

Note: If you do not see the WPFLiveBinding item in the Toolbox, or if nothing happens after dragging it to the design area, verify that your project is set to target the .NET Framework 4.7 or later.

5.3.1.4.2 Component Tray

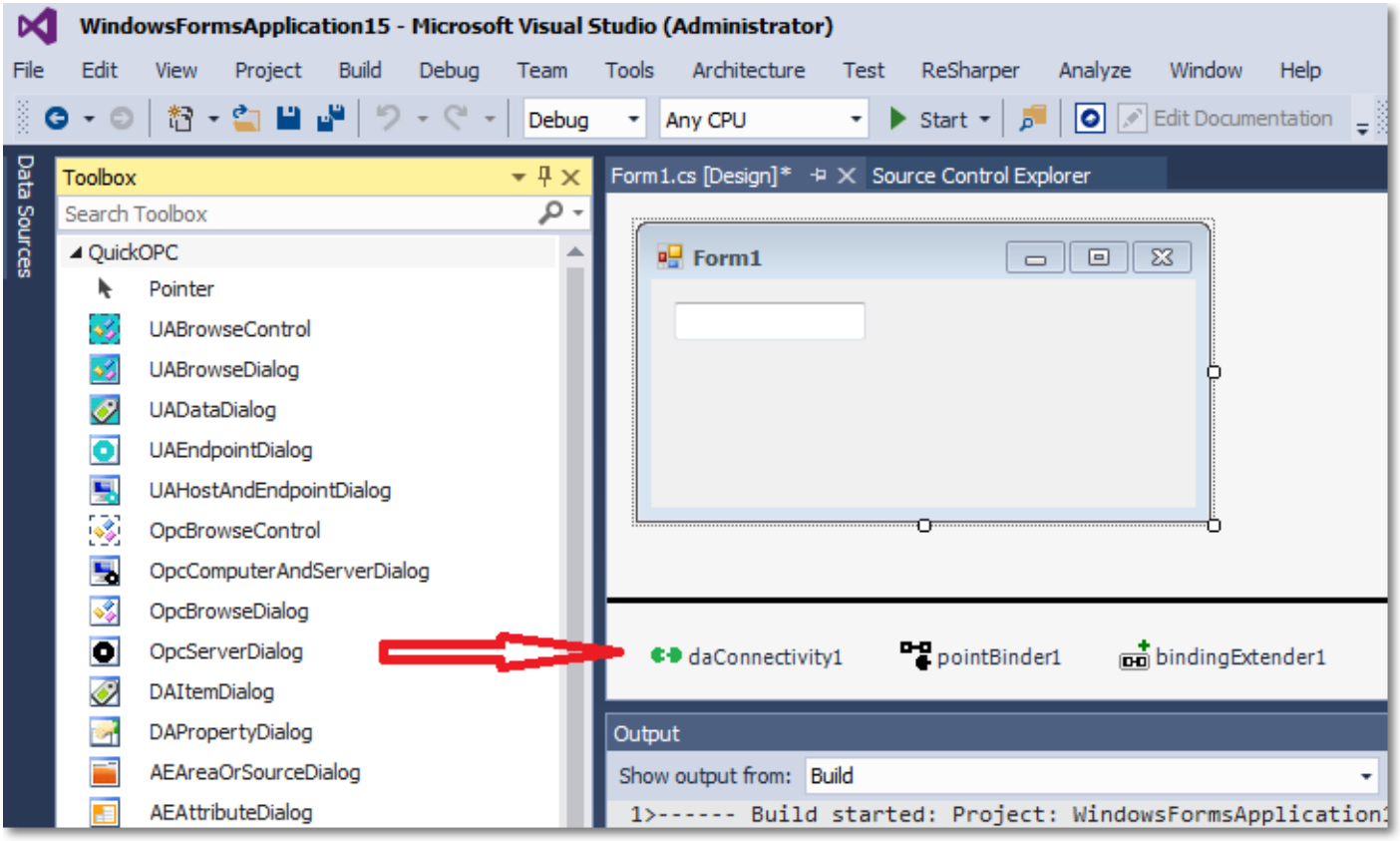
The Component Tray shows icons for nonvisual items in the designer. It allows you to work with nonvisual items separately from the controls on the form or the window being designed.

Wherever this documentation refers to a Component Tray, it needs to be interpreted depending on whether you are working with a Windows Forms or a WPF project, as described below.

In This Topic

Windows Forms

In Windows Forms, the Component Tray is provided by Visual Studio itself. It appears at the bottom of the Windows Forms Designer, as shown on the following picture:



Following operations are available:

- To add an item to the component tray, drag it from the Visual Studio **Toolbox** to the component tray area.
- To remove an item from the component tray, select it, and press the ***Delete*** key. Alternatively, after selecting the item, invoke the Delete command from its context menu, or Edit -> Delete from the main menu.
- To view or modify properties of an item on the component tray, select the item first, and then use the **Properties** window of the Visual Studio.

Changes you make to the Component Tray are reflected in the code file that the Visual Studio designer generates and

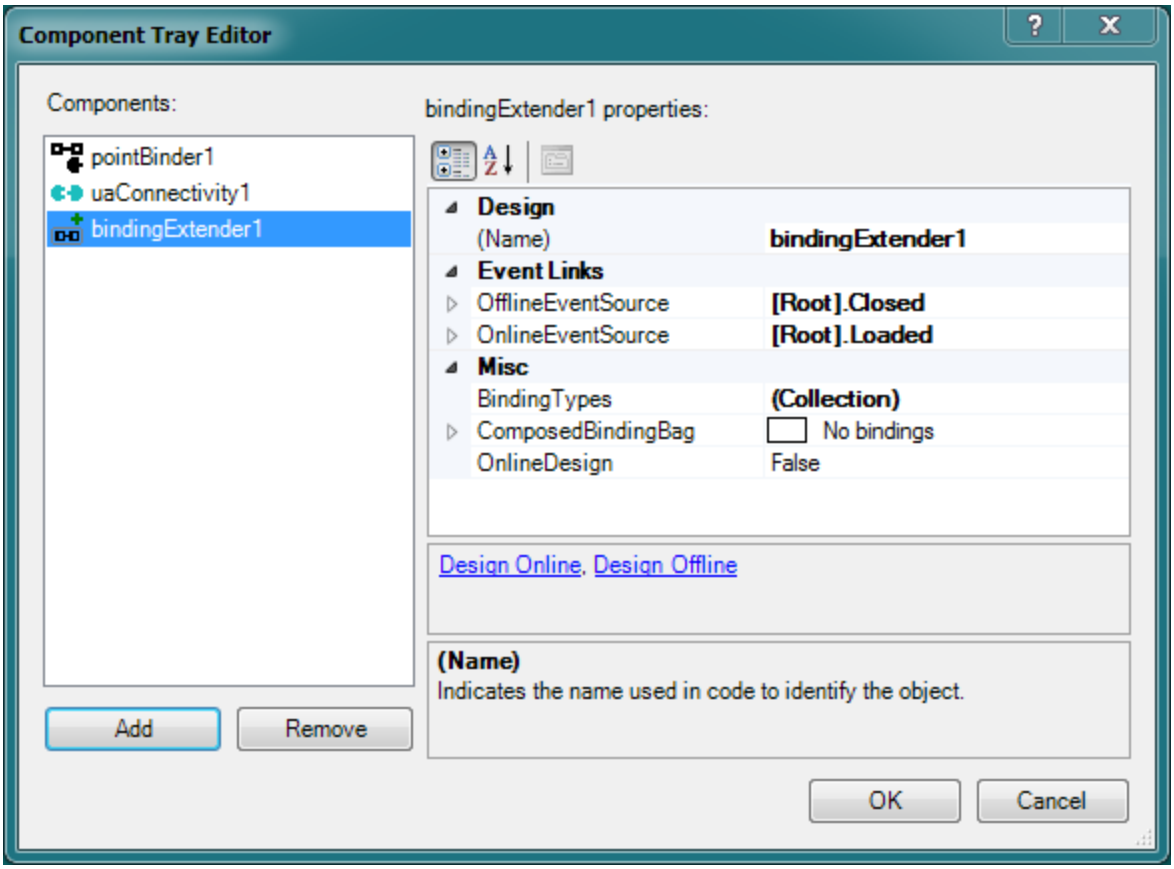
maintains for your form.

It is, of course, also possible to modify the generated code by hand, but we recommend against doing so unless you have a compelling reason for that, and know very well what you are doing.

WPF

Traditionally, WPF uses XAML to describe visual components only. For this reason, the WPF Designer does not show a component tray. In Live Binding for WPF, its functionality is provided by an add-on dialog, a **Component Tray Editor**.

In order to invoke this dialog to view or modify the items in the component tray, right-click in the design area of your WPF designer, and select the Component Tray context menu command.



Following operations are available:

- To add an item to the component tray, press the Add buton, and in the **Toolbox Items** dialog (see subsequent chapter), select the item and press OK.
- To remove an item from the component tray, select it in the **Components** list, and press the Remove button.
- To view or modify properties of an item on the component tray, select the item in the **Components** list first, and then use the property grid on the right hand side of the dialog.

Changes that you make to the Component Tray are reflected in the XAML code that the Visual Studio Designer generates and maintains for your window. Specifically, they are placed into the following XAML element:

```
<ComponentModel:Live.ComponentBag>
```

In the XAML tab of the WPF designer, it is also possible to view and modify properties of component tray items in the **Properties** window of the Visual Studio.

It is, of course, also possible to modify the generated markup by hand, but we recommend against doing so unless you have a compelling reason for that, and know very well what you are doing.

5.3.1.4.3 Toolbox Items

> ⚠ Caution: Potentially confusing terminology here. Read carefully please. "Toolbox Items" do not always reside in "Toolbox" !

In This Topic

In Visual Studio, the main place where various components originate their way to your project is the Visual Studio **Toolbox**.

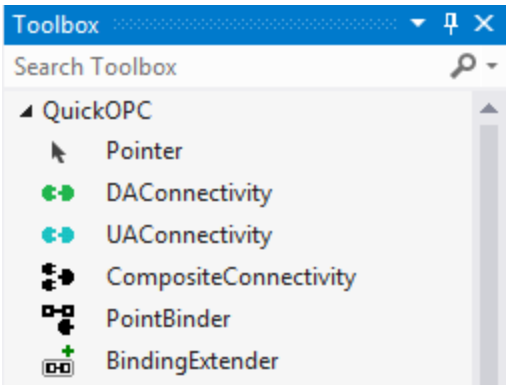
- In Windows Forms (and Web Forms), the **Toolbox** shows both visual, and nonvisual components. Toolbox Items are shown directly in Visual Studio **Toolbox**.
- In WPF, however, the **Toolbox** can only show visual components (controls). In The Live Binding uses nonvisual components as well, so in order to support them, we have added a dialog from which the components can be chosen added to Component Tray. This dialog, used for selection of nonvisual components, is referred to as the **Toolbox Items** dialog.

Wherever this documentation refers to Toolbox Items, it needs to be interpreted depending on whether you are working with a Windows Forms or a WPF project, as described below.

If the needed components do not show in the **Toolbox**: See **Troubleshooting Visual Studio Toolbox (Section 14.2)**.

Windows Forms

In Windows Forms projects, the Toolbox Items (live binding components) appear right in Visual Studio's **Toolbox** under the "QuickOPC" tab:

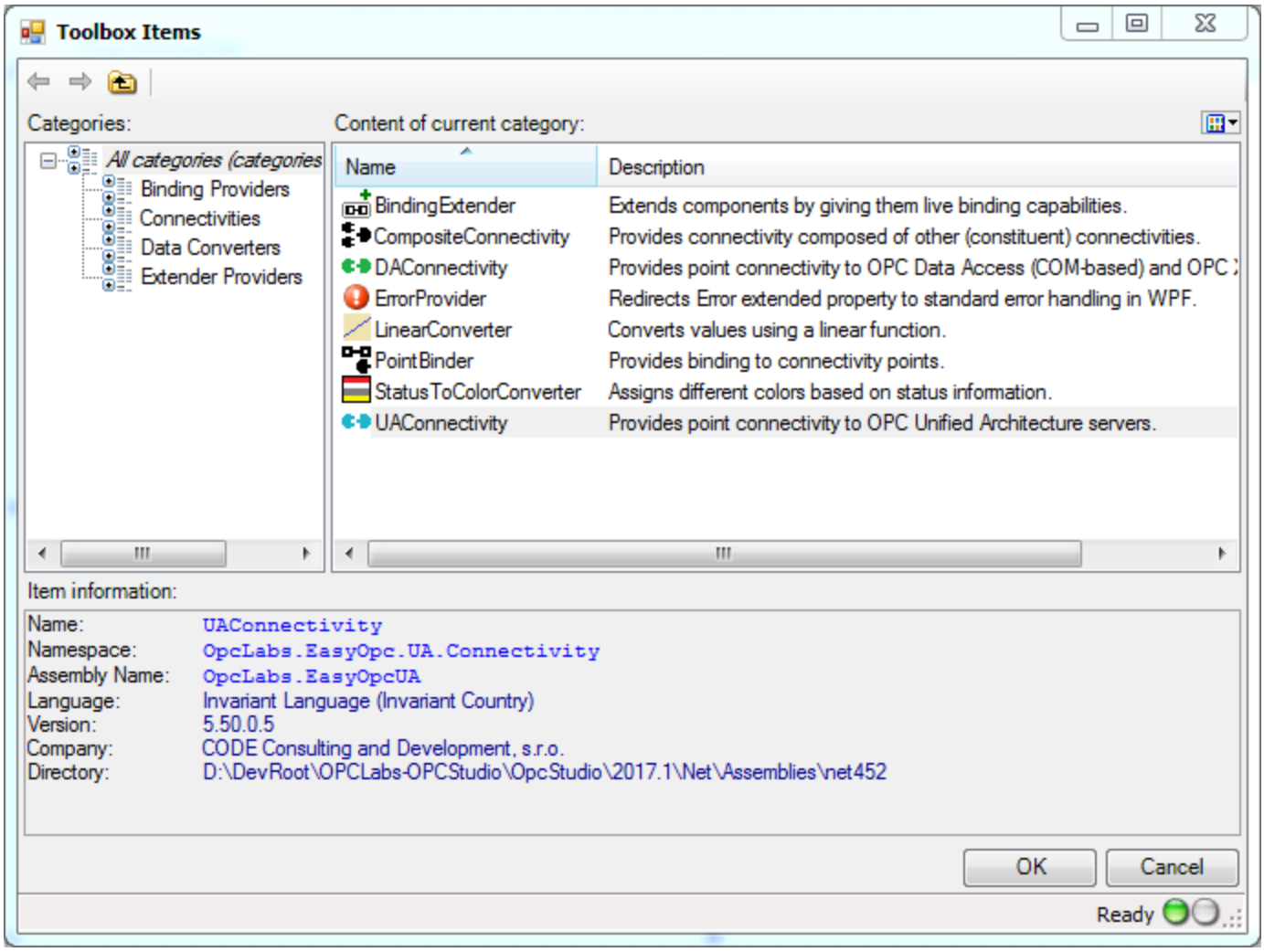


Note that in Windows Forms, the Toolbox show both visual, and non-visual items.

Note: For the components to be visible in the toolbox, your project needs to be set to target the.NET Framework 4.7 or later.

WPF

In WPF projects, you need to first invoke the **Component Tray** dialog in order to work with the items on the component tray. You can then add components from Toolbox Items to the tray by pressing the Add button. A **Toolbox Items** dialog will appear:



The **Toolbox Items** dialog in WPF shows only nonvisual components applicable to WPF projects.

Common Concerns

In order to work with live binding and enable it in your project, you need to place the desired component from the toolbox items to the component tray of your form (container) or window.

Because the PointBinder must always be used with BindingExtender in order to be usable, the toolbox item for the PointBinder uses a special logic, and assures that there is an associated BindingExtender in the container (usually, a form) as well. When you place a PointBinder onto the component tray, we first check whether some BindingExtender already exists in the container, and if so, we configure the newly created binder component to use it (by setting its BindingExtender property). If there is no BindingExtender component on the component tray, we create a new one, and configure the new binder to use it as well.

Similarly, any connectivity component is normally used with the PointBinder. Placing DAConnectivity, UAConnectivity or CompositeConnectivity onto the component tray takes with (or configures) a PointBinder as well.

As a result, in most cases you can simply “forget” about the need to place the BindingExtender or the PointBinder onto the component tray – simply place the connectivity (DAConnectivity, UAConnectivity or CompositeConnectivity) from the toolbox items onto the component tray, and the rest will be taken care of. Just remember that you should not delete the BindingExtender or the PointBinder from your form (window), even though you have not explicitly placed it there.

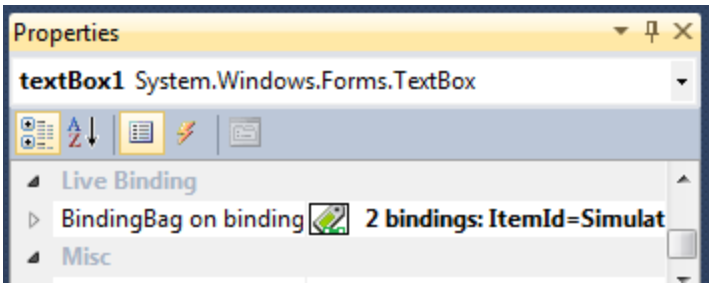
5.3.1.4.4 Extender Properties

The binding extender (BindingExtender component) has the ability to extend the properties of existing controls. When a binding extender is in the container (is placed onto the component tray), each eligible control receives new properties. Some of these properties are hidden, the remaining appear under the “Live Binding” category in the Properties window for each extended control.

In This Topic

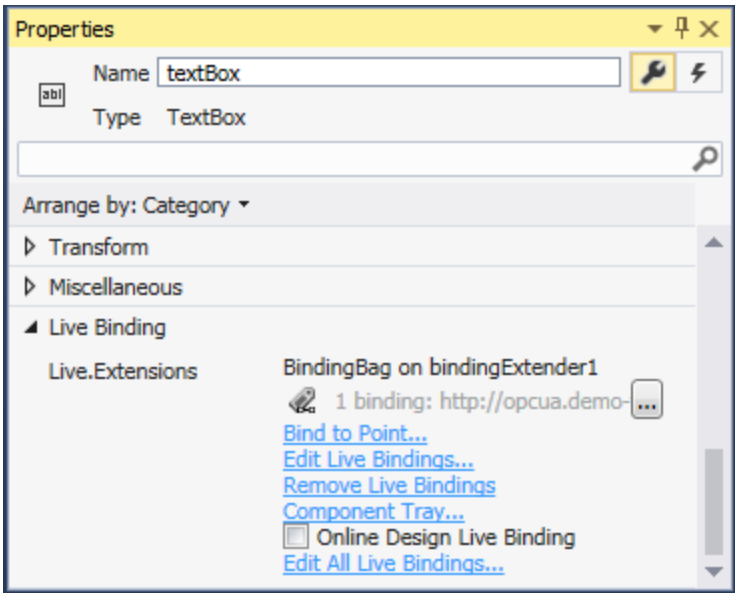
Windows Forms

In Windows Forms projects, extender properties are usually shown in the **Properties** window under the Live Binding category:



WPF

Similarly to Windows Forms projects, extender properties in WPF are usually shown in the **Properties** window under the Live Binding category, but a bit differently:



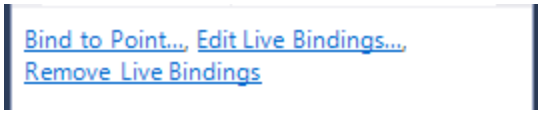
Common Concerns

The BindingBag extender property contains the binding bag, i.e. the collection of live bindings on the control. The designer shows the number of bindings contained in the bag. If they have the same type, it also shows a corresponding icon for that binding type, and if they have the same source, it also shows what the source is.

5.3.1.4.5 Designer Commands

Placing a BindingExtender onto a component tray automatically extends the set of commands (so-called Designer Verbs) available for other components (controls) in the same container (usually, a form in Windows Forms, or a window in WPF).

The new commands are available in the context menu for each component (when you right-click on the component or control), and also in the property grid for that component (in the **Properties** window), when the component is selected:



In Windows Forms projects, if you do not see the commands in the **Properties** window, right-click in the property grid area, and make sure that the "Commands" menu item is checked.

Following designer commands are available:

Bind to Point:

Allows you to select a point (OPC-DA item, property, OPC UA node/attribute) first, and then binds its value to the selected control, for the Read and Subscribe operations. It binds to the default target member of the control.

You can subsequently use the "Edit Live Bindings" command if you want to modify the target member selection, or other characteristics of the binding.

Edit Live Bindings:

Invokes the Binding Collection Editor, allowing you to add, remove, and modify bindings on the selected control.

Remove Live Bindings:

Removes all live bindings currently defined on the selected control.

When repeatedly using any of the "Bind to" commands, the browsing continues at the recently selected node. This allows for easier binding of multiple components to a set of related nodes.

When repeatedly using any of the "Bind to" commands, the browsing dialog retains its location and size between invocations, making it easy for the user to position it conveniently during the design session.

5.3.1.4.6 Binding Collection Editor

The binding collection editor allows adding, removing, and modifying bindings. It is usually invoked to work on bindings for a specified control, but it can also be used to work on bindings of all controls in a container (a form in Windows Form, or a window in WPF, typically). It is basically an editor for Binding Bags.

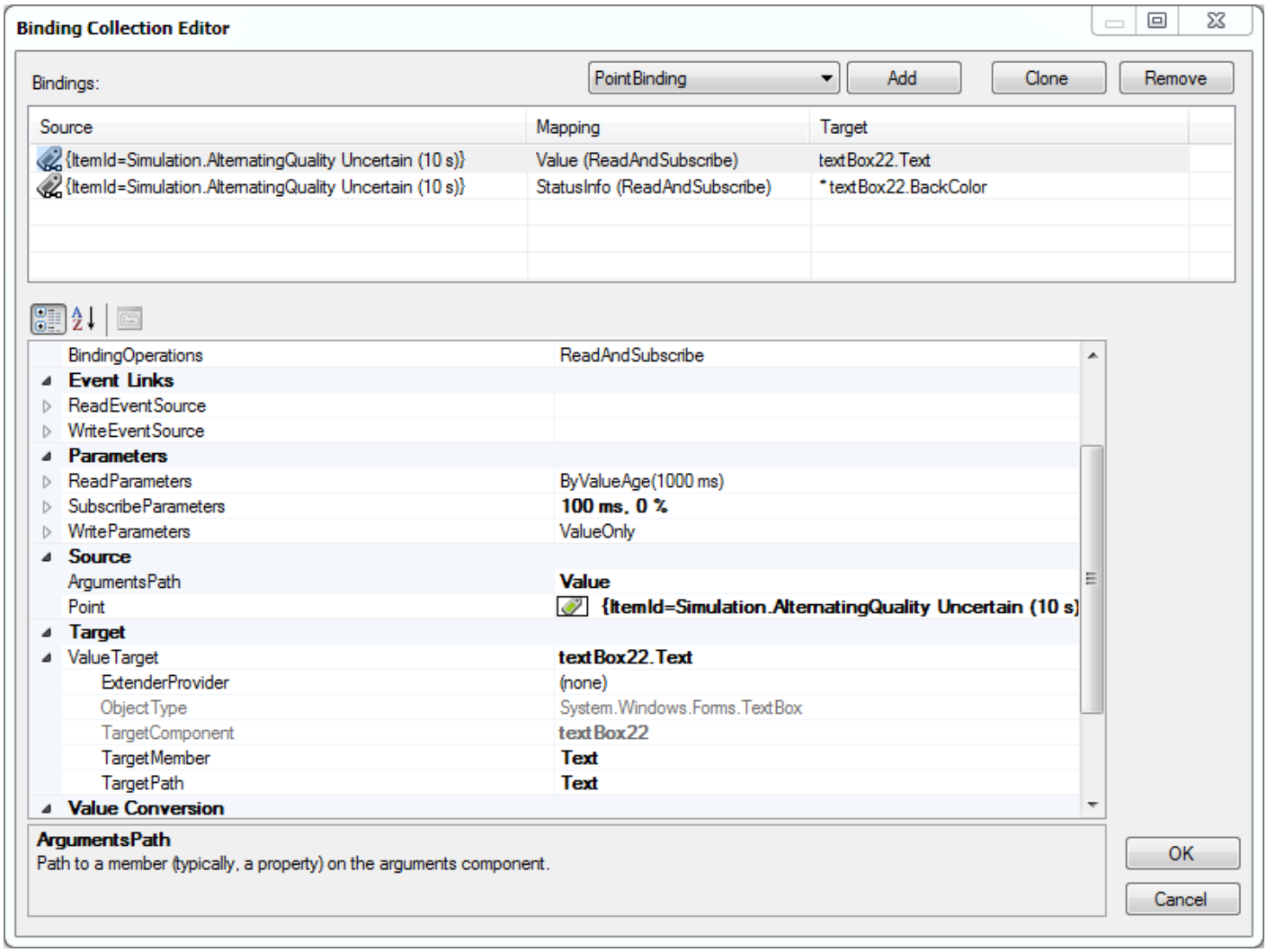
To show the binding collection editor for a specific control, you can either:

a. Right-click on the control, and select "Edit Live Bindings..." from the context menu.
b. Select the control, and then click on "Edit Live Bindings..." command in the Properties window (if commands are not visible in the property grid, right-click on it and check the Commands menu item).
c. Select the control, and in the Properties window, find the BindingBag property (under the Live Binding category). Select this property, and then click the small "..." button on the right side.

Note: If you do not see any of the above commands, the likely reason is that you have not placed the BindingExtender component on the form.

To show the binding collection editor for all controls on a Windows form, select the BindingExtender component, and in the Properties window, find the ComposedBindingBag property. Select this property, and then click the small "..." button on the right side.

When invoked, the binding collection editor looks similar to this:



The “Bindings” list displays all bindings in the collection, and the most important information about them (the source, the binding operations, and the target of the binding). Selecting any row in the “Bindings” list displays the properties of the selected binding in the “Binding properties” property grid. You can then view and modify the detailed properties of the binding in the property grid.

> You can also view and modify the design-time properties of each binding: The name (Name) used in code to identify the object, whether a member variable will be generated for the component (GenerateMember), and the visibility level of the object (Modifiers).

There are also following commands available in the binding collection editor:

Add:

Creates a new binding of the selected type, and adds it to the binding collection. Before pressing Add, you need to select the desired type of binding in the drop-down list left to the Add button.

The newly created binding gets some of its parameters from the templates defined on the binder or the connectivity component. If you create many similar bindings (such as always referring to the same OPC Server), you can save yourself lots of work by pre-filling the template with parameters that you commonly use.

Clone:

Create a duplicate of the selected binding (or bindings). This can be useful when you have made the main binding on the control, usually with ArgumentsPath of Value, and you want to extend it further, e.g. by adding a tooltip, or change the background color. In this case you need to bind to the same source and using the same parameters, and you will be changing just the arguments path and the binding target.

Remove:

Remove the selected binding (or bindings).

OK:

Confirms the changes made to the bindings in the dialog, and closes the dialog.

Cancel:

Dismisses any changes made to the bindings in the dialog, and closes the dialog.

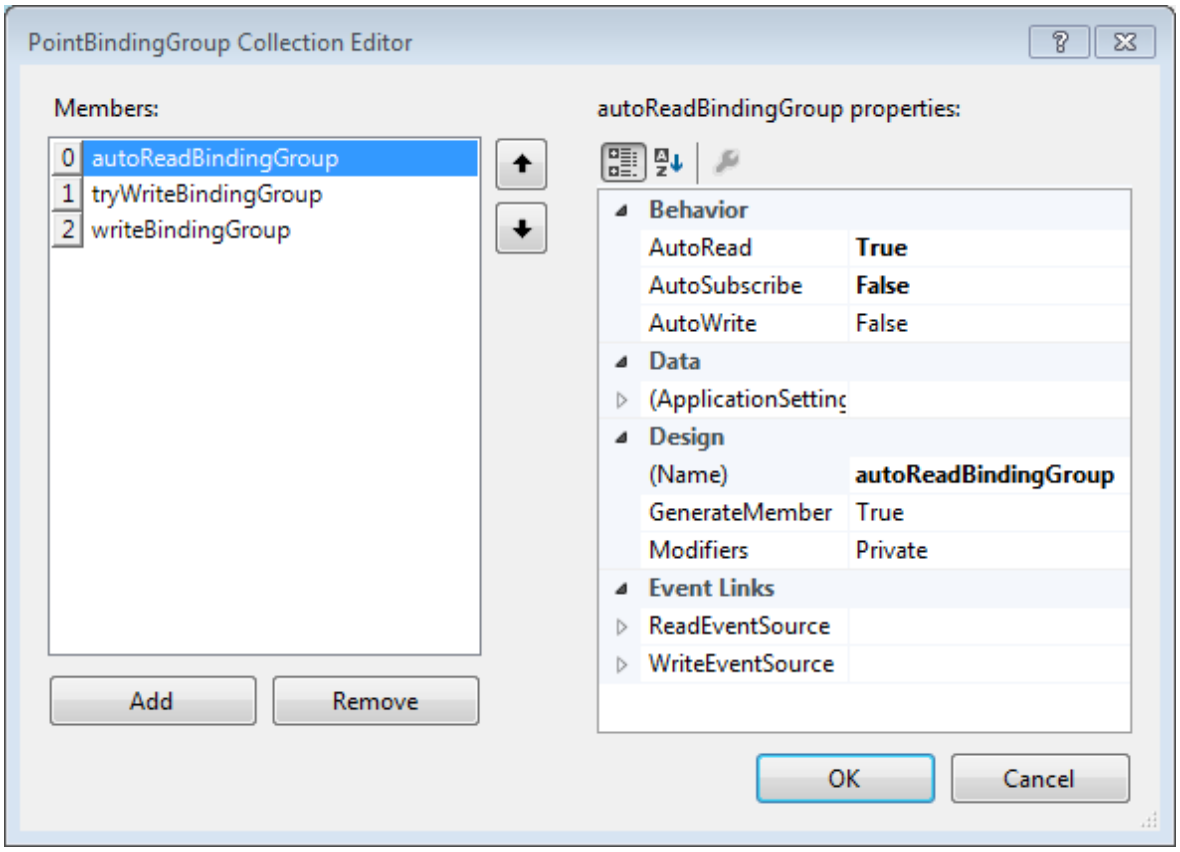
All changes that you make to the bindings in the binding collection editor are temporary, until you press OK. You can revert to the original state by pressing Cancel at any moment.

Note: Emptying the ValueTarget.TargetComponent of any binding will effectively remove the binding from the binding collection when the dialog changes are confirmed. This is because bindings must always be associated with their target.

5.3.1.4.7 Binding Group Collection Editor

The binding group collection editor allows adding, removing, and modifying the Binding Groups. To show the binding group editor, select the PointBinder component placed on your component tray, and in the Properties window (or, in WPF in the property grid next to the Components list), select its BindingGroups property. Then, click the small "..." button on the right side.

When invoked, the binding group collection editor looks similar to this:



Note that the default binding group does not appear in the BindingGroups collection. It is a separate object that you can edit directly, as the DefaultBindingGroup property of the PointBinder component.

5.3.1.4.8 Online Design

The live binding model includes an Online Design feature that allows you to explore and verify most aspects of the configured live bindings without building and running your application. You can see the binding truly live – i.e. while still designing in Visual Studio, the binding operations can be performed. This means that when you configure e.g. a Read or Subscribe operation with your binding, you can pre-view live OPC data on your form, and verify that the binding works as intended.

In This Topic

WPF

Initially, the Online Design mode is turned off. In order to control the Online Design mode, you can either right-click on the BindingExtender component on your component tray, and select “Design Online” or “Design Offline” command from its context menu (Windows Forms only), or you can select the BindingExtender component in the component tray first, and then use the same commands in Properties windows or property grid next to the Components list (provided that commands are enabled in the property grid; if not, right-click on it and check the Commands item).

Because the live binding makes modifications to property values of components in your program, be aware that the online design has the same effect. If you do not want the values resulting from online design to persist in your project, you need to refrain from saving the changes made by online design.

When the design is made online, it can be much more demanding on computer resources, because each value update causes value changes that need to be reflected by the designer, regenerating code and making other changes in the environment. In order to maintain responsiveness of the development environment, the update rate in the online design mode is limited. Even if you set the update rate to a fast value, you may get slower update rates during the design.

WPF

In WPF (as opposed to Windows Forms), if you make a change to binding properties while the Online Design is turned on, the changed are not immediately picked up. You need to turn the Online Design off and on again.

In WPF, screen updates for Online Design are performed on background, with low priority. As a result, it is possible that not all updates always occur, especially if you have many bound controls.

In WPF, we try to prevent the changed property values (from Online Design) be stored back into the XAML, but depending on circumestances, it is not always 100% reliable.

5.3.1.5 Live Binding Details

This chapter describes more advanced workings of the live binding, and some internals that help in understanding it.

5.3.1.5.1 How the Binding Extender Automatically Goes Online and Offline

The BindingExtender component has two configurable event links that allow it to automatically go online or offline:

- OnlineEventSource: Event triggered by this source puts the binding extender into the online mode.
- OfflineEventSource: Event triggered by this source puts the binding extender into the online mode.

Depending on the kind of project, Live Binding uses following rules to determine the default settings of these properties:

- In Windows Forms, when you initially place the BindingExtender component onto a Windows form by dragging it from a Toolbox, its OnlineEventSource is set to the Form's Shown event, and its OfflineEventSource is set to the Form's FormClosing event.
- In WPF, when you initially place the BindingExtender component onto a component tray by adding it from Toolbox Items, its OnlineEventSource is set to the root object's Loaded event, and its OfflineEventSource is set to the root object's Closed event. The root object is normally the window being designed.

The effect of this configuration is such that during runtime, the binding extender is automatically online while the form or window is open, and it also goes offline when the form or window closes.

This default setting makes it easy to create forms that are "live" (online) without further coding or configuration. If you, however, want to have control over the process, you can clear these event sources, and use the BindingExtender's Online property to set or reset the online state.

5.3.1.5.2 What Happens When the Binding Extender Is Set Online or Offline

When the binding extender is set to online (i.e. when its Online property is set to 'true'), following steps are performed internally:

1. Each configured binding makes its “links”, i.e. hooks the handlers for its configured event sources. For example, the PointBinding-s have ReadEventSource and WriteEventSource, so if they are configured, the binding will link the events with handlers that perform Read or Write operation on the point binding.
2. All binding groups perform their configured automatic operations. For PointBindingGroup-s, this means that:
    a. If AutoRead property is set, the Read operation is executed for all PointBinding-s that have Read included in their BindingOperations property (this includes “Get” of OPC-DA properties).
    b. If AutoSubscribe property is set, the Subscribe(true) operation is executed for all PointBinding-s that have Subscribe included in their BindingOperations property.
3. All binding groups make their “links”, i.e. hook the handlers for their configured event sources. For PointBindingGroup-s, this means that the events for ReadEventSource and WriteEventSource will be linked with handlers that perform Read or Write operation on the binding group.
4. The OnlineChanged event is raised on the BindingExtender.

When setting the binding extender to offline, the above steps are (roughly said) reversed. There are, however, differences in Step 2: For PointBindingGroups-s, the Read operations are not executed; instead, if AutoWrite property is set, the Write operation is executed for all PointBinding-s that have Write included in their BindingOperations property. Also, if AutoSubscribe property is set, the Subscribe(false) operation is executed (i.e. unsubscribe, instead of subscribe).

5.3.1.5.3 What Happens When a Binding Operation Method Is Called

When any operation is invoked either automatically (with help of configures event sources), or programmatically, the live binding system first determines the set of bindings involved in the operation, and then executes the requested operation.

For operations invoked directly on the binding, the operation is performed on this single binding.

For operations invoked on some binding group, the group first determines the bindings that belong to it. It then performs the operation together on all bindings in this group.

5.3.1.5.4 Programmatic Access to Live Binding

Below are the most commonly needed programmatic actions:

- To set the binding extender (all bindings on it) to online or offline from your code, set its Online property to true or false. The binding extender is a component placed on your form, so you can refer to it by its name.
- To execute the Read or Write operation on concrete binding, call the PointBinding’s ExecuteRead or ExecuteWrite method. Note that although the individual binding do not appear on the design surface (of the form) or on the component tray, they are still components of your form, and you can refer to individual bindings by their name (as of now, you cannot change the name from the binding collection editor, so you need to look it up in the designer-generated code, and either use it as is, or rename manually).
- To execute any of the operations on a concrete binding group (for all binding in that group that have that operation allowed), call a corresponding method on the PointBindingGroup object. The binding groups do not appear on thecomponent tray (of the form or window), but they are still components that belong to your form or window, and you can refer to them by their name. The name of the binding group can be viewed or modified in the Binding Group Collection Editor. The method names are PointBindingGroup.ExecuteRead, ExecuteSubscribe and ExecuteWrite. The ExecuteSubscribe has a bool argument that determines whether you want to subscribe or

unsubscribe.

You can also override some of the error handling in live binding (see **Error Handling in Live Binding (Section 5.3.1.5.5)**).

5.3.1.5.5 Error Handling in Live Binding

There are several types of errors that can occur in relation to live binding, and they are treated differently.

I. When you configure the live binding in the designer, there may be one or more *binding errors*, caused by improper configuration. When this happens, the designer displays a form that lists all the binding errors detected.

   You can press the "Details..." button to view the details of the binding errors listed on the form.

   The binding errors are caused by BindingException-s during the design, and are generally "benign", meaning that you can continue your design work, but you'd better fix the errors, otherwise they can cause more serious errors during runtime.

   Examples of such binding errors are:

   - A binding where the value target has no TargetComponent specified.
   - Member linking error: The specified target member does not exist.
   - Member linking error: Cannot determine the data type of the target member.

   In the runtime, binding errors are reported by the BindingExtender.BindingErrors event. Unless you hook your own handler to this event, they are silently ignored.

II. When the binding extender is set online or offline, there can also be *event linking errors*. They are similar to binding errors in that they are caused by improper configuration – this time, an improper configuration of some of the event sources used to automatically trigger the operations.

   Unless you decide to handle them, event linking errors are silently ignored both during design-time and in runtime. For event sources defined on the binding extender, any event linking error is signaled by the BindingExtender.EventLinkingFailure event. For event sources defined on the binder component (PointBinder, usually), its binding groups or its bindings, any event linking error is signaled by the BinderBase.EventLinkingFailure event.

III. When a target could not be updated with a new value, an *update failure* occurs. Unless you decide to handle them, update failures are silently ignored both during design-time and in runtime. Each failure is signaled by the PointBinder.UpdateFailure event. The event notification contains an Exception indicating the cause of the update failure, and a reference to the related binding. You can hook your event handler to this event if you need some action be performed in such case.

   There may be various reasons for update failures; the most common are *target access errors*. They happen when the live binding encounters a failure accessing your objects (binding targets). For example, setting or getting a property on your object may throw an exception, and such exception will be caught. Target access error is represented by an instance of TargetAccessException in an update failure.

Errors that originate in the binding source, such as OPC item or OPC property, are considered a completely separate issue. For example, the OPC server may be shut down, or the item ID might have changed name and is no longer accessible, or its value just has a bad "quality". Such errors are treated as just another data to be transferred by the binding. There are arguments (see Arguments and Results) such as Exception or ErrorId that allow you to set up members on your objects that will receive error information in case of errors that originate in the binding source.

5.3.1.5.6 Usage Guidelines (Live Binding)

Configuration and Programming

Below are some pieces of advice that may help you configure and use the live binding properly:

- If operations need to be performed together, they must be assigned to the same binding group and invoked on the group. Let's describe the wrong approach first: When the user presses a button, you want to grab values from several controls on the form, and write them to various OPC items. You configure the WriteEventSource on each binding to the Click event of the Button. Well, while this "kind of" works, it is very ineffective approach, because each binding will be handled separately, and the OPC writes will be executed one by one, sequentially. The proper approach is to create a binding group for this purpose, set all bindings involved in the "write" to be members of this binding group, and then set the WriteEventSource just once, on the binding group itself. This way, the binding group will attempt to write all values into their corresponding OPC items using a single call with multiple items in it.
- Remember to always set the BindingExtender back to offline mode. This guideline only applies if you are controlling the BindingExtender-s Online property in a way that differs from the default described in **How the Binding Extender Automatically Goes Online and Offline (Section 5.3.1.5.1)**. In such case, you need to be aware that while the BindingExtender is online mode, there are event handlers set up that may prevent the normal application shutdown. Basically, by setting the Online property of BindingExtender to 'true', you indicate that for whatever reason, you are interested in being connected to the binding targets – and you have to explicitly set it back to false in order to indicate that this is no longer needed.

Combination with Imperative Programming Model

For more advanced tasks, you may want to combine the Live Binding model with the **Imperative Programming Model (Section 5.1)**. Such combination is perfectly possible and supported.

When doing so, consider the fact that Live Binding uses its own, "isolated" client objects (such as EasyDAClient or EasyUAClient). When you create instances of the client objects in your own code, your code will have separate connections (sessions) from those used in the Live Binding. This may be fine, but can cause issues in some cases (e.g. when the target OPC Server is an embedded server with limited number of concurrent sessions).

In order to prevent this from happening, you can "borrow" the client object from the Live Binding, and use it in your code. To do so, call the GetClient methods on the DAConnectivity or UAConnectivity component that you already have for the Live Binding. The method returns an IEasyDAClient (or IEasyUAClient) interface, and you can use it to perform any necessary operations through imperative programming.

5.3.1.6 Typical Binding Scenarios

The live binding model is very powerful in terms of what can be achieved without any manual coding. In order to configure it right, however, you need to know how, and understand a little about the internal workings of live binding. The scenarios described in this chapter are designed just for that – to give instructions that will guide you through configuring the live binding for typical usages.

Each scenario contains a description of its purpose, and step by step instructions explaining how to configure it.

You should be able to use these scenarios both as a “catalog”, a reference to choose from and follow the steps, and as a learning tool: If you try several scenarios, you will get a feeling of how to configure the live binding for proper behavior.

Many of the scenarios listed here are also demonstrated live in the “Live Binding Demo” applications that are installed with the product. For OPC Data Access (OPC-DA), its source code is contained in the WindowsForms\LiveBindingDemo project. For OPC Unified Architecture (OPC-UA), its source code is contained in the WindowsForms\UALiveBindingDemo project.

5.3.1.6.1 Automatic Subscription (With the Form or Window)

This is the default behavior. The binding is online while the form or window is open. You can bind to all types of components and controls. You can bind to any property or field. You can also bind to some methods and events.

How to configure this feature for OPC-DA:

If you have not done so yet, place the DAConnectivity component onto the component tray of the form or window (this is a common step for all binding tasks).

Use the "Bind to Point" command on the target control, and select the source OPC server and OPC item.

That's it. You can verify the result immediately, without recompiling your application, by using the "Design Online" command on the BindingExtender component.

How to configure this feature for OPC-UA:

If you have not done so yet, place the UAConnectivity component onto the component tray of the form or window (this is a common step for all binding tasks).

Use the "Bind to Point" command on the target control, and select the source OPC-UA server and OPC node.

That's it. You can verify the result immediately, without recompiling your application, by using the "Design Online" command on the BindingExtender component.

5.3.1.6.2 Automatic Read (On Form or Window Load)

OPC-DA item values (or OPC-UA node attributes) can be also read once, when the form or window loads.

How to configure this feature for OPC-DA:

On a PointBinder, edit the BindingGroups collection, "Add" a new binding group, set its AutoRead property to True, and make sure that its AutoSubscribe property is set to False.

On a TextBox (or other control), use "Bind to Point" command and select the OPC item you want to read. Then, use the "Edit Live Bindings" command, and set the BindingGroup property of the item binding to the binding group you have created earlier. The Read operation is automatically executed for all item bindings in this group when the form loads.

How to configure this feature for OPC-UA:

On a PointBinder, edit the BindingGroups collection, "Add" a new binding group, set its AutoRead property to True, and make sure that its AutoSubscribe property is set to False.

On a TextBox, use "Bind to Point" command and select the OPC-UA node you want to read. Then, use the "Edit Live Bindings" command, and set the BindingGroup property of the data binding to the binding group you have created earlier. The Read operation is automatically executed for all data bindings in this group when the form loads.

5.3.1.6.3 Read On Custom Event

For example, clicking the Read button will execute the operation and read the OPC item value into the text box.

How to configure this feature for OPC-DA:

On a TextBox (or other control), use "Bind to Point" command and select the OPC item you want to read (alternatively, use "Edit Live Bindings" command, "Add" a PointBinding, and configure its point's ServerDescriptor and ItemDescriptor).

Then, use the "Edit Live Bindings" command, and for BindingOperations property, leave just Read checked (and uncheck the Subscribe). In order to have the Read invoked when the button is pressed, set the ReadEventSource.SourceComponent to the Button control; the SourceMember of the ReadEventSource will be automatically set to the Click event. This causes the Click on the Button execute the Read operation, i.e. obtain the OPC item value and store it into the TextBox.

Note: In order to read multiple items at once on a custom event, it is suggested that you set up a binding group under the PointBinder, assign the bindings to the group, and link the ReadEventSource just once, on the binding group.

How to configure this feature for OPC-UA:

On a TextBox, use "Bind to Point" command and select the OPC-UA node you want to read (alternatively, use "Edit Live Bindings" command, "Add" a PointBinding, and configure its EndpointDescriptor and NodeDescriptor).

Then, use the "Edit Live Bindings" command, and for BindingOperations property, leave just Read checked (and uncheck the Subscribe). In order to have the Read invoked when the button is pressed, set the ReadEventSource.SourceComponent to the Button control; the SourceMember of the ReadEventSource will be automatically set to the Click event. This causes the Click on the Button execute the Read operation, i.e. obtain the OPC-UA node value and store it into the TextBox.

Note: In order to read multiple nodes at once on a custom event, it is suggested that you set up a binding group under the PointBinder, assign the bindings to the group, and link the ReadEventSource just once, on the binding group.

5.3.1.6.4 String Formatting

All standard .NET formatting features can be used to specify how the OPC values should be displayed.

How to configure this feature:

Bind the OPC-DA item (or OPC-UA node attribute) to the TextBox or similar control as usually.

Then, use the "Edit Live Bindings" command on the control, and set the StringFormat property on the item binding as needed, using all standard .NET formatting capabilities. {0} in the formatting string represents the value to be formatted. For example, {0:F3} means a floating point value formatted with 3 decimal places.

5.3.1.6.5 Change Color According to Status

Values can be converted in both directions by IDataConverter objects. For example, StatusToColorConverter provides a color value based on operation status, such as a yellow background when quality is Uncertain.

How to configure this feature:

Place the StatusToColorConverter component (from the toolbox items) onto the component tray. Bind the OPC-DA item (or OPC-UA node attribute) to your control as usually.

Then, use the "Edit Live Bindings" command, "Clone" the existing binding, and change the cloned binding as follows: Set the ArgumentsPath to StatusInfo. Set the ValueTarget.TargetMember to the BackColor (in Windows Forms), Background (in WPF), or other suitable property of your control. For the Converter property, select the StatusToColorConverter you have placed to the form earlier. This will cause the status information from the OPC item be converted to one of

configurable colors, and change the background color of the control appropriately.

5.3.1.6.6 ToolTip and Other Extenders

In general, Live Binding supports extender providers and extender properties both in Windows Forms and in WPF. The procedure described below for WIndows Forms, however, is not usable with WPF, because in WPF. tooltips are not implemented using an extender provider, but are instead a property right on the control level.

In This Topic

Windows Forms

You can bind to properties made available by extender providers, such as ToolTip. Hovering over the control associated with the ToolTip then allows the user to see more details about the value.

How to configure this feature for OPC-DA:

Drag the ToolTip extender (standard Microsoft component), or other extender provider you need, from the toolbox to the form. Configure the binding in a usual way.

Then, use the "Edit Live Bindings" command, "Clone" the existing binding, and change the cloned binding as follows: Set the ValueTarget.ExtenderProvider to the ToolTip (extender provider) component on your form, and set the ValueTarget.TargetMember to the Tooltip member. Set the ArgumentsPath to Vtq (or Timestamp, Quality, or whatever you'll want to be displayed in the tooltip). This will cause the additional information from the OPC item be displayed by the ToolTip extender provider for the control.

How to configure this feature for OPC-UA:

Drag the ToolTip extender (standard Microsoft component), or other extender provider you need, from the toolbox to the form. Configure the binding in a usual way.

Then, use the "Edit Live Bindings" command, "Clone" the existing binding, and change the cloned binding as follows: Set the ValueTarget.ExtenderProvider to the ToolTip (extender provider) component on your form, and set the ValueTarget.TargetMember to the Tooltip member. Set the ArgumentsPath to AttributeData (or SourceTimestamp, ServerTimestamp, StatusCode, or whatever you'll want to be displayed in the tooltip). This will cause the additional information from the OPC-UA be displayed by the ToolTip extender provider for the control.

WPF

How to configure this feature:

Configure the binding in a usual way.

Then, use the "Edit Live Bindings" command, "Clone" the existing binding, and change the cloned binding as follows: Set the ValueTarget to the Tooltip member. Set the ArgumentsPath to AttributeData (or SourceTimestamp, ServerTimestamp, StatusCode, or whatever you'll want to be displayed in the tooltip). Set the StringFormat property on the data binding to {0} or as needed. Note that the displayed tooltip is not dynamically updated when StringFormat is not set.

5.3.1.6.7 Display Errors with ErrorProvider

Windows Forms

The standard ErrorProvider extender is useful for displaying OPC operation error icons, with tooltips.

How to configure this feature:

Drag the ErrorProvider extender (standard Microsoft component) from the toolbox to the form. Configure the binding in a usual way.

Then, use the "Edit Live Bindings" command, "Clone" the existing binding, and change the cloned binding as follows: Set the ArgumentsPath to ErrorMessage. Set the ValueTarget.ExtenderProvider to the ErrorProvider component on your form, and set the ValueTarget.TargetMember to the Error member. All in all, this will cause the error message from the OPC-DA item (or OPC-UA node attribute) be stored into the error text displayed by the ErrorProvider for the control.

Note: If you configure this with explicit Read or Write operations (not with Subscribe), you need to set up a binding group, assign related bindings to the same group, and have the operation invoked on a binding group.

WPF

Configure the binding in a usual way.

Place the ErrorProvider extender (this is not a standard Microsoft component) from the toolbox items onto the Component Tray. Configure the binding in a usual way.

Then, use the "Edit Live Bindings" command, "Clone" the existing binding, and change the cloned binding as follows: Set the ArgumentsPath to ErrorMessage. Set the ValueTarget.ExtenderProvider to the ErrorProvider component on your form, and set the ValueTarget.TargetMember to the Error member. All in all, this will cause the error message from the OPC-UA be stored into the error text displayed by the ErrorProvider for the control.

Note: If you configure this with explicit Read or Write operations (not with Subscription), you need to set up a binding group, assign related bindings to the same group, and have the operation invoked on a binding group.

Tip: The standard Validation.ErrorTemplate simply displays a red rectangle around control, but without the error text. In order to display the error text as well, you need to use styles - define your own Validation.ErrorTemplate in XAML <Window.Resource>. Then, set Validation.ErrorTemplate="{StaticResource ErrorTemplate}" for the control.

ErrorTemplate example (indicates the error by a red circle with an exclamation mark, and the error text):

```
<Window.Resources>
        <Style x:Key="BlinkedEllipse" TargetType="{x:Type Ellipse}">
            <Style.Triggers>
                <EventTrigger RoutedEvent="Ellipse.Loaded">
                    <EventTrigger.Actions>
                        <BeginStoryboard>
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="Opacity"
From="1" To="0" Duration="0:0:0.5" AutoReverse="True" RepeatBehavior="3x"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger.Actions>
                </EventTrigger>
            </Style.Triggers>
        </Style>
        <ControlTemplate x:Key="ErrorTemplate">
```

```
            <DockPanel LastChildFill="True">
                <Grid Width="16" Height="16" DockPanel.Dock="Right">
                    <Ellipse Width="16" Height="16" HorizontalAlignment="Center"
VerticalAlignment="Center" Stroke="Black" StrokeThickness="1" ToolTip="{Binding
AdornedElement.(Validation.Errors)[0].ErrorContent, ElementName=ErrorAdorner}" Style="
{StaticResource BlinkedEllipse}">
                        <Ellipse.Fill>
                            <LinearGradientBrush StartPoint="0,0" EndPoint="1,1" >
                                <GradientStop Color="White" Offset="0" />
                                <GradientStop Color="Red" Offset="1.0" />
                            </LinearGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <TextBlock Foreground="White" FontWeight="Heavy" FontSize="12"
HorizontalAlignment="Center"
                                            VerticalAlignment="Center"
TextAlignment="Center"
                        ToolTip="{Binding AdornedElement.(Validation.Errors)
[0].ErrorContent, ElementName=ErrorAdorner}"
                                            Text="!"/>
                </Grid>
                <AdornedElementPlaceholder x:Name="ErrorAdorner" />
            </DockPanel>
        </ControlTemplate>
</Window.Resources>
```

Tip: If ErrorProvider is used in binding for control that is inside TabItem or GroupBox, then <AdornerDecorator> should be used, see:

```
<GroupBox>
    <AdornerDecorator>
        <Grid>
            <TextBox> <!-- TextBox with LiveBinding using ErrorProvider !-->
            </TextBox>
        </Grid>
    </AdornerDecorator>
</GroupBox>
```

5.3.1.6.8 Various Kinds of Binding

You normally bind to the OPC value itself, but you can select from many other binding kinds, such as ErrorId, ErrorMessage, StatusInfo, (for OPC-DA) Timestamp, Quality, Vtq, or (for OPC-UA) ServerTimestamp, SourceTimestamp, AttributeData, StatusCode, etc.

How to configure this feature:

Bind the OPC item to your control as usually.

Then, use the "Edit Live Bindings" command on the control, and in the ArgumentsPath property, select the kind of binding you want.

5.3.1.6.9 Write Single Value on Custom Event

A value of control property is written to the associated OPC item when a specified event is triggered. For example, you can link the Write to the button click.

How to configure this feature:

On a TextBox (or similar control), use "Bind to Point" command and select the OPC-DA item (or OPC-UA node) you want to write to.

Then, use the "Edit Live Bindings" command, and for the BindingOperations options, select only "Write". In order to have the Write invoked when the button is pressed, set the WriteEventSource.SourceComponent to the Button control; the SourceMember of the WriteEventSource will be automatically set to the Click event. This causes the Click on the Button execute the Write operation, i.e. obtain the value from the control and write it into the OPC item.

5.3.1.6.10 Write Group of Values on Custom Event

In this scenario, multiple property values are gathered and written to their associated OPC items in a single operation, linked to a specified event – for example, a button click.

How to configure this feature for OPC-DA:

On a PointBinder, edit the BindingGroups collection, "Add" a new binding group, set its AutoSubscribe property to False, and set its WriteEventSource.SourceComponent to the button that should invoke the Write.

On each of the value controls involved, use "Bind to Point" command and select the OPC item you want to write. Then, use the "Edit Live Bindings" command, and for the BindingOperations options, select only "Write". Also, set the BindingGroup property of the item binding to the binding group you have created earlier. The Write operation will be executed for all item bindings in this group when the button is clicked.

How to configure this feature for OPC-UA:

On a PointBinder, edit the BindingGroups collection, "Add" a new binding group, set its AutoSubscribe property to False, and set its WriteEventSource.SourceComponent to the button that should invoke the Write.

On each of the value controls involved, use "Bind to Point" command and select the OPC-UA node you want to write. Then, use the "Edit Live Bindings" command, and for the BindingOperations options, select only "Write". Also, set the BindingGroup property of the data binding to the binding group you have created earlier. The Write operation will be executed for all data bindings in this group when the button is clicked.

5.3.1.6.11 Subscribe & Write

A control can reflect OPC value changes, and also write the updated values to the OPC server when modified by the user. For example, you can bind a TrackBar and a NumericUpDown controls to the same OPC item, and they will influence each other through OPC.

In This Topic

How to configure this feature:

On the target control, use "Bind to Point" command and select the OPC-DA item (or OPC-UA node) you want to bind to.

Then, use the "Edit Live Bindings" command, and for BindingOperations property, add a check next to the Write operation. In order to have the Write invoked when the control changes, set the WriteEventSource.SourceComponent to your control; the SourceMember of the WriteEventSource will be automatically determined, but you can change it, if you want to link to a different event.

Note: In OPC-DA, some controls will require you to set the ItemDescriptor.RequestedDataType so that the OPC value type matches the type of the target property.

Note: In OPC-UA, some controls or OPC-UA servers will require you to set the ValueTypeCode (under the WriteParameters property) so that the OPC-UA value type matches the type of the target property. It is also recommended to set the ValueTypeCode for performance reasons.

WPF

In WPF, there is no precise equivalent of the NumericUpDown control used in our Windows Forms example. The WPF Live Binding Demo uses styles to achieve a comparable result.

5.3.1.6.12 Display Write Errors

When the Write fails, you want the user be notified. One way to do this is with the standard ErrorProvider control. For example, if you attempt to write a value that is too large, an error icon appears, and the error message can be seen in the associated tooltip.

How to configure this feature:

On a PointBinder, edit the BindingGroups collection, "Add" a new binding group, set its AutoSubscribe property to False, and set its WriteEventSource.SourceComponent to the button that should invoke the Write.

Place the ErrorProvider extender from the toolbox items onto the component tray.

Configure the binding in a usual way. Use the "Edit Live Bindings" command, and for the BindingOperations options, select only "Write", and set the BindingGroup property of the item binding to the binding group you have created earlier. Then, "Clone" the existing binding, and change the cloned binding as follows: Set the ArgumentsPath to ErrorMessage. Set the ValueTarget.ExtenderProvider to the ErrorProvider component on your form, and set the ValueTarget.TargetMember to the Error member. This will cause the error message from the OPC item be stored into the error text displayed by the ErrorProvider for the control.

5.3.1.6.13 Get an OPC Property on Custom Event

You can bind to OPC-DA properties as well. For example, clicking on the Get button can obtain the value of the HighEU property.

Note: In OPC-UA Unified Architecture, there is no need for this – just use Read of any attribute of a node.

How to configure this feature:

On a TextBox, use "Bind to Point" command and select the OPC item and property you want to get (alternatively, use "Edit Live Bindings" command, "Add" a PointBinding, and configure its ServerDescriptor and PropertyDescriptor).

In order to have the Get invoked when the button is pressed, set the ReadEventSource.SourceComponent to the Button control; the SourceMember of the ReadEventSource will be automatically set to the Click event. This causes the Click on the Button execute the Get operation, i.e. obtain the OPC property value and store it into the TextBox.

5.3.1.6.14 Automatically Get an OPC Property (On Form Load)

The OPC-DA property value can also be obtained automatically when the form loads.

Note: In OPC-UA Unified Architecture, there is no need for this – just use Read.

How to configure this feature:

On a PointBinder, edit the BindingGroups collection, "Add" a new binding group, and set its AutoRead property to True.

On a TextBox, use "Bind to Point" command and select the OPC item and property you want to get. Then, use the "Edit Live Bindings" command, and set the BindingGroup property of the property binding to the binding group you have created earlier. The Get operation is automatically executed for all property bindings in this group when the form loads.

5.4 Reactive Programming Model

The reactive programming development model merges the world of Microsoft Reactive Extensions (Rx) for .NET with OPC. The Reactive Extensions (http://msdn.microsoft.com/en-us/data/gg577609.aspx) is a library to compose asynchronous and event-based programs using observable collections (data streams) and LINQ-style query operators.

OPC Data Client provides OPC-specific classes that implement the Rx interfaces. For example, you can easily create an Rx observable sequence with OPC-DA. OPC XML-DA or OPC-UA item changes, and process it further. Similarly, you can create an Rx observable sequence with OPC-A&E event notifications.

For complex event and stream processing that builds on top of the reactive programming model, see also

- **StreamInsight Option (Section 12.2)**
- **Installed Examples - Console - SimpleTrillApplication (Section 13.1.4.1.9)** (Microsoft Trill: https://github.com/microsoft/Trill)

The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

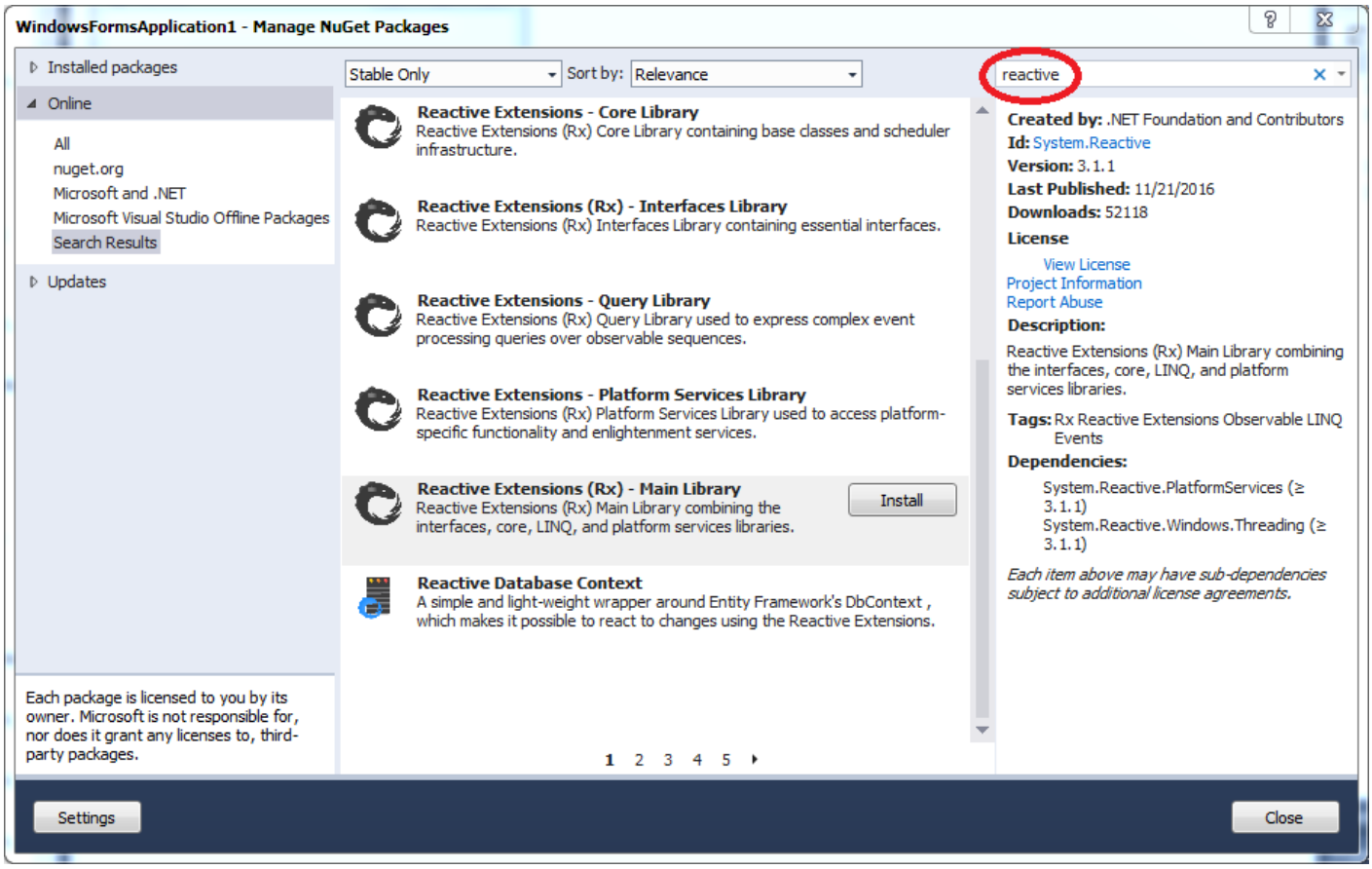
This functionality is not yet available for OPC UA PubSub.

5.4.1 Reactive Programming Model for OPC Data (Classic and UA)

Microsoft Reactive Extensions (Rx) and reactive programming in general is a big subject and cannot be covered in any detail here. We will only describe the specifics that OPC Data Client brings to Rx, by bridging it with the world of OPC.

The power of reactive extensions is in their high level of abstraction, asynchronous nature, and ability of composition.

Microsoft provides Reactive Extensions (Rx) as a NuGet package. Reference the corresponding package in your project, in order to get access to the Rx types. In Visual Studio, use the “Tools -> Library Package Manager” command, and search for “Reactive Extensions – Main Library”, as on the following picture.



5.4.1.1 OPC Reactive Extensions (Rx/OPC)

OPC Reactive Extensions (Rx/OPC) are OPC-specific classes that implement the Rx interfaces. With Rx/OPC, you can e.g.:

- Create observable collections (data streams) that provide OPC data.
- Consume (observe) data streams, and process them using OPC.
- Combine data flows to create more complex operations.

For OPC Data Access, OPC XML-DA and OPC-UA data, Rx/OPC gives you the ability to work with flows representing OPC (monitored) item changes, or flows that represents values to be written into an OPC item (node).

In the example below, your application is required to continuously monitor the value of an OPC item, and when it is available (there is no failure obtaining the value), make some computations with it (we will multiply the value by 1000), and write the results into a different OPC item. This logic can be expressed by following code:

Reactive transfer of OPC DA item values

```
DAItemChangedObservable.Create<int>(
        "", "OPCLabs.KitServer.2", "Simulation.Incrementing (1 s)", 100)
    .Where(e => e.Exception == null)
    .Select(e => e.Vtq.Value * 1000)
    .Subscribe(DAWriteItemValueObserver.Create<int>(
```

```
        "", "OPCLabs.KitServer.2", "Simulation.Register_I4"));
```

Let's dissect what this example does:

1. It creates an observable sequence for significant changes in OPC-DA item "Simulation.Incrementing (1 s)".
2. The "Where" clause filters (from the observable sequence) only the changes that have a null Exception, i.e. those that carry a valid DAVtq object (value/timestamp/quality).
3. The "Select" clause takes the actual value from the Vtq property (it is of type DAVtq<int>), and returns the value multiplied by 1000.
4. An observer that writes incoming vales into the "Simulation.Register_I4" OPC-DA item is created.
5. The observer is subscribed to the transformed (processed) observable sequence.

As you can see, the code that does this kind of OPC data processing is very concise – all the extra "plumbing" needed in imperative programming model is gone, and only the meaningful pieces remain. Programs written in this way clearly express their intent, and the logic that handles certain functionality is concentrated in one place and not spread around various classes and methods.

5.4.1.2 OPC Data Observables

The DAItemChangedObservable<TValue> class is an observable for changes in OPC Data Access item or multiple items of type TValue. The UADataChangeNotificationObservable<TValue> class is an observable for changes in OPC-UA monitored item or multiple items of type TValue. It represents a data stream with information about significant data changes in the subscribed items.

Each significant change is represented by an OnNext message of type EasyDAItemChangedEventArgs<T> (for OPC Classic) or EasyUADataChangeNotificationEventArgs<T> (for OPC-UA). This message is used both for successes, when the Exception property is a null reference and the Vtq property (for OPC Classic) or the AttributeData property (for OPC-UA) contains valid data, and for failures, when the Exception property is non-null and contains the failure information.

The OnCompleted and OnError messages (methods of the IObserver) are never sent (not even in case of error related to the OPC item), thus the data stream is not terminated. If your application requires, you can process the data stream further, and filter it or split it by success/failure as needed.

For OPC Classic, you can create instances of DAItemChangedObservable<TValue> either by using its constructor, or with use of a static DAItemChangedObservable class with several overloads of the Create method. The static DAItemChangedObservable.Create methods use the default underlying EasyDAClient object for OPC reactive extensions. If you need to set some parameters in the client object, you can use the ClientSelector property to specify them.

For OPC-UA, you can create instances of UADataChangeNotificationObservable<TValue> either by using its constructor, or with use of a static UADataChangeNotificationObservable class with several overloads of the Create method. The static UADataChangeNotificationObservable.Create methods use the default underlying EasyUAClient object for OPC reactive extensions. If you need to set some parameters in the client object, you can use the ClientSelector property to specify them.

This approach allows the code be expressed only in terms of pure OPC logic, and be not tied to the actual way it is implemented.

The following code fragment creates the observable using one of the Create methods:

Creating OPC DA observable

```
DAItemChangedObservable<double> ramp =
     DAItemChangedObservable.Create<double>(
         "", "OPCLabs.KitServer.2", "Demo.Ramp", 1000);
```

It is recommended that you create the instances using the DAItemChangedObservable.Create (or UADataChangeNotificationObservable.Create) methods unless you have special needs.

5.4.1.3 OPC Data Observers

The DAWriteItemValueObserver<TValue> is an observer that writes incoming values into an OPC Data Access item. The UAWriteValueObserver<TValue> is an observer that writes incoming values into an OPC-UA node (more precisely, into an attribute of a node).

Each OnNext method call on its IObserver<TValue> interface writes the value in the argument into the OPC item or node attribute associated with the observer. OnCompleted and OnError methods do nothing.

For OPC Classic, you can create instances of DAWriteItemValueObserver<TValue> either by using its constructor, or with use of a static DAWriteItemValueObserver class with several overloads of the Create method. The static DAWriteItemValueObserver.Create methods use the default underlying EasyDAClient object for OPC reactive extensions. If you need to set some parameters in the client object, you can use the ClientSelector property to specify them.

For OPC-UA, you can create instances of UAWriteValueObserver<TValue> either by using its constructor, or with use of a static UAWriteValueObserver class with several overloads of the Create method. The static UAWriteValueObserver.Create methods use the default underlying EasyUAClient object for OPC reactive extensions. If you need to set some parameters in the client object, you can use the ClientSelector property to specify them.

This approach allows the code be expressed only in terms of pure OPC logic, and be not tied to the actual way it is implemented.

The following code fragment creates the observer using one of the Create methods:

Creating OPC DA write observer

```
DAWriteItemValueObserver<int> observer =
     DAWriteItemValueObserver.Create<int>(
         "", "OPCLabs.KitServer.2", "Simulation.Register_I4");
```

It is recommended that you create the instances using the DAWriteItemValueObserver.Create (or UAWriteValueObserver.Create) methods unless you have special needs.

5.4.2 Reactive Programming Model for OPC Alarms and Events

Microsoft Reactive Extensions (Rx) and reactive programming in general is a big subject and cannot be covered in any detail here. We will only describe the specifics that OPC Data Client brings to Rx, by bridging it with the world of OPC.

The power of reactive extensions is in their high level of abstraction, asynchronous nature, and ability of composition.

5.4.2.1 OPC Reactive Extensions (Rx/OPC)

OPC Reactive Extensions (Rx/OPC) are OPC-specific classes that implement the Rx interfaces. With Rx/OPC, you can e.g.:

- Create observable collections (data streams) that provide OPC events.
- Consume (observe) data streams, and process them using OPC.
- Combine data flows to create more complex operations.

For OPC Alarms & Events, Rx/OPC gives you the ability to work with flows representing OPC event notifications, or flows that represent OPC event conditions to be acknowledged.

5.4.2.2 OPC-A&E Observables

The AENotificationObservable class is an observable for notifications about OPC events. It represents a data stream with information about subscribed events.

Each significant change is represented by an OnNext message of type EasyAENotificationEventArgs. This message is used both for successes, when the Exception property is a null reference, and for failures, when the Exception property is non-null and contains the failure information.

The OnCompleted and OnError messages (methods of the IObserver) are never sent (not even in case of error related to the OPC server communication), thus the data stream is not terminated. If your application requires, you can process the data stream further, and filter it or split it by success/failure as needed.

You can create instances of AENotificationObservable either by using its constructor, or with use of several overloads of the static Create method. The static AENotificationObservable.Create methods use the default underlying EasyAEClient object for OPC reactive extensions. If you need to set some parameters in the client object, you can use the ClientSelector property to specify them. This allows the code be expressed only in terms of pure OPC logic, and be not tied to the actual way it is implemented.

It is recommended that you create the instances using the AENotificationObservable.Create methods unless you have special needs.

5.4.2.3 OPC-A&E Observers

The AEAcknowledgeConditionObserver is an observer that acknowledges OPC event conditions according to incoming values.

Each OnNext method call on its IObserver<AEAcknowledgeConditionArguments> interface acknowledges an OPC event

condition according to the arguments passed to it. OnCompleted and OnError methods do nothing.

You can create instances of AEAcknowledgeConditionObserver either by using its constructor, or with use of several overloads of the static Create method. The static AEAcknowledgeConditionObserver.Create methods use the default underlying EasyAEClient object for OPC reactive extensions. If you need to set some parameters in the client object, you can use the ClientSelector property to specify them. This allows the code be expressed only in terms of pure OPC logic, and be not tied to the actual way it is implemented.

It is recommended that you create the instances using the AEAcknowledgeConditionObserver.Create methods unless you have special needs.

6 Features

This section describes some of the larger OPC Data Client features that deserve separate documentation. They are

- **User Interface (Section 6.1)**
- **Connectivity Model (Section 6.2)**
- **Services (Section 6.3)**
- **Specialized Client Objects (Section 6.4)**

6.1 User Interface

The following table shows which functionality is available for different development platforms:

Development Platform:	**COM**	**.NET Framework**	**.NET Standard**	**Excel** (with **Excel Option (Section 12.1)**)
Features				
OPC Common Dialogs	✓	✓	×	× (1)
OPC Controls	×	✓	×	×
Unsolicited User Interaction				
Console	✓	✓	✓	×
Windows Forms	✓	✓	×	✓

Notes: (1) This functionality is not directly available on the development platform, but can be achieved by combining with VBA code (through COM platform).

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

6.1.1 OPC Common Dialogs

OPC Data Client contains a set of Windows Forms dialog boxes for performing common OPC-related tasks such as selecting an OPC server or OPC item in OPC "Classic", or selecting an OPC server endpoint, or an OPC data node within the server, in OPC Unified Archiecture.

The dialog objects are all derived from System.Windows.Forms.CommonDialog, providing consistent and well-known programming interface to use.

You can drag any of the dialog components from the Visual Studio toolbox to the designer surface, and then configure its properties. It is also possible to use the dialog components without the designer support – simply instantiate the dialog

class, and work with its properties and methods from your code.

The properties of each dialog that have significance for its functionality are categorized as follows:

Category	Description
Mode	Determines the overall behavior of the dialog. For example, some dialogs allow you to choose whether a single selection or multi-selection will be performed.
Inputs	Information necessary for running the dialog that the user cannot change. For example, with an OPC-DA Item Dialog, the dialog inputs consists of the machine name and the OPC server to be browsed.
Inputs/Outputs	Information that can be provided as initial data, but the dialog user can change it. For example, with an OPC-DA Item Dialog, the information about the selected item can passed into the dialog so that it is pre-selected at the beginning. Corresponding information may also exist in the Outputs category; the difference is that the Outputs contain more details (typically in the form of some of the XXXXElement classes), while the Inputs/Outputs only contains the necessary identifying data (could be in form of XXXXDescriptor classes).
Outputs	The selection(s) that the user has made in the dialog.

All OPC browsing dialogs have an additional SizeFactor property. The property allows the developer to influence the (initial) size of the dialog on screen. It can be set to one of the pre-defined values (SizeFactors.Small/Normal/Large/ExtraLarge), or to any SizeF structure, defining a width and height relative to the normal size. An additional SizeFactorName is provided primarily to allow easy access to this feature to COM callers.

All OPC browsing dialogs also have an additional RetainAppearance property. When set (which is the default behavior), the dialog retains its appearance (mainly, the location and size), between invocations. It is also possible to call the RevertAppearance() method on the dialog, to restore the original values.

When any of the common dialog components is placed onto the designer surface and its properties configured, it can be tested without a need to build the project. To perform the test, use the “Test Dialog” command, available from the context menu on the dialog component itself, or the same command in the Properties window, when the dialog component is selected (you may have to right-click in the property grid and check "Commands" first, to display the commands under the property grid).

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

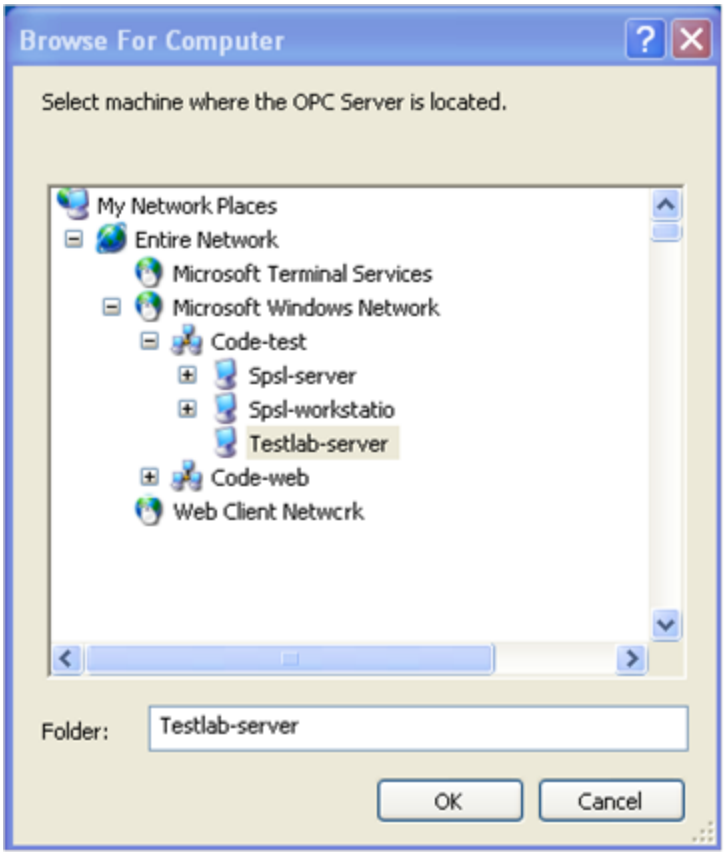
6.1.1.1 OPC-DA Common Dialogs

Articles in this section apply to OPC "Classic" DA and OPC XML-DA.

6.1.1.1.1 Computer Browser Dialog (OPC DA usage)

Icon: 

OPC servers are usually deployed on the network, and accessed via DCOM. In order to let the user select the remote computer where the OPC server resides, you can use the ComputerBrowserDialog object.



Call the ShowDialog method, and if the result is equal to DialogResult.OK, the user has selected the computer, and its name can be retrieved from the SelectedName property.

C++

```
// This example shows how to let the user browse for computers on the network.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _ComputerBrowserDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _ComputerBrowserDialogPtr DialogPtr(__uuidof(ComputerBrowserDialog));

            //
            DialogResult dialogResult = DialogPtr->ShowDialog(NULL);
```

```
            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(DialogPtr->SelectedName));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for computers on the network.

class procedure ShowDialog.Main;
var
  Dialog: ComputerBrowserDialog;
begin
  // Instantiate the dialog object
  Dialog := CoComputerBrowserDialog.Create;

  Dialog.ShowDialog(nil);
  WriteLn(Dialog.SelectedName);
end;
```

Object Pascal

```
// This example shows how to let the user browse for computers on the network.

class procedure ShowDialog.Main;
var
  ComputerBrowserDialog: OpcLabs_BaseLibForms_TLB._ComputerBrowserDialog;
begin
  // Instantiate the dialog object
  ComputerBrowserDialog := CoComputerBrowserDialog.Create;

  ComputerBrowserDialog.ShowDialog(nil);
  WriteLn(ComputerBrowserDialog.SelectedName);
end;
```

PHP

```
// This example shows how to let the user browse for computers on the network.

$Dialog = new COM("OpcLabs.BaseLib.Forms.Browsing.ComputerBrowserDialog");
printf("%d\n", $Dialog->ShowDialog);

// Display results
printf("%s\n", $Dialog->SelectedName);
```

Python

```
# This example shows how to let the user browse for computers on the network.

import win32com.client

dialog =
```

```
win32com.client.Dispatch('OpcLabs.BaseLib.Forms.Browsing.ComputerBrowserDialog')
print(dialog.ShowDialog())

# Display results
print(dialog.SelectedName)
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for computers on the network.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim Dialog As New ComputerBrowserDialog
    Dim DialogResult
    DialogResult = Dialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & Dialog.SelectedName & vbCrLf
End Sub
```

VBScript

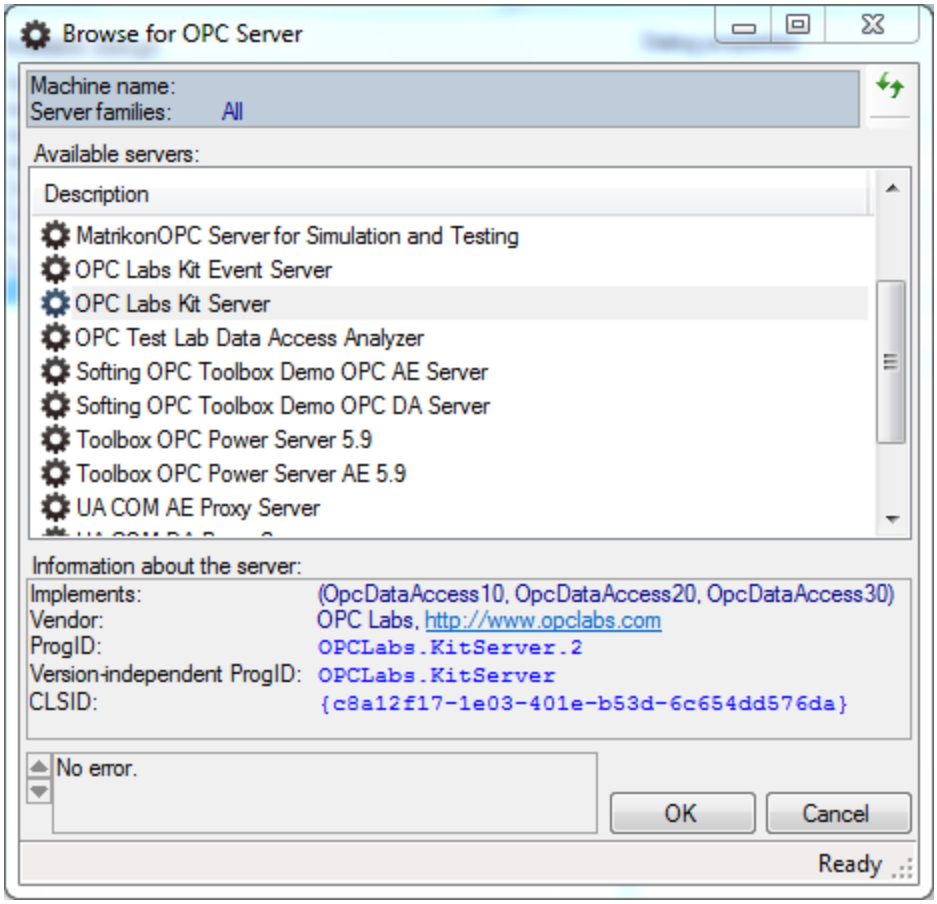
```
Rem This example shows how to let the user browse for computers on the network.

Option Explicit

Const DialogResult_OK = 1

Dim Dialog: Set Dialog =
CreateObject("OpcLabs.BaseLib.Forms.Browsing.Specialized.ComputerBrowserDialog")
Dim dialogResult: dialogResult = Dialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo Dialog.SelectedName
```

6.1.1.1.2 OPC Server Dialog (OPC DA usage)

Icon : 

If you do not know upfront which OPC server to connect to, and do not have this information from any other source, your application will need to allow the user select the OPC server(s) to work with. The OPC Server Dialog (OpcServerDialog class) allows the user to select the OPC server interactively from the list of OPC Data Access servers installed on a particular machine.

Here is an example of OPC Server dialog in action:



The ServerFamilies property of the dialog determines whether OPC Data Access servers, OPC Alarms&Events servers, other servers, or a combination of them will be browsed. By default, all server families are browsed. In order to limit the browsing to OPC Data Access servers only, set the ServerFamilies property to OpcDataAccess enumeration member.

To run the dialog, set the Location property to the name of the computer that is to be browsed, and call the ShowDialog method. If the result is equal to DialogResult.OK, the user has selected the OPC Data Access server, and information about it can be retrieved from the ServerElement property.

VBScript

```
Rem This example shows how to let the user browse for an OPC "Classic" server.

Option Explicit

Const DialogResult_OK = 1

Dim ServerDialog: Set ServerDialog =
CreateObject("OpcLabs.EasyOpc.Forms.Browsing.OpcServerDialog")
'ServerDialog.Location = ""
Dim dialogResult: dialogResult = ServerDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
```

```
End If

' Display results
WScript.Echo ServerDialog.ServerElement
```

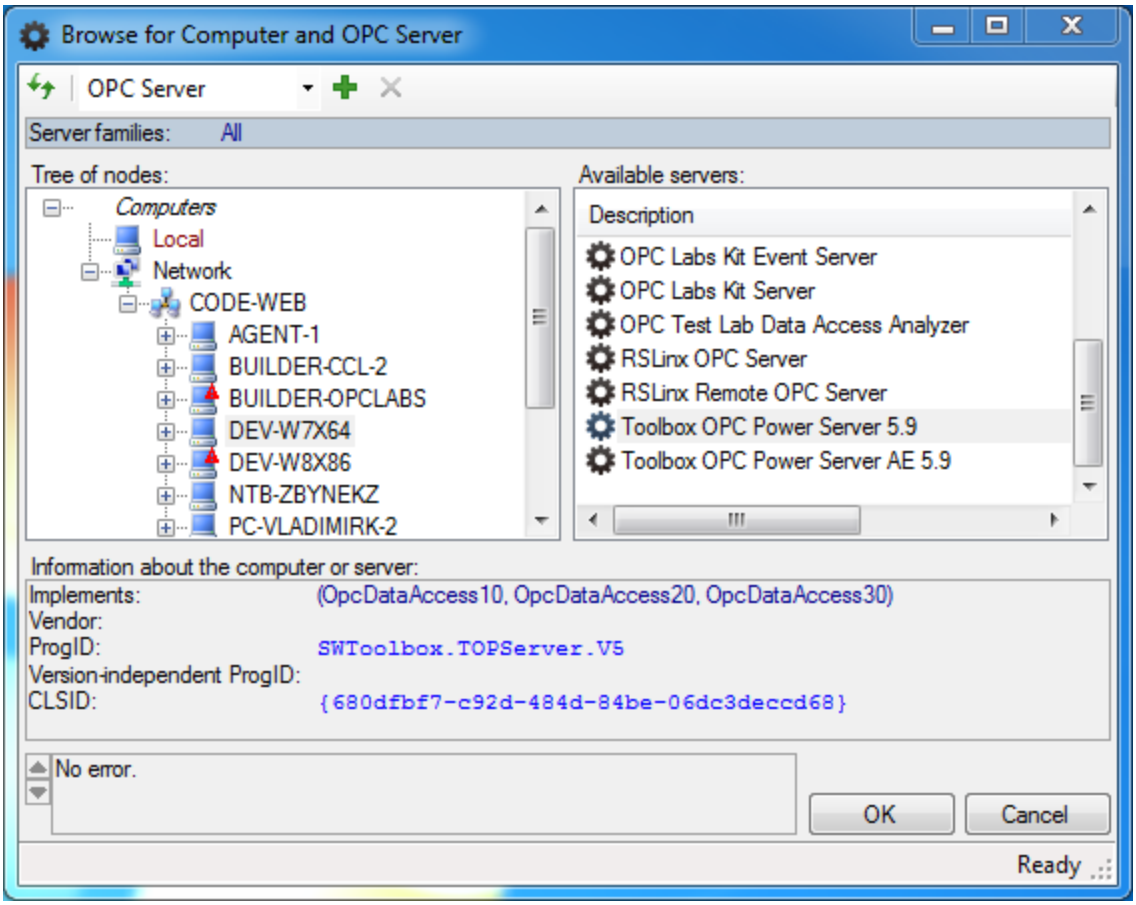
6.1.1.1.3 OPC Computer and Server Dialog

Icon: 

With OpcComputerAndServerDialog, your application can integrate a dialog box from which the user can select a computer and an OPC server residing on it. This dialog box combines functions of the Computer Browser Dialog -  and OPC Server Dialog into a single dialog.

In addition, the user can add Computer nodes in case the particular computer is not visible on the network, or add OPC Server nodes in case the OPC Server is known to exist but is not returned by OPC Server enumeration feature.

Here is an example of OPC Computer and Server dialog in action:

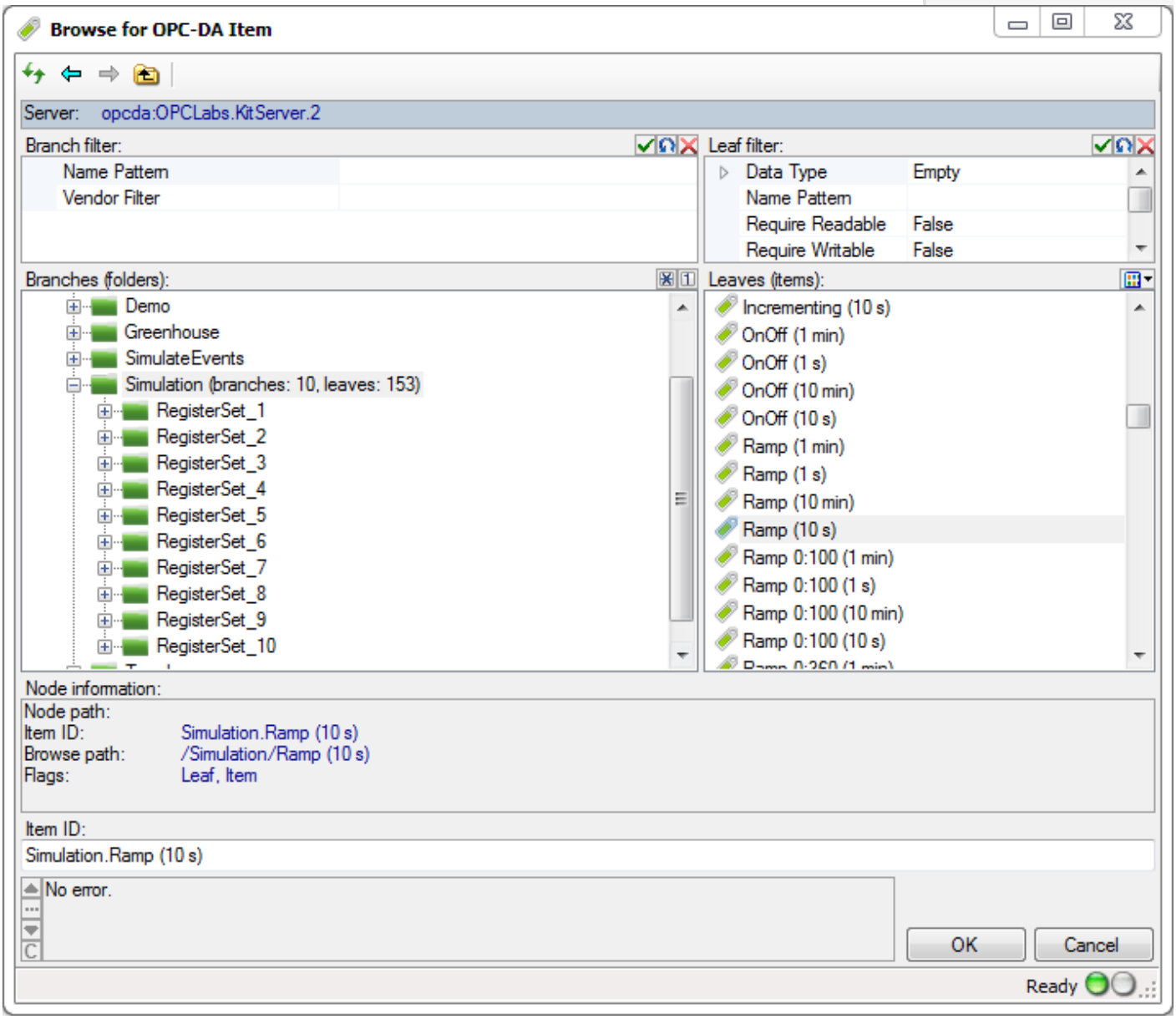


6.1.1.1.4 OPC-DA Item Dialog

Icon: 

The OPC-DA Item Dialog (DAItemDialog class) allows the user to interactively select the OPC item residing in a specific OPC server.

In This Topic



Use the ServerDescriptor property to specify the OPC Data Access server whose items are to be browsed, and call the ShowDialog method. If the result is equal to DialogResult.OK, the user has selected the OPC item, and information about it can be retrieved from the NodeElement property.

Object Pascal

```
// This example shows how to let the user browse for an OPC Data Access item.

class procedure ShowDialog.Main;
```

```
var
  ItemDialog: OpcLabs_EasyOpcForms_TLB._DAItemDialog;
begin
  // Instantiate the dialog object
  ItemDialog := CoDAItemDialog.Create;

  ItemDialog.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';

  ItemDialog.ShowDialog(nil);

  // Display results
  WriteLn(ItemDialog.NodeElement.ToString);
end;
```

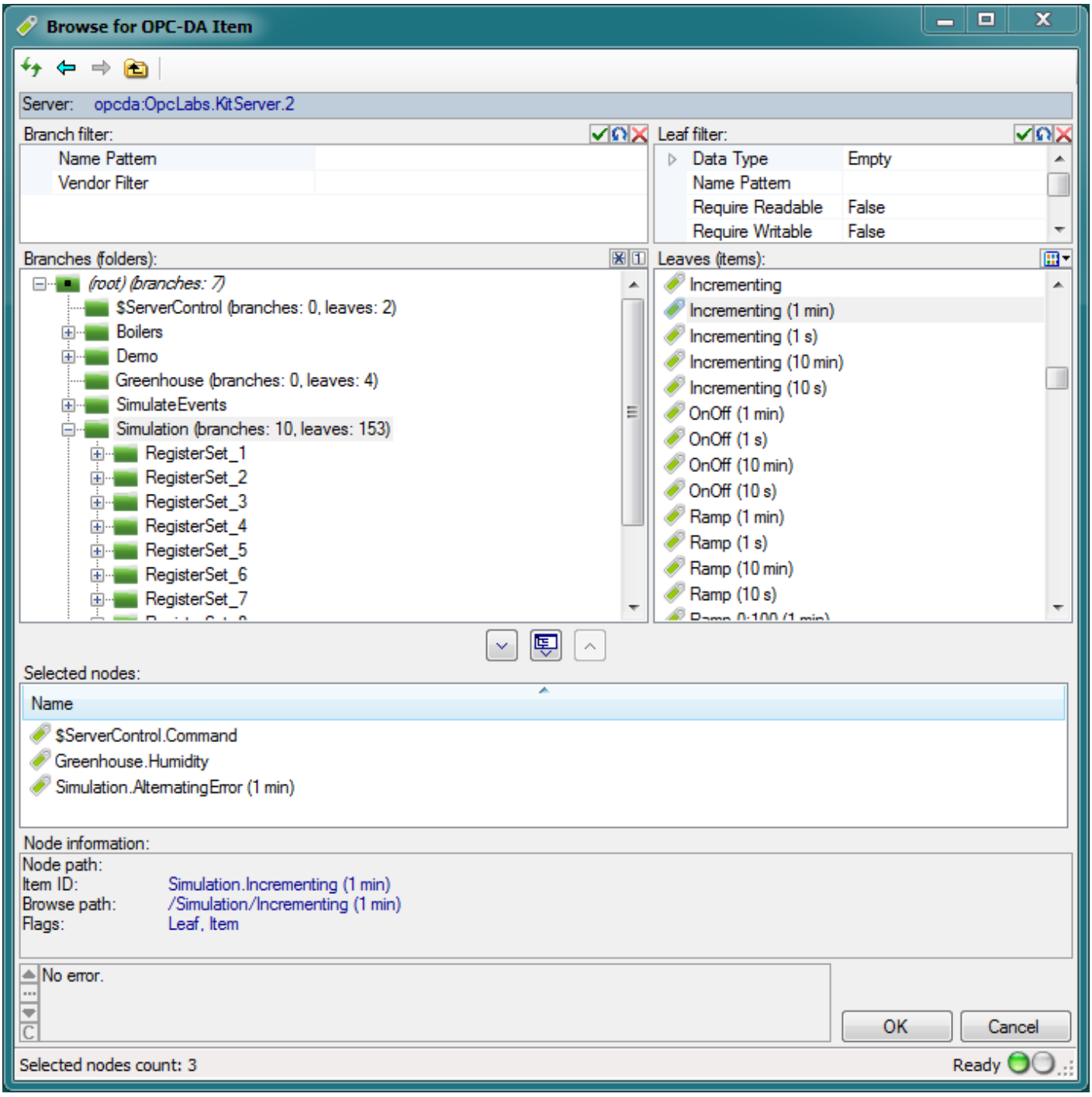
VBScript

```
Rem This example shows how to let the user browse for an OPC Data Access item.

Option Explicit

Const DialogResult_OK = 1

Dim ItemDialog: Set ItemDialog =
CreateObject("OpcLabs.EasyOpc.DataAccess.Forms.Browsing.DAItemDialog")
ItemDialog.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
Dim dialogResult: dialogResult = ItemDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo "NodeElement: " & ItemDialog.NodeElement
```

The DAItemDialog component retains the filter setting for each node between the invocations of the dialog, making it faster for the user to navigate during the subsequent invocations.

Multi-selection

When you set the MultiSelect property of the DAItemDialog to true, the dialog will allow the user to select any number of OPC-DA items. In the multi-select mode, the dialog looks similar to this:



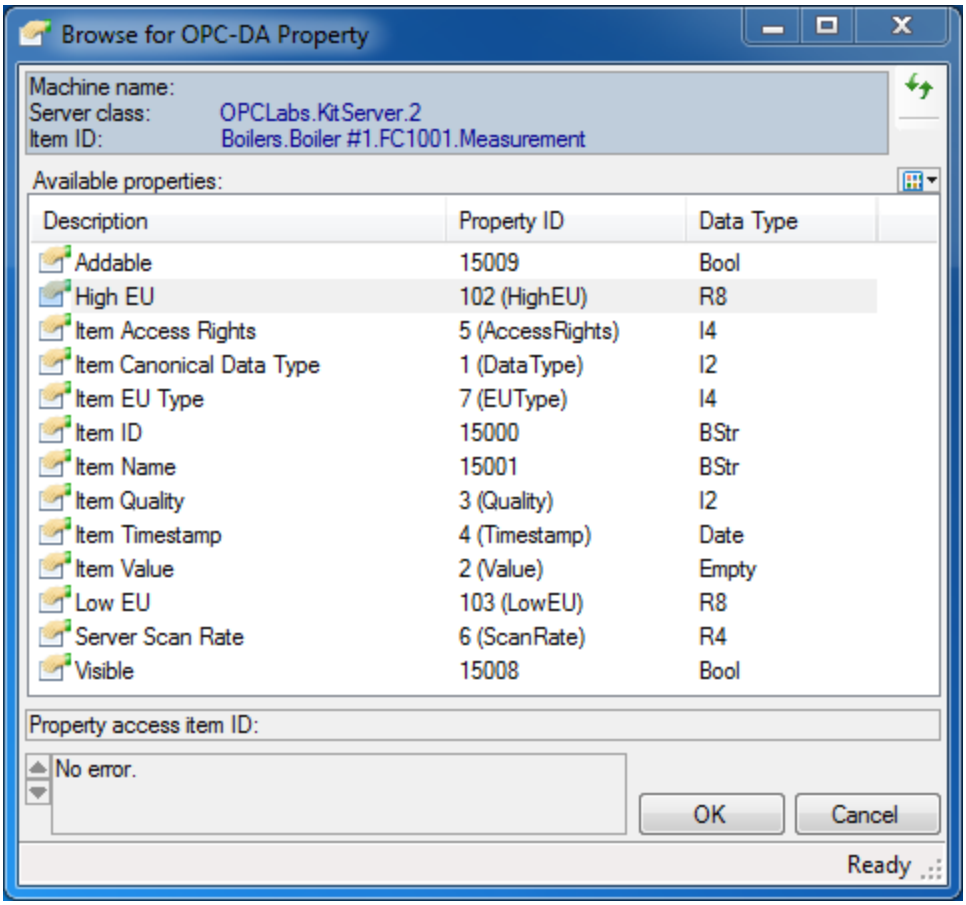
The list below the branches and leaves (labeled “Selected nodes”) contains the set of nodes that the user has selected in the dialog. The user can freely add nodes to this list, or remove them. The selected set is carried over to next invocation of the dialog, unless you change it from the code.

In the multi-selection mode, the set of nodes selected on input (if any) is in the NodeDescriptors property. On the output, the dialog fills the information about selected nodes into the NodeElements property (and updated the NodeDescriptors property as well).

6.1.1.1.5 OPC-DA Property Dialog

Icon: 

The OPC-DA Property Dialog (DAPropertyDialog class) allows the user to interactively select the OPC property on a specific OPC item.



Use the ServerDescriptor property to specify the OPC Data Access server whose items are to be browsed, set the NodeDescriptor property to the OPC Item Id needed, and then call the ShowDialog method. If the result is equal to DialogResult.OK, the user has selected the OPC property, and information about it can be retrieved from the PropertyElement property.

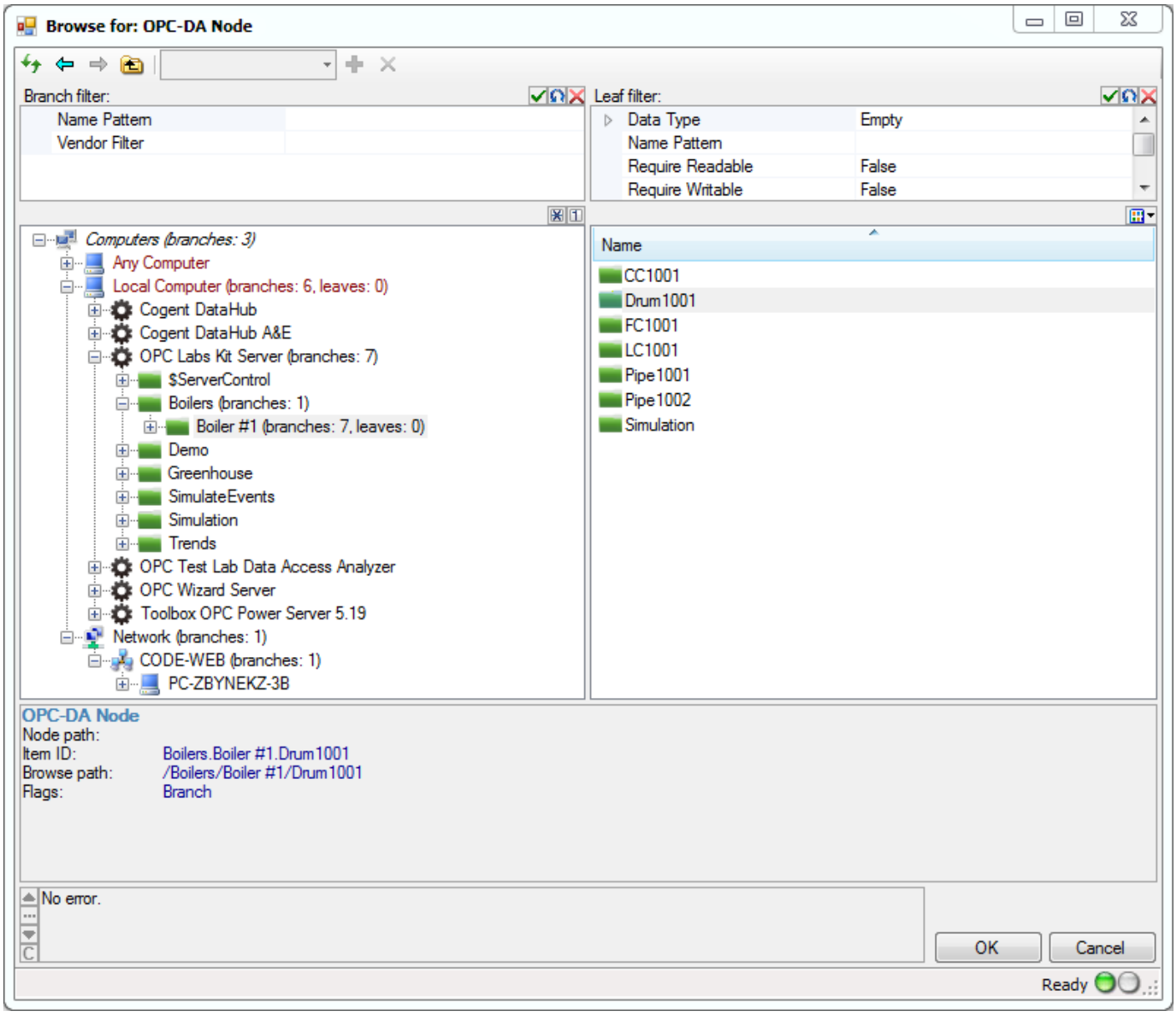
6.1.1.1.6 Generic OPC Browsing Dialog (OPC DA usage)

Icon: 

With OpcBrowseDialog, your application can integrate a dialog with various OPC nodes from which the user can select. This dialog can be configured to serve many different purposes.

Here is an example of the generic OPC browsing dialog in action:

In This Topic

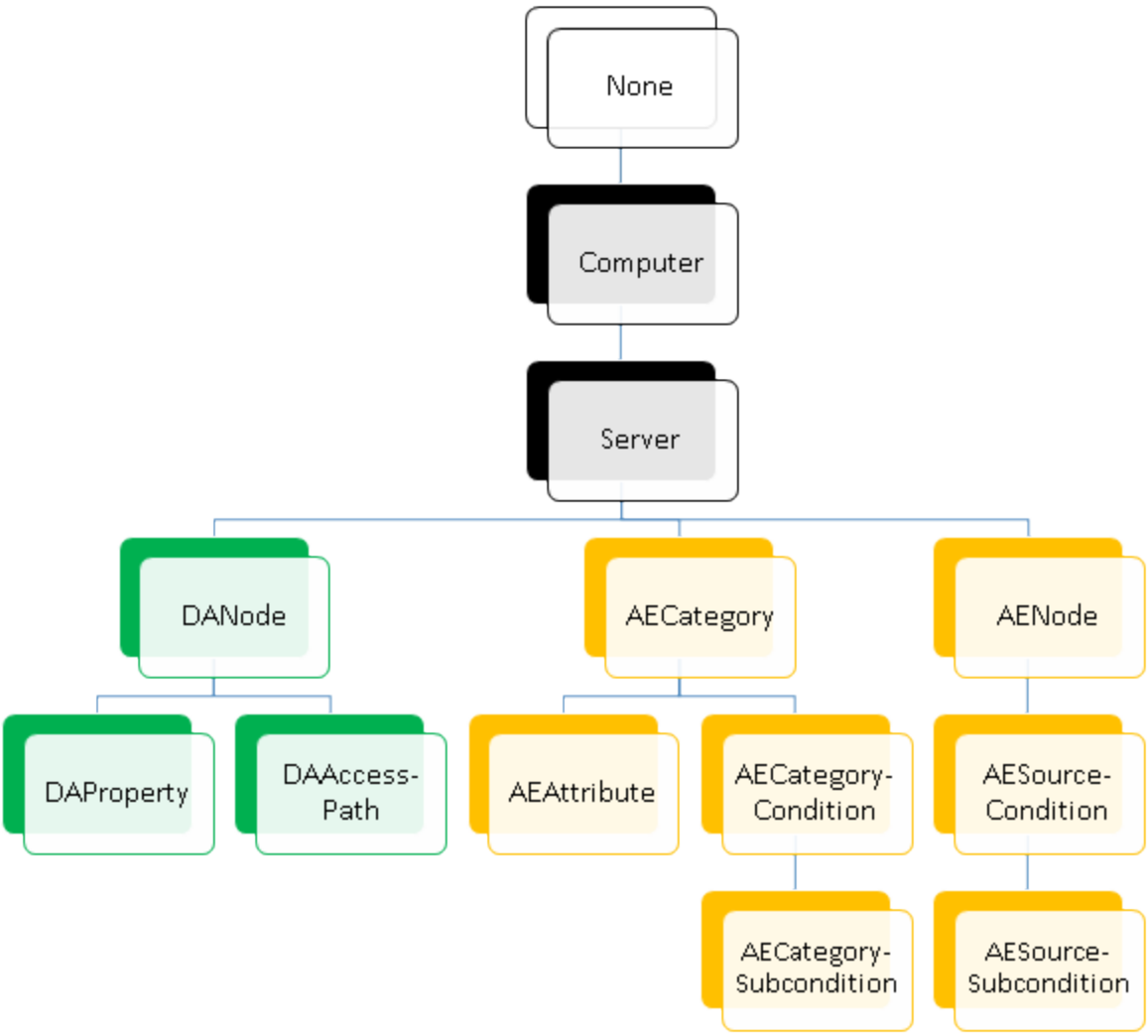


The way the dialog operates is controlled by two main properties:

- Mode.AnchorElementType determines the part of the selection that is given on input and cannot be changed by the user.
- Mode.SelectElementType determines the type of the element that the user needs to select in order to finalize the dialog.

Values of these properties can be selected from the OpcElementType enumeration, which has members for various types of elements that you encounter when working with OPC.

The following chart shows a hierarchy of element types that you can choose from:



For example, let' say that you set Mode.AnchorElementType to Server, and Mode.SelectElementType to DAProperty. This will cause the dialog to allow the user to browse for an OPC Item (node) on the server you specify, and then for an OPC property on that item.

In this case, before you run the dialog, you need to provide it with values for the InputOutputs.CurrentNodeDescriptor.ServerDescriptor.Location and InputOutputs.CurrentNodeDescriptor.ServerDescriptor.ServerClass properties, because those define your "anchor" element (Server) that the user cannot change. The dialog will only allow the user to finalize it (besides cancelling) after an OPC property is fully selected, because that is your Mode.SelectElementType. After the dialog is successfully finalized, the information about the user's choice will be available in the Outputs.CurrentNodeElement.DANodeElement and Outputs.CurrentNodeElement.DAPropertyElement properties.

Note that in addition to the "minimal" scenario described above, you can also pre-set the initial node or property, using the InputOutputs.CurrentNodeDescriptor.DANodeDescriptor or InputOutputs.CurrentNodeDescriptor.DAPropertyDescriptor.PropertyId properties, and after the selection is made, these properties will be updated to the new selection as well. This way, if you run the dialog again with the same value, the initial selection will be where the user has left it the last time the dialog was run.

Obviously, the chosen Mode.SelectElementType must be a child or indirect ancestor of chosen Mode.AnchorElementType in the hierarchy. For example, it would be an error to set Mode.AnchorElementType to AECategory and Mode.SelectElementType to DAProperty.

Object Pascal

```
// This example shows how to let the user browse for an OPC Data Access node.

class procedure ShowDialog.Main;
var
  BrowseDialog: OpcLabs_EasyOpcForms_TLB._OpcBrowseDialog;
begin
  // Instantiate the dialog object
  BrowseDialog := CoOpcBrowseDialog.Create;

  BrowseDialog.ShowDialog(nil);

  // Display results
  WriteLn(BrowseDialog.Outputs.CurrentNodeElement.DANodeElement.ToString);
end;
```

VBScript

```
Rem This example shows how to let the user browse for an OPC Data Access node.

Option Explicit

Const DialogResult_OK = 1

Dim BrowseDialog: Set BrowseDialog =
CreateObject("OpcLabs.EasyOpc.Forms.Browsing.OpcBrowseDialog")
Dim dialogResult: dialogResult = BrowseDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo BrowseDialog.Outputs.CurrentNodeElement.DANodeElement
```

Multi-selection

It is also possible to configure the dialog for a multi-selection. In this mode, the user can select zero, one, or more nodes. In order to enable the multi-select mode, set the Mode.MultiSelect property to true. In the multi-select mode, the initial

set of the selected nodes (when the dialog is first displayed to the user) is given by the contents of the InputOutputs.SelectionDescriptors collection. When the user makes the selection and accepts it by closing the dialog, this collection is updated, and also, all information about the selected nodes is placed to the Outputs.SelectionElements collection.

Other settings

There are also ways to control some finer aspects of the dialog. For example, the Mode.ShowListBranches property (defaults to true) controls whether the branches of the tree are also displayed in the list view.

6.1.1.2 OPC-A&E Common Dialogs

6.1.1.2.1 Computer Browser Dialog (OPC A&E)

Icon: 

With ComputerBrowserDialog object, you can present your user with a dialog for selecting the remote computer where the OPC server resides.

For more information, see the Computer Browser dialog under “User Interface for OPC-DA”.

6.1.1.2.2 OPC Server Dialog (OPC A&E usage)

Icon: 

The OPC Server Dialog (OpcServerDialog class) allows the user to select the OPC server interactively from the list of OPC “Classic” servers installed on a particular machine.

To select from OPC Alarms&Events servers, you use the same dialog as with OPC Data Access servers – for details on that dialog, see OPC Server Dialog under “User Interface for OPC-DA”. The only difference is that before running the dialog, you need to make sure that its ServerFamilies property is set so that it includes (or only contains) the OpcAlarmsAndEvents enumeration member.

6.1.1.2.3 OPC Computer and Server Dialog

Icon: 

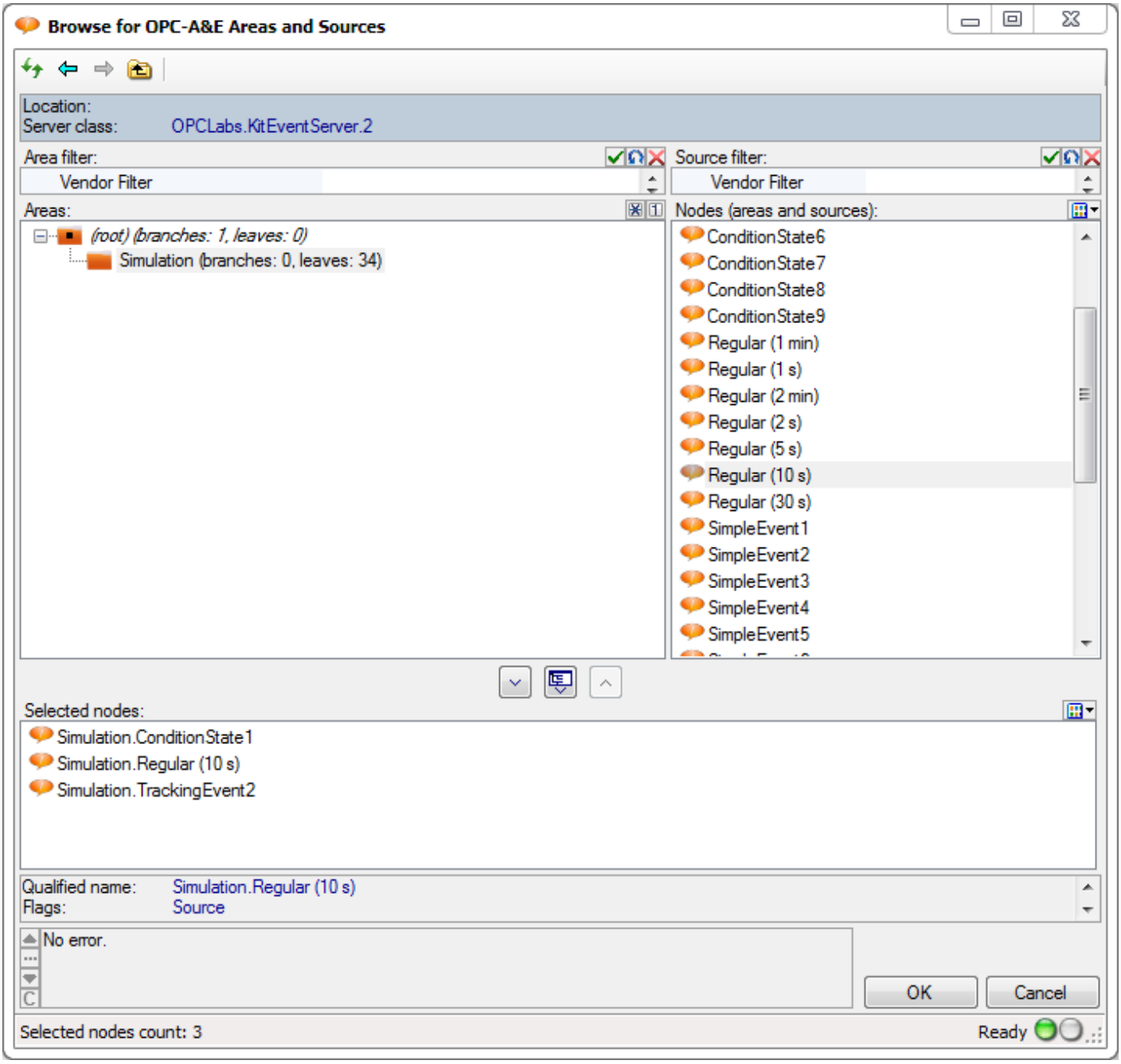
With OpcComputerAndServerDialog, your application can integrate a dialog box from which the user can select a computer and an OPC server residing on it.

This dialog is identical to OPC Computer and Server Dialog described in the OPC-DA Common Dialogs chapter. The only difference is that before running the dialog, you need to make sure that its ServerFamilies property is set so that it includes (or only contains) the OpcAlarmsAndEvents enumeration member.

6.1.1.2.4 OPC-A&E Area or Source Dialog

Icon: 

With AEAreaOrSourceDialog, your applicate can integrate a dialog box from which the user can select OPC-A&E event areas or sources:



The MultiSelect property determines whether the dialog allows the user to select multiple nodes as output.

Use the ServerDescriptor property to specify the OPC Alarms&Events server whose items are to be browsed, and call the ShowDialog method. If the result is equal to DialogResult.OK, the user has finished the selection, and information about it can be retrieved.

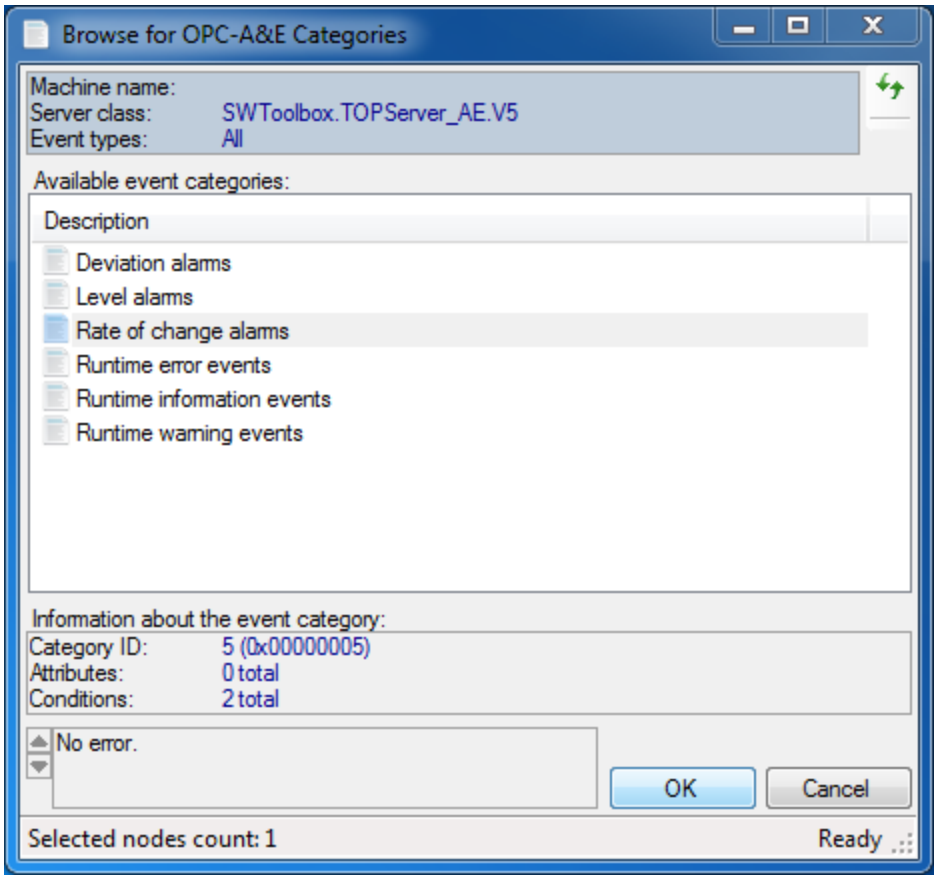
When the dialog is in single-select mode, you can obtain the information about the selected event area or source from the NodeDescriptor property. When the dialog is in multi-select mode, the information about the selected event areas is in the AreaElements property, and the information about the selected event sources in the SourceElements property; all

event areas and nodes are available together in the NodeElements property.

6.1.1.2.5 OPC-A&E Category Dialog

Icon: 

With AECategoryDialog, your application can integrate a dialog box from which the user can select from OPC-A&E event categories provided by the OPC server:



The MultiSelect property determines whether the dialog allows the user to select multiple categories as output.
The EventTypes property determines which event type(s) will be offered for selection.

Use the ServerDescriptor property to specify the OPC Alarms&Events server whose categories are to be browsed, and call the ShowDialog method. If the result is equal to DialogResult.OK, the user has finished the selection, and information about it can be retrieved.

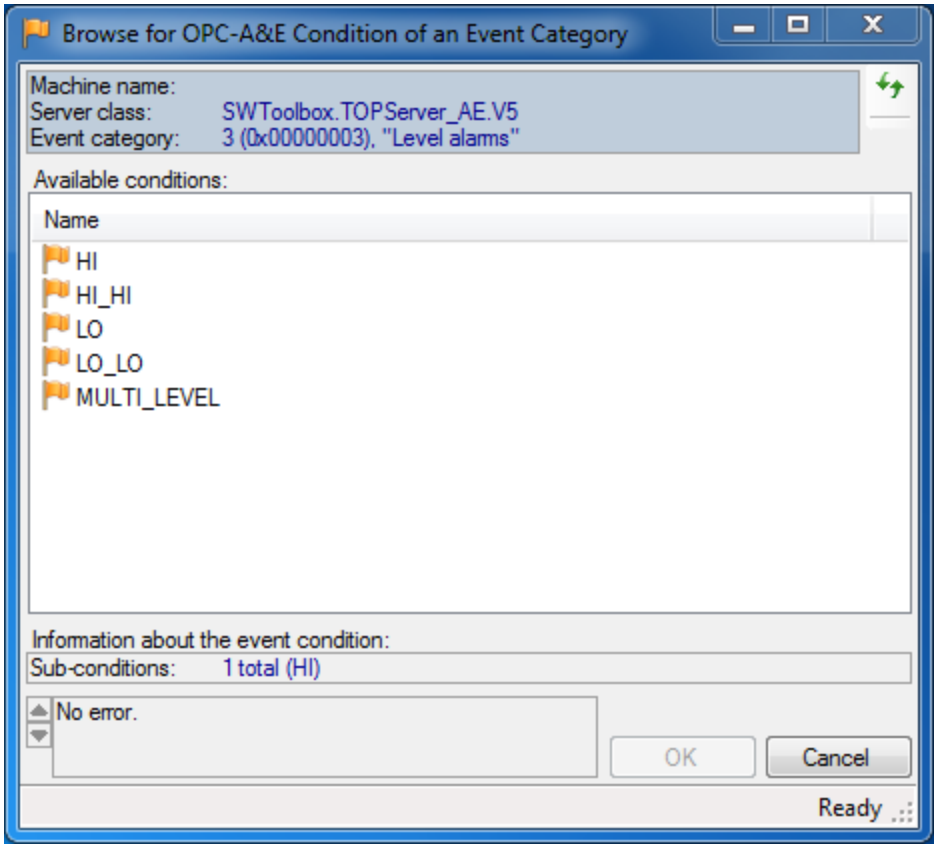
When the dialog is in single-select mode, you can obtain the information about the selected category from the CategoryElement or CategoryId property. When the dialog is in multi-select mode, the information about the selected categories is in the CategoryElements and CategoryIds properties.

6.1.1.2.6 OPC-A&E Category Condition Dialog

Icon: 

With AECategoryConditionDialog, your application can integrate a dialog box from which the user can select OPC-A&E

category available on a specified event condition:

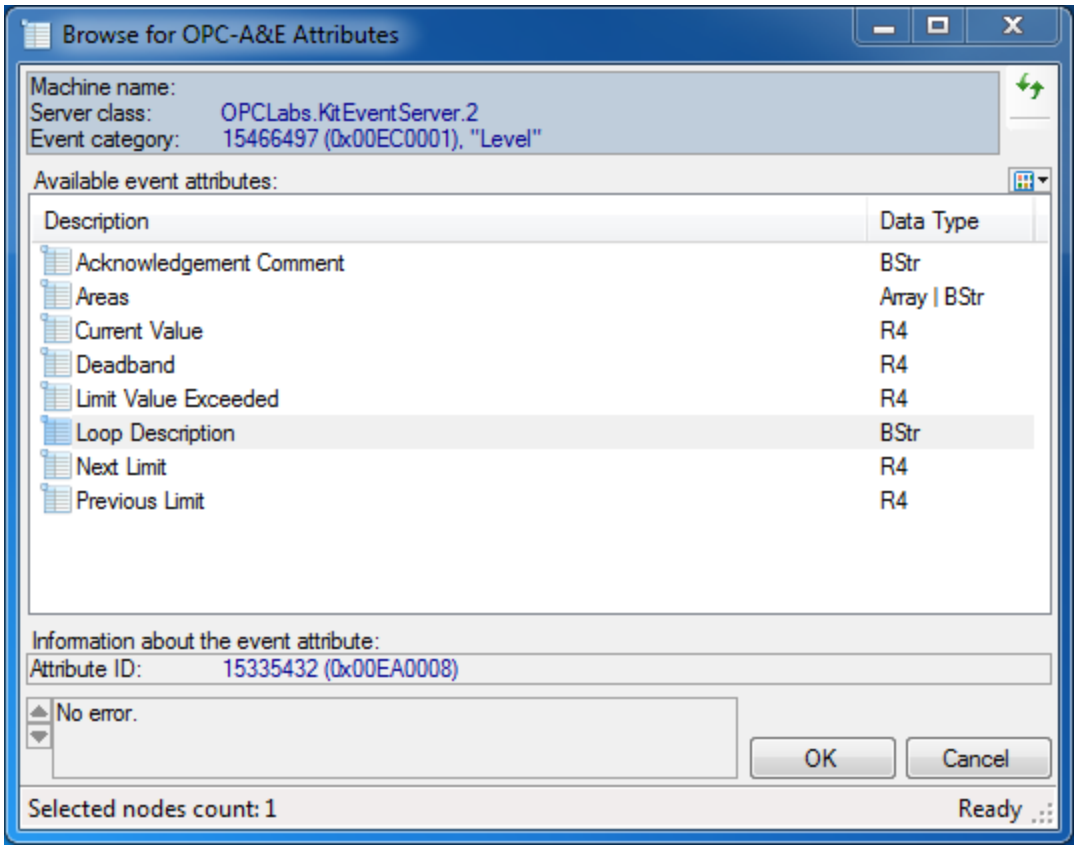


Use the ServerDescriptor property to specify the OPC Alarms&Events server, and the CategoryId property to specify the event category to be browsed. Then, call the ShowDialog method. If the result is equal to DialogResult.OK, the user has selected the event condition, and information about it can be retrieved from the ConditionElement or ConditionName property.

6.1.1.2.7 OPC-A&E Attribute Dialog

Icon: 

With AEAttributeDialog, your application can integrate a dialog box from which the user can select OPC-A&E event attributes:



The MultiSelect property determines whether the dialog allows the user to select multiple attributes as output.

Use the ServerDescriptor property to specify the OPC Alarms&Events server and the CategoryId property to specify the event category to be browsed. Then, call the ShowDialog method. If the result is equal to DialogResult.OK, the user has finished the selection, and information about it can be retrieved.

When the dialog is in single-select mode, you can obtain the information about the selected attribute from the AttributeElement or AttributeId property. When the dialog is in multi-select mode, the information about the selected categories is in the AttributeElements and AttributeIds properties.

6.1.1.2.8 Generic OPC Browsing Dialog (OPC A&E usage)

Icon: 

With OpcBrowseDialog, your application can integrate a dialog with various OPC nodes from which the user can select. This dialog can be configured to serve many different purposes.

As this dialog is shared with other OPC specifications, please refer to Generic OPC Browsing Dialog above for more information.

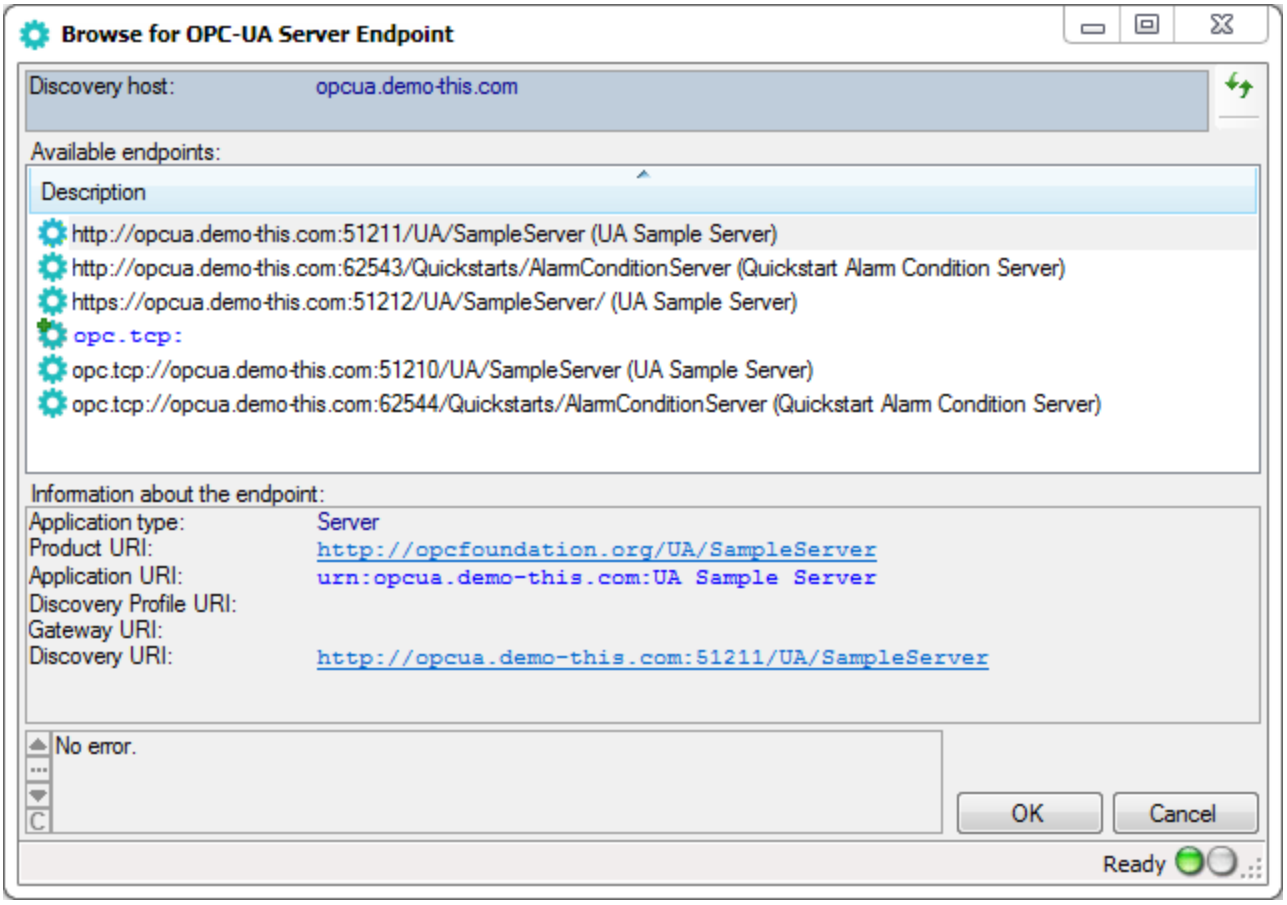
6.1.1.3 OPC-UA Common Dialogs

6.1.1.3.1 OPC-UA Endpoint Dialog

Icon: 

If you do not know upfront which OPC-UA server and its endpoint to connect to, and do not have this information from any other source, your application will need to allow the user select the OPC server(s) to work with. The OPC-UA Endpoint Dialog (UAEndpointDialog class) allows the user to select the OPC server and its endpoint interactively from the list of OPC Unified Architecture servers provided by a LDS (Local Discovery Server) installed on a machine.

Here is an example of OPC-UA Endpoint dialog in action:



To run the dialog, call the ShowDialog method. If the result is equal to DialogResult.OK, the user has selected the OPC UA server, and information about it can be retrieved from the DiscoveryElement property.

Currently, the dialog shows endpoint provided by a LDS (Local Discovery Server) that you set by a static property on the EasyUAClient object. This approach may change in future.

C++

```
// This example shows how to let the user browse for an OPC-UA server endpoint.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UAEndpointDialog
{
    void ShowDialog1::Main()
    {
        // Initialize the COM library
```

```
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UAEndpointDialogPtr EndpointDialogPtr(__uuidof(UAEndpointDialog));

            //
            EndpointDialogPtr->DiscoveryHost = L"opcua.demo-this.com";

            //
            DialogResult dialogResult = EndpointDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(EndpointDialogPtr->DiscoveryElement-
>ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for an OPC-UA server endpoint.

class procedure ShowDialog.Main;
var
  EndpointDialog: UAEndpointDialog;
begin
  // Instantiate the dialog object
  EndpointDialog := CoUAEndpointDialog.Create;

  EndpointDialog.DiscoveryHost := 'opcua.demo-this.com';

  EndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn(EndpointDialog.DiscoveryElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for an OPC-UA server endpoint.

class procedure ShowDialog.Main;
var
  EndpointDialog: OpcLabs_EasyOpcForms_TLB._UAEndpointDialog;
begin
  // Instantiate the dialog object
  EndpointDialog := CoUAEndpointDialog.Create;

  EndpointDialog.DiscoveryHost := 'opcua.demo-this.com';

  EndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn(EndpointDialog.DiscoveryElement.ToString);
```

```
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for an OPC-UA server endpoint.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim EndpointDialog As New UAEndpointDialog
    EndpointDialog.DiscoveryHost = "opcua.demo-this.com"

    Dim DialogResult
    DialogResult = EndpointDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & EndpointDialog.DiscoveryElement & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for an OPC-UA server endpoint.

Option Explicit

Const DialogResult_OK = 1

Dim EndpointDialog: Set EndpointDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UAEndpointDialog")
EndpointDialog.DiscoveryHost = "opcua.demo-this.com"

Dim dialogResult: dialogResult = EndpointDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo EndpointDialog.DiscoveryElement
```

6.1.1.3.2 OPC-UA Host and Endpoint Dialog

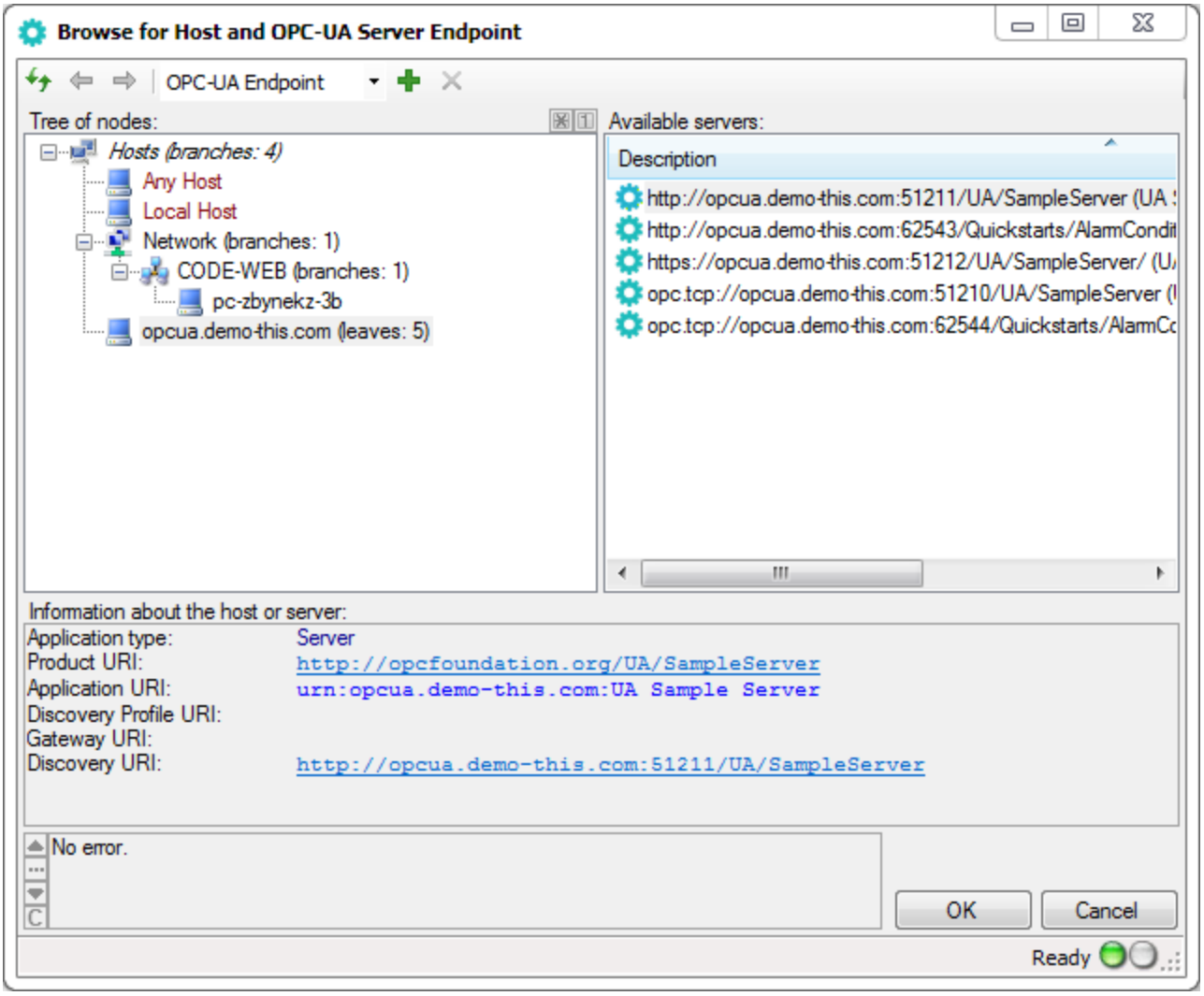
Icon: 

With UAHostAndEndpointDialog, your application can integrate a dialog box from which the user can select a host (computer) and an endpoint of OPC-UA server residing on it. This dialog box combines functions of the Computer Browser Dialog and OPC-UA Endpoint Dialog -  into a single dialog.

In addition, the user can add Host nodes in case the particular computer is not visible on the network, or add OPC-UA

Endpoint nodes in case the OPC_UA Server is known to exist but is not returned by OPC-UA discovery feature.

Here is an example of OPC-UA Host and Endpoint dialog in action:



C++

```
// This example shows how to let the user browse for a host (computer) and an endpoint
of an OPC-UA server residing on it.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UAHostAndEndpointDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UAHostAndEndpointDialogPtr
```

```
HostAndEndpointDialogPtr(__uuidof(UAHostAndEndpointDialog));

            //
            HostAndEndpointDialogPtr->EndpointDescriptor->Host = L"opcua.demo-
this.com";

            //
            DialogResult dialogResult = HostAndEndpointDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("HostElement: %s\n"), (LPCTSTR)CW2CT(HostAndEndpointDialogPtr-
>HostElement->ToString));
            _tprintf(_T("DiscoveryElement: %s\n"),
(LPCTSTR)CW2CT(HostAndEndpointDialogPtr->DiscoveryElement->ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for a host (computer) and
// an endpoint of an OPC-UA server residing on it.

class procedure ShowDialog.Main;
var
  HostAndEndpointDialog: UAHostAndEndpointDialog;
begin
  // Instantiate the dialog object
  HostAndEndpointDialog := CoUAHostAndEndpointDialog.Create;

  HostAndEndpointDialog.EndpointDescriptor.Host := 'opcua.demo-this.com';

  HostAndEndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn('HostElement: ', HostAndEndpointDialog.HostElement.ToString);
  WriteLn('DiscoveryElement: ', HostAndEndpointDialog.DiscoveryElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for a host (computer) and
// an endpoint of an OPC-UA server residing on it.

class procedure ShowDialog.Main;
var
  HostAndEndpointDialog: OpcLabs_EasyOpcForms_TLB._UAHostAndEndpointDialog;
begin
  // Instantiate the dialog object
  HostAndEndpointDialog := CoUAHostAndEndpointDialog.Create;

  HostAndEndpointDialog.EndpointDescriptor.Host := 'opcua.demo-this.com';

  HostAndEndpointDialog.ShowDialog(nil);
```

```
  // Display results
  WriteLn('HostElement: ', HostAndEndpointDialog.HostElement.ToString);
  WriteLn('DiscoveryElement: ', HostAndEndpointDialog.DiscoveryElement.ToString);
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for a host (computer) and an endpoint
of an OPC-UA server residing on it.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim HostAndEndpointDialog As New UAHostAndEndpointDialog
    HostAndEndpointDialog.EndpointDescriptor.Host = "opcua.demo-this.com"

    Dim DialogResult
    DialogResult = HostAndEndpointDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & "HostElement: " & HostAndEndpointDialog.HostElement &
vbCrLf
    OutputText = OutputText & "DiscoveryElement: " &
HostAndEndpointDialog.DiscoveryElement & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for a host (computer) and an endpoint
of an OPC-UA server residing on it.

Option Explicit

Const DialogResult_OK = 1

Dim HostAndEndpointDialog: Set HostAndEndpointDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UAHostAndEndpointDialog")
HostAndEndpointDialog.EndpointDescriptor.Host = "opcua.demo-this.com"

Dim dialogResult: dialogResult = HostAndEndpointDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo "HostElement: " & HostAndEndpointDialog.HostElement
WScript.Echo "DiscoveryElement: " & HostAndEndpointDialog.DiscoveryElement
```

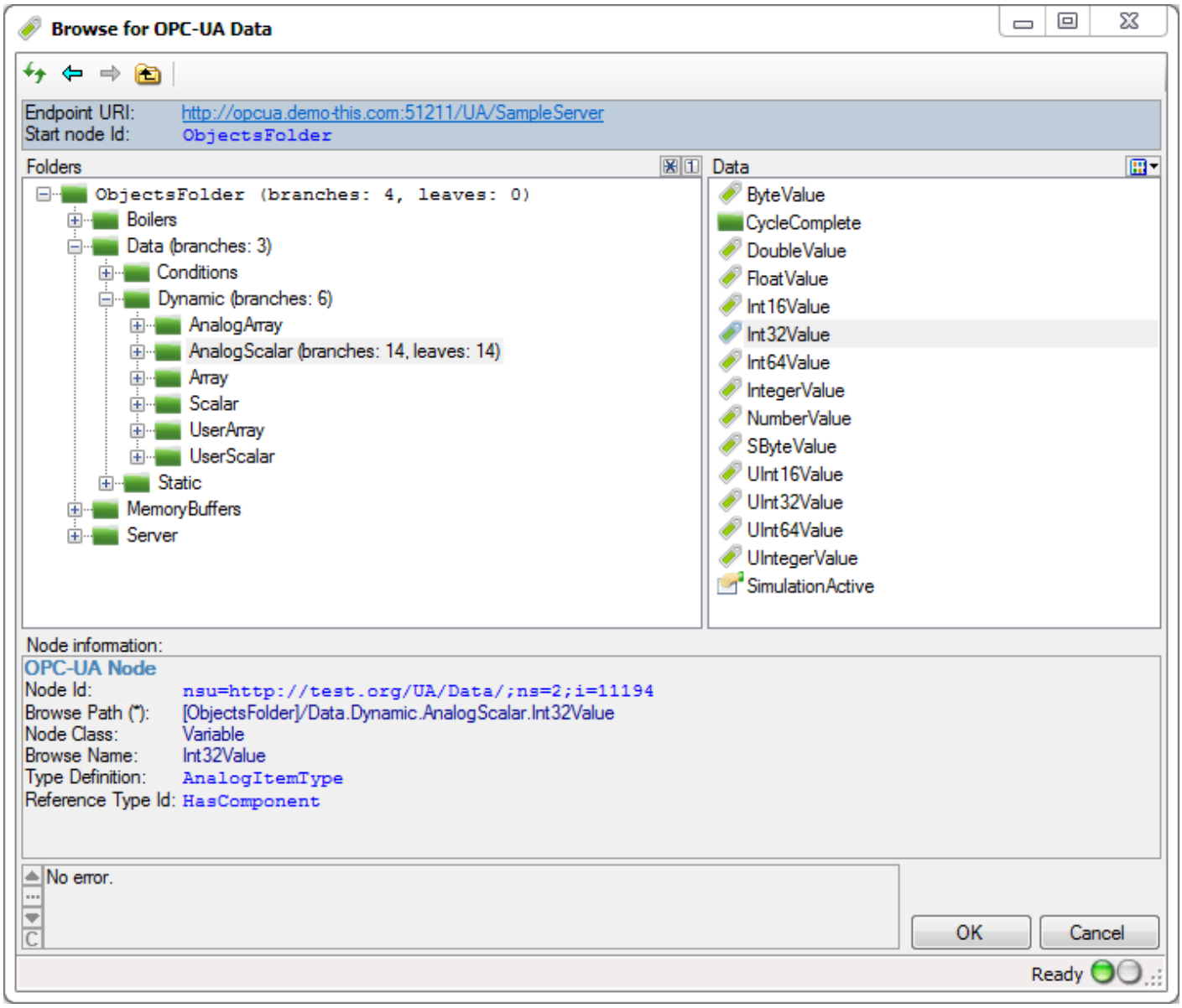
6.1.1.3.3 OPC-UA Data Dialog

Icon: 

The OPC-UA Data Dialog (UADataDialog class) allows the user to interactively select the OPC data node residing in a specific OPC Unified Architecture server. It also has a different mode (controlled by the UserPickEndpoint property) which allows the user to start the node selection by choosing the host and server endpoint first.

In This Topic

Here is an example of OPC-UA Data dialog in action:



In the default mode (when UserPickEndpoint is false), your code should set the EndpointDescriptor property to specify the OPC Unified Architecture server whose nodes are to be browsed. If you want the user to pick the endpoint, set the UserPickEndpoint property to true; in this case, setting the EndpointDescriptor is not needed on input.

After setting the inputs as needed, your code calls the ShowDialog method. If the result is equal to DialogResult.OK, the

user has selected the OPC data node, and information about it can be retrieved from the NodeElement (and also NodeDescriptor) property. If UserPickEndpoint is true, the chosen server endpoint can be retrieved from the EndpointDescriptor property.

The user can browse through the UA Objects, and select from Data Variables or Properties.

C++

```
// This example shows how to let the user browse for an OPC-UA data node (a Data
Variable or a Property).

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UADataDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UADataDialogPtr DataDialogPtr(__uuidof(UADataDialog));

            //
            DataDialogPtr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            DataDialogPtr->UserPickEndpoint = true;

            //
            DialogResult dialogResult = DataDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("EndpointDescriptor: %s\n"), (LPCTSTR)CW2CT(DataDialogPtr-
>EndpointDescriptor->ToString));
            _tprintf(_T("NodeElement: %s\n"), (LPCTSTR)CW2CT(DataDialogPtr-
>NodeElement->ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for an OPC-UA data node
// (a Data Variable or a Property).

class procedure ShowDialog.Main;
var
  DataDialog: UADataDialog;
begin
  // Instantiate the dialog object
  DataDialog := CoUADataDialog.Create;
```

```
  DataDialog.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  DataDialog.UserPickEndpoint := True;

  DataDialog.ShowDialog(nil);

  // Display results
  WriteLn('EndpointDescriptor: ', DataDialog.EndpointDescriptor.ToString);
  WriteLn('NodeElement: ', DataDialog.NodeElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for an OPC-UA data node
// (a Data Variable or a Property).

class procedure ShowDialog.Main;
var
  DataDialog: OpcLabs_EasyOpcForms_TLB._UADataDialog;
begin
  // Instantiate the dialog object
  DataDialog := CoUADataDialog.Create;

  DataDialog.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  DataDialog.UserPickEndpoint := True;

  DataDialog.ShowDialog(nil);

  // Display results
  WriteLn('EndpointDescriptor: ', DataDialog.EndpointDescriptor.ToString);
  WriteLn('NodeElement: ', DataDialog.NodeElement.ToString);
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for an OPC-UA data node (a Data
Variable or a Property).

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim DataDialog As New UADataDialog
    DataDialog.EndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
    DataDialog.UserPickEndpoint = True

    Dim DialogResult
    DialogResult = DataDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & "EndpointDescriptor: " & DataDialog.EndpointDescriptor &
vbCrLf
    OutputText = OutputText & "NodeElement: " & DataDialog.NodeElement & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for an OPC-UA data node (a Data
Variable or a Property).

Option Explicit

Const DialogResult_OK = 1

Dim DataDialog: Set DataDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UADataDialog")
DataDialog.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
DataDialog.UserPickEndpoint = True

Dim dialogResult: dialogResult = DataDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo "EndpointDescriptor: " & DataDialog.EndpointDescriptor
WScript.Echo "NodeElement: " & DataDialog.NodeElement
```

Multi-selection

When you set the MultiSelect property of the UADataDialog to true, the dialog will allow the user to select any number of OPC-UA nodes. The list below the folders and data nodes (labeled "Selected nodes") contains the set of nodes that the user has selected in the dialog. The user can freely add nodes to this list, or remove them. The selected set is carried over to next invocation of the dialog, unless you change it from the code.

In the multi-selection mode, the set of nodes selected on input (if any) is in the NodeDescriptors property. On the output, the dialog fills the information about selected nodes into the NodeElements property (and updated the NodeDescriptors property as well). If UserPickEndpoint is true, each node may come from a different server. In this case, the EndpointDescriptors array contains the server endpoints for each node in NodeDescriptors, with corresponding indices in both arrays.

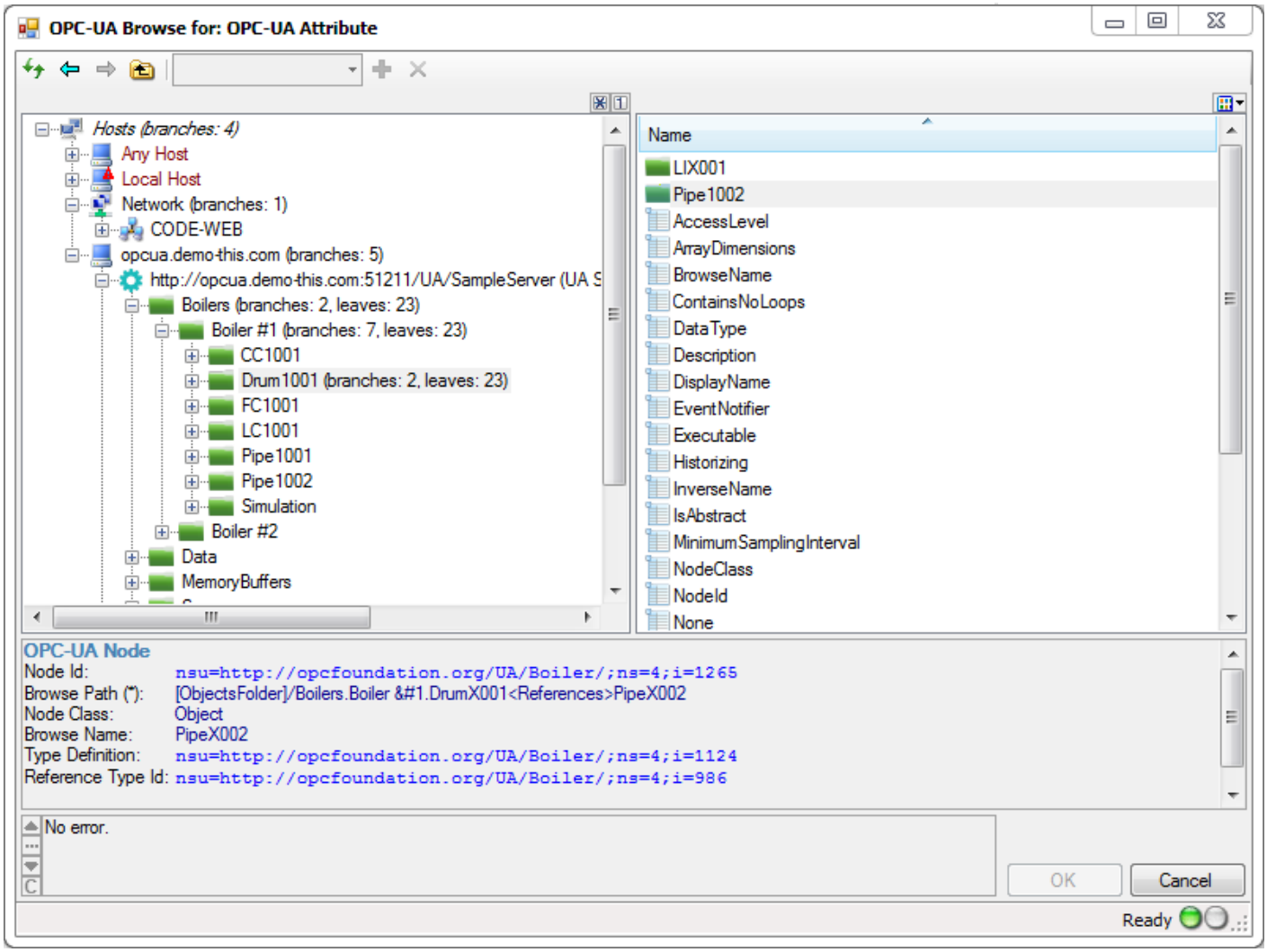
6.1.1.3.4 Generic OPC-UA Browsing Dialog

Icon: 

With UABrowseDialog, your application can integrate a dialog with various OPC-UA elements from which the user can select. This dialog can be configured to serve many different purposes.

Here is an example of the generic OPC-UA browsing dialog in action:

In This Topic

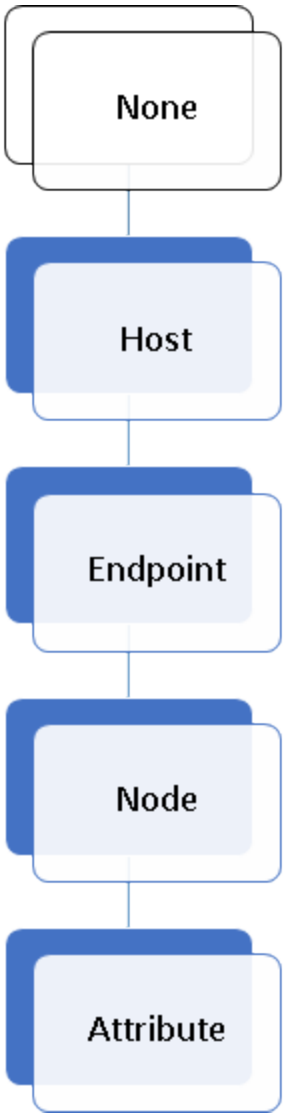


The way the dialog operates is controlled by two main properties:

- Mode.AnchorElementType determines the part of the selection that is given on input and cannot be changed by the user.
- Mode.SelectElementType determines the type of the element that the user needs to select in order to finalize the dialog.

Values of these properties can be selected from the UAElementType enumeration, which has members for various types of elements that you encounter when working with OPC-UA.

The following chart shows a hierarchy of element types that you can choose from:



For example, let' say that you set Mode.AnchorElementType to Endpoint, and Mode.SelectElementType to Attribute. This will cause the dialog to allow the user to browse for an OPC-UA Node on a given server, and then for an OPC-UA attribute on that node.

In this case, before you run the dialog, you need to provide it with values for the InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor property, because those define your "anchor" element (Endpoint) that the user cannot change. The dialog will only allow the user to finalize it (besides cancelling) after an OPC-UA attribute is fully selected, because that is your Mode.SelectElementType. After the dialog is successfully finalized, the information about the user's choice will be available in the Outputs.CurrentNodeElement.NodeElement and InputsOutputs.CurrentNodeDescriptor.AttributeId properties.

Note that in addition to the "minimal" scenario described above, you can also pre-set the initial node or attribute, using the InputsOutputs.CurrentNodeDescriptor.NodeDescriptor or InputsOutputs.CurrentNodeDescriptor.AttributeId properties, and after the selection is made, these properties will be updated to the new selection as well. This way, if you run the dialog again with the same value, the initial selection will be where the user has left it the last time the dialog was run.

Obviously, the chosen Mode.SelectElementType must be a child or indirect ancestor of chosen Mode.AnchorElementType in the hierarchy.

C++

```
// This example shows how to let the user browse for an OPC-UA node.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UABrowseDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UABrowseDialogPtr BrowseDialogPtr(__uuidof(UABrowseDialog));

            //
            BrowseDialogPtr->InputsOutputs->CurrentNodeDescriptor->EndpointDescriptor-
>Host = L"opcua.demo-this.com";
            BrowseDialogPtr->Mode->AnchorElementType = UAElementType_Host;

            //
            DialogResult dialogResult = BrowseDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(BrowseDialogPtr->Outputs-
>CurrentNodeElement->NodeElement->ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for an OPC-UA node.

class procedure ShowDialog.Main;
var
  BrowseDialog: UABrowseDialog;
begin
  // Instantiate the dialog object
  BrowseDialog := CoUABrowseDialog.Create;
  BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host :=
'opcua.demo-this.com';
  BrowseDialog.Mode.AnchorElementType := UAElementType_Host;

  BrowseDialog.ShowDialog(nil);

  // Display results
  WriteLn(BrowseDialog.Outputs.CurrentNodeElement.NodeElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for an OPC-UA node.

class procedure ShowDialog.Main;
var
  BrowseDialog: OpcLabs_EasyOpcForms_TLB._UABrowseDialog;
begin
  // Instantiate the dialog object
  BrowseDialog := CoUABrowseDialog.Create;
  BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host :=
'opcua.demo-this.com';
  BrowseDialog.Mode.AnchorElementType := UAElementType_Host;

  BrowseDialog.ShowDialog(nil);

  // Display results
  WriteLn(BrowseDialog.Outputs.CurrentNodeElement.NodeElement.ToString);
end;
```

PHP

```
// This example shows how to let the user browse for an OPC-UA node.

$UAElementType_Host = 1;

$BrowseDialog = new COM("OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog");
$BrowseDialog->InputsOutputs->CurrentNodeDescriptor->EndpointDescriptor->Host =
"opcua.demo-this.com";
$BrowseDialog->Mode->AnchorElementType = $UAElementType_Host;

printf("%d\n", $BrowseDialog->ShowDialog);

// Display results
printf("%s\n", $BrowseDialog->Outputs->CurrentNodeElement->NodeElement);
```

Python

```
# This example shows how to let the user browse for an OPC-UA node.

import win32com.client

UAElementType_Host = 1

browseDialog =
win32com.client.Dispatch('OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog')
browseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host = "opcua.demo-
this.com"
browseDialog.Mode.AnchorElementType = UAElementType_Host

print(browseDialog.ShowDialog())

# Display results
print(browseDialog.Outputs.CurrentNodeElement.NodeElement)
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for an OPC-UA node.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim BrowseDialog As New UABrowseDialog
    BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host =
"opcua.demo-this.com"

    Dim DialogResult
    DialogResult = BrowseDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & BrowseDialog.outputs.CurrentNodeElement.NodeElement &
vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for an OPC-UA node.

Option Explicit

Const DialogResult_OK = 1

Const UAElementType_Host = 1

Dim BrowseDialog: Set BrowseDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog")
BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host = "opcua.demo-
this.com"
BrowseDialog.Mode.AnchorElementType = UAElementType_Host

Dim dialogResult: dialogResult = BrowseDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo BrowseDialog.Outputs.CurrentNodeElement.NodeElement
```

Multi-selection

It is also possible to configure the dialog for a multi-selection. In this mode, the user can select zero, one, or more nodes. In order to enable the multi-select mode, set the Mode.MultiSelect property to true. In the multi-select mode, the initial set of the selected nodes (when the dialog is first displayed to the user) is given by the contents of the InputOutputs.SelectionDescriptors collection. When the user makes the selection and accepts it by closing the dialog,

this collection is updated, and also, all information about the selected nodes is placed to the Outputs.SelectionElements collection.

VBScript

```
Rem This example shows how to let the user browse for multiple OPC-UA nodes.

Option Explicit

Const DialogResult_OK = 1

Const UAElementType_Host = 1

Dim BrowseDialog: Set BrowseDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog")
BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host = "opcua.demo-
this.com"
BrowseDialog.Mode.AnchorElementType = UAElementType_Host
BrowseDialog.Mode.MultiSelect = True


Dim dialogResult: dialogResult = BrowseDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
Dim SelectionElements: Set SelectionElements = BrowseDialog.Outputs.SelectionElements
Dim i: For i = 0 To SelectionElements.Count - 1
    Dim Element: Set Element = SelectionElements(i)
    WScript.Echo "SelectionElements(" & i & "): " & Element.NodeElement
Next
```

Other settings

There are also ways to control some finer aspects of the dialog. For example, the Mode.ShowListBranches property (defaults to true) controls whether the branches of the tree are also displayed in the list view.

When you set the Simulated property of the dialog to true, the dialog will provide its contents from a pre-defined, simulated view of the world, with fake networks, computers, OPC servers, and their contents. This can be useful for experimentation and testing, either by the developer during the design (right in Visual Studio), or by the end-user (if you expose this functionality in your application), when the environment is not accessible.

6.1.2 OPC Controls

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product

> Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

6.1.2.1 OPC Classic Controls

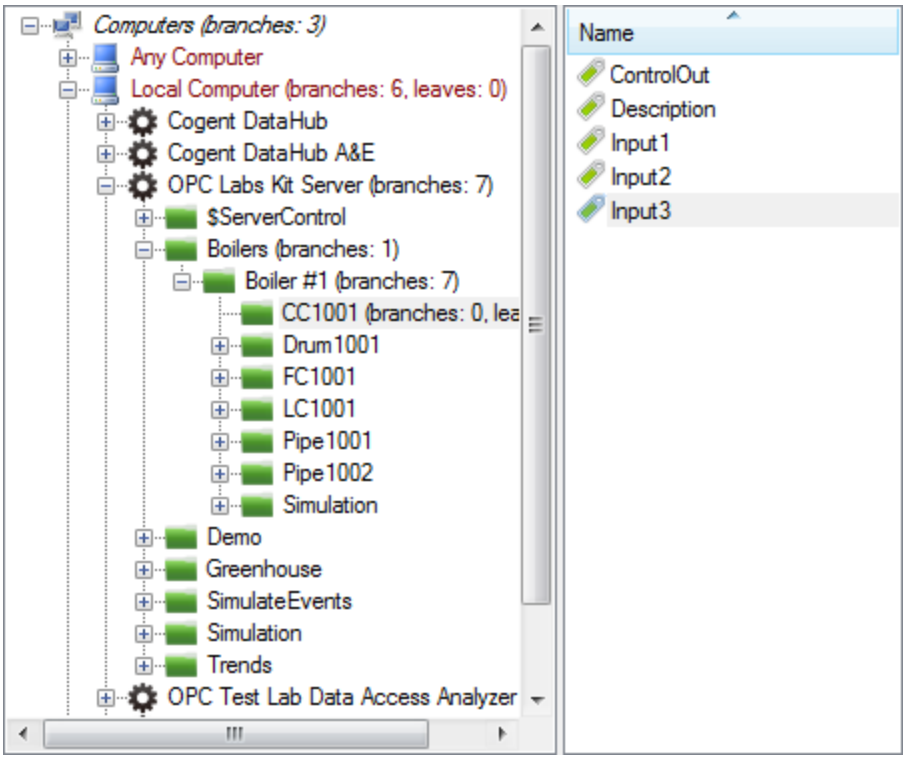
OPC Data Client.NET contains OPC-enabled .NET controls that can be placed onto your Windows Forms and configured and programmed to cooperate with other controls.

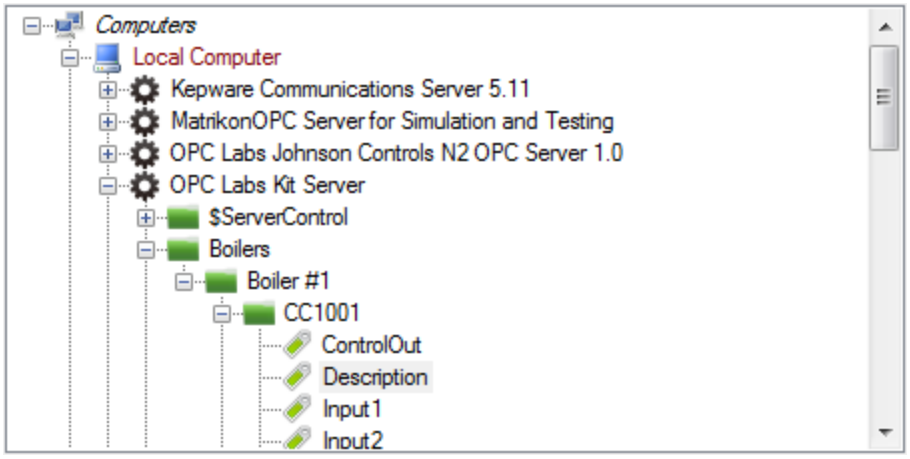
Articles in this section apply to OPC "Classic" DA and OPC XML-DA.

6.1.2.1.1 Generic OPC Classic Browsing Control

With OpcBrowseControl, your forms can integrate a control with various OPC nodes from which the user can select. This control can be configured to serve many different purposes.

Here are examples of the generic OPC browsing control in action:





The functionality and properties of the OpcBrowseControl are similar to that of OpcBrowseDialog, described earlier in this text. Please refer to the documentation of OpcBrowseDialog for details. Here are the major differences:

- There is no toolbar, no filter controls, no node information box, no error box, and no status bar. Events are provided (described further below) that allow you to implement similar functionality yourself, if needed.
- Changes you make in run-time to relevant values of the Inputs and InputsOutputs properties are immediately reflected in the control. Analogically, actions by the user that change the current node or the selection set are immediately projected into the InputsOutputs and Outputs properties.
- When multi-selection mode is enabled, the nodes are always selected directly in the list view, and not in a separate list. For this reason, the nodes in a multi-selection must all belong under a single parent branch.
- If only a tree view is displayed, the tree will also contain the leaves.
- If only a list view is displayed, the list will also always contain the branches, regardless of the Mode.ShowListBranches property value.
- When a node is selected in the tree view, a first node in the list view is not automatically focused.
- In the design-time mode, the control always shows simulated data, even if the Simulated property is set to false.

Using the Kind property, the browsing control can be configured to provide a tree view only (BrowseControlKinds.Tree), a list view only (BrowseControlKinds.List), or a combined tree view and list view (BrowseControlKinds.TreeAndList; this is the default).

The View property (of System.Windows.Forms.View enumeration type) controls how the list view items are displayed. Possible values are LargeIcon, Details (the default), SmallIcon, List, or Tile.

In order to achieve tight integration with other controls on your form, you can hook to events that the browsing control provides.

The CurrentNodeChanged event occurs when the current node changes. You can obtain the information about the new current node from InputsOutputs.CurrentNodeDescriptor and Outputs.CurrentNodeElement properties inside the event handler.

The SelectionChanged event is meant for multi-selection mode, and occurs when the selection set changes. You can obtain the information about the new selection set from InputsOutputs.SelectionDescriptors and Outputs.SelectionElements collections inside the event handler.

The NodeDoubleClick event occurs when a node is double-clicked. This is the current node, and therefore the information about it can be obtained in the same way as in the CurrentNodeChanged event handler, described above.

The BrowseFailure event indicates that an exception has occurred during browsing. The information about the exception is contained in the event arguments.

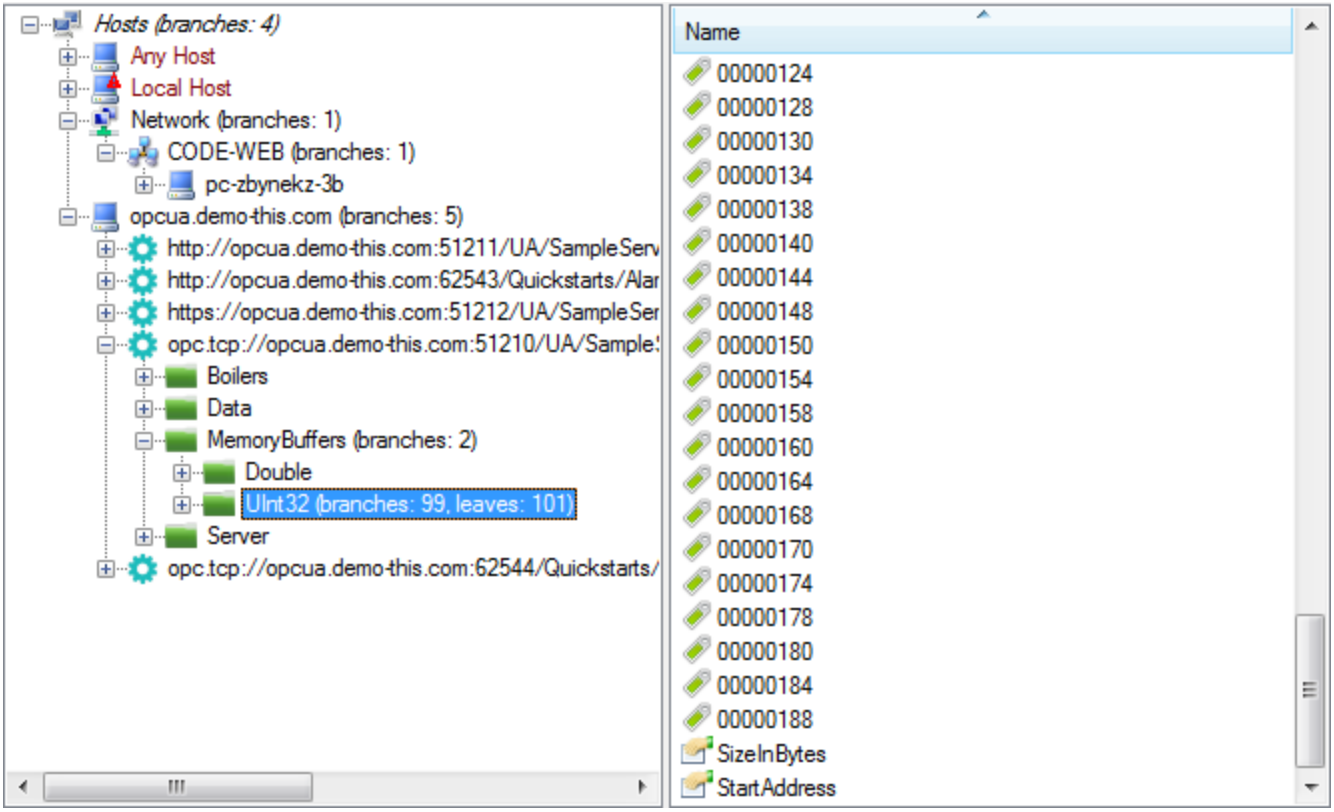
6.1.2.2 OPC-UA Controls

OPC Data Client-UA contains OPC-enabled .NET controls that can be placed onto your Windows Forms and configured and programmed to cooperate with other controls.

6.1.2.2.1 Generic OPC UA Browsing Control

With UABrowseControl, your forms can integrate a control with various OPC nodes from which the user can select. This control can be configured to serve many different purposes.

Here is an example of the generic OPC browsing control in action:



The functionality and properties of the UABrowseControl are similar to that of UABrowseControl, described earlier in this text. Please refer to the documentation of UABrowseControl for details. Here are the major differences:

- There is no toolbar, no filter controls, no node information box, no error box, and no status bar. Events are provided (described further below) that allow you to implement similar functionality yourself, if needed.
- Changes you make in run-time to relevant values of the Inputs and InputsOutputs properties are immediately reflected in the control. Analogically, actions by the user that change the current node or the selection set are immediately projected into the InputsOutputs and Outputs properties.
- When multi-selection mode is enabled, the nodes are always selected directly in the list view, and not in a separate list. For this reason, the nodes in a multi-selection must all belong under a single parent branch.
- If only a tree view is displayed, the tree will also contain the leaves.
- If only a list view is displayed, the list will also always contain the branches, regardless of the Mode.ShowListBranches property value.
- When a node is selected in the tree view, a first node in the list view is not automatically focused.

- In the design-time mode, the control always shows simulated data, even if the Simulated property is set to false.

Using the Kind property, the browsing control can be configured to provide a tree view only (BrowseControlKinds.Tree), a list view only (BrowseControlKinds.List), or a combined tree view and list view (BrowseControlKinds.TreeAndList; this is the default).

The View property (of System.Windows.Forms.View enumeration type) controls how the list view items are displayed. Possible values are LargeIcon, Details (the default), SmallIcon, List, or Tile.

In order to achieve tight integration with other controls on your form, you can hook to events that the browsing control provides.

The CurrentNodeChanged event occurs when the current node changes. You can obtain the information about the new current node from InputsOutputs.CurrentNodeDescriptor and Outputs.CurrentNodeElement properties inside the event handler.

The SelectionChanged event is meant for multi-selection mode, and occurs when the selection set changes. You can obtain the information about the new selection set from InputsOutputs.SelectionDescriptors and Outputs.SelectionElements collections inside the event handler.

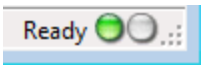
The NodeDoubleClick event occurs when a node is double-clicked. This is the current node, and therefore the information about it can be obtained in the same way as in the CurrentNodeChanged event handler, described above.

The BrowseFailure event indicates that an exception has occurred during browsing. The information about the exception is contained in the event arguments.

6.1.3 Common Functionality in Browsing Dialogs and Controls

The tree view in the browsing dialogs displays the number of node sub-branches and sub-leaves (when known) in the parentheses next to the node name.

The browsing dialogs also have a “traffic light” status indicator in the lower right corner, allowing for quick recognition of the status by the user.



The hierarchical browsing dialogs have Back and Forward buttons, allowing better navigation experience in some scenarios. The buttons behave similarly to those commonly found in Internet browsers.

Depending on the active node, the user can also invoke various context-dependent commands, as described in **Context Commands in Browsing (Section 6.1.3.1)**.

In many browse dialogs and controls, the user can use a Search box to enter the text to search for, choose various search options, and navigate in the tree and/or list to nodes that fulfill the search criteria: **Search in Browsing (Section 6.1.3.2)**.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product

> Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

6.1.3.1 Context Commands in Browsing

During the browsing, various nodes expose different functionality, depending on their type and settings. The node-specific functionality is made available using context commands, such as those on the node context menu (accessible through the right-click), or in the **Actions** pane of the browsing control.

For a list and description of these commands, depending on the domain, refer to one of the following chapters:

- **Context Commands in OPC Classic Browsing (Section 6.1.3.1.1)**
- **Context Commands in OPC UA Browsing (Section 6.1.3.1.2)**
- **Search in Browsing (Section 6.1.3.2)**

6.1.3.1.1 Context Commands in OPC Classic Browsing

Commands described in this article are available when browsing in the OPC Classic domain.

In This Topic

OPC Computer Node

Troubleshooting: Ping Computer Test

Makes a quick connectivity check to the selected computer. Note that the PING command may fail but the computer may still be accessible via OPC.

OPC Server Node

Duplicate Server Node

Creates a duplicate of an existing OPC server node.

Troubleshooting: Ping Computer Test

Makes a quick connectivity check to the selected computer. Note that the PING command may fail but the computer may still be accessible via OPC.

Troubleshooting: TCP Connect Test

This command attempts to open a TCP socket to the selected endpoint (without any OPC-specific actions or communication), in order to verify the network connectivity and the presence of the server or endpoint.

The command is only enabled on OPC XML-DA server nodes.

6.1.3.1.2 Context Commands in OPC UA Browsing

Commands described in this article are available when browsing in the OPC Unified Architecture (OPC UA) domain.

In This Topic

OPC UA Root Node

Settings: Endpoint Selection Policy

Configures the default endpoint selection policy, valid on endpoints which do not specify their endpoint selection policy explicitly.

Settings: Certificate Acceptance Policy

Globally configures how other-party application certificates are accepted.

Application: View Instance Certificate

Locates, retrieves and displays the application instance certificate currently used by the application.

OPC UA Host Node

Enable Local Discovery

When set, connects to the OPC UA Local Discovery Server (LDS) on the host, and shows the applications and/or discovery endpoints registered on the host.

Troubleshooting: Ping Host Test

Makes a quick connectivity check to the selected host. Note that the PING command may fail but the host may still be accessible via OPC.

Browse As: Applications

Groups the information returned from the OPC UA browsing by OPC UA applications.

Browse As: Discovery Endpoints

Shows the individual OPC UA discovery endpoints returned from the OPC UA browsing.

Discovery Endpoints Display: URL, Application Name

The discovery endpoints are displayed using their URL, with an appended application name in parenthesis. For example: opc.tcp://opcua.demo-this.com:51210/UA/SampleServer (UA Sample Server) .

Discovery Endpoints Display: URL

The discovery endpoints are displayed using their URL only. For example: opc.tcp://opcua.demo-

this.com:51210/UA/SampleServer .

OPC UA Application Node (from local discovery)

Browse As: Discovery Endpoints

Shows the individual OPC UA discovery endpoints returned from the OPC UA browsing.

Browse As: Session Endpoints

Shows session endpoints available for the discovery endpoint or application underneath its node.

Discovery Endpoints Display: URL

The discovery endpoints are displayed using their URL. For example: opc.tcp://opcua.demo-this.com:51210/UA/SampleServer .

Discovery Endpoints Display: Scheme

The session endpoints are displayed using their scheme. For example: http: .

Session Endpoints Display: URL, Policy Specifier

The session endpoints are displayed using their URL (host part may be replaced by "..." if it clear from some ancestor node), and a policy specifier. Example: http://...:51211/UA/SamplesServer - [SignAndEncrypt:Basic128Rsa15:] .

Session Endpoints Display: Scheme, Policy Specifier

The session endpoints are displayed using their scheme, and a policy specifier. For example: http: - [SignAndEncrypt:Basic128Rsa15:] .

Session Endpoints Display: Policy, Mode, Profile

The session endpoints are displayed using their policy name and security mode, with an appended profile name in parenthesis. For example: Basic128Rsa15 - SignAndEncrypt (soaphttp-wssc-uaxml-uabinary) .

OPC UA Application Node (from network or global discovery)

(none)

OPC UA Endpoint Node

Duplicate Endpoint Node

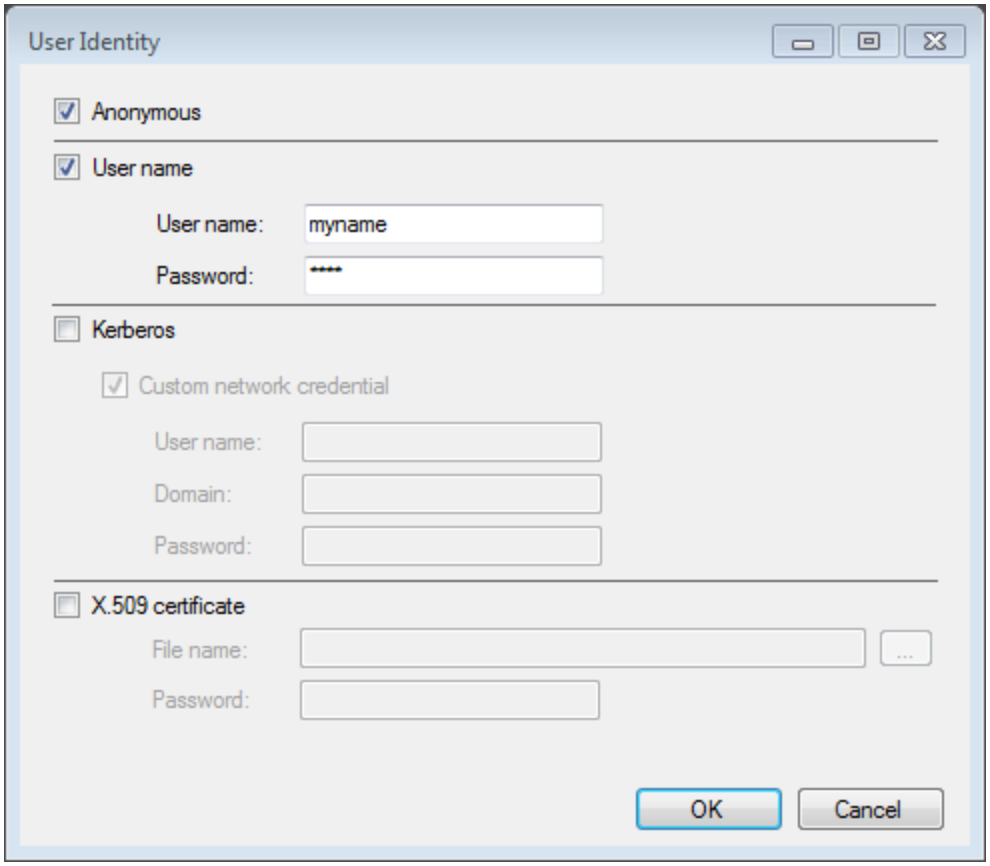
Creates a duplicate of an existing OPC UA endpoint node. By creating a duplicate, you may set parameters of the endpoint (such as endpoint selection policy, or user identity) differently from the original node.

Treat As GDS

When set, treats the endpoint node as OPC UA Global Discovery Server (GDS). In this mode, the applications registered in the GDS are shown underneath the endpoint node (instead of normal OPC UA address space), and additional commands are made available on the node, for GDS registration, Certificate Managemer (CM) operations, etc.

Endpoint Parameters: User Identity

This command allows the user to specify the identity (or identities) that should be used during the connection attempt(s) to the server.



Endpoint Parameters: Endpoint Selection Policy

Allows the user to influence the endpoint selection, specifically for the selected OPC UA endpoint node.

Troubleshooting: Ping Host Test

Makes a quick connectivity check to the selected host. Note that the PING command may fail but the host may still be accessible via OPC.

Troubleshooting: TCP Connect Test

This command attempts to open a TCP socket to the selected endpoint (without any OPC-specific actions or communication), in order to verify the network connectivity and the presence of the server or endpoint.

OPC UA Endpoint Node (discovery endpoint)

Browse As: Session Endpoints

Shows session endpoints available for the discovery endpoint or application underneath its node.

Browse As: Address Space

Shows the address space of the OPC UA server underneath the discovery endpoint node.

Session Endpoints Display: URL, Policy Specifier

The session endpoints are displayed using their URL (host part may be replaced by "..." if it clear from some ancestor node), and a policy specifier. Example: http://...:51211/UA/SamplesServer - [SignAndEncrypt:Basic128Rsa15:] .

Session Endpoints Display: Scheme, Policy Specifier

The session endpoints are displayed using their scheme, and a policy specifier. For example: http: - [SignAndEncrypt:Basic128Rsa15:] .

Session Endpoints Display: Policy, Mode, Profile

The session endpoints are displayed using their policy name and security mode, with an appended profile name in parenthesis. For example: Basic128Rsa15 - SignAndEncrypt (soaphttp-wssc-uaxml-uabinary) .

OPC UA Endpoint Node (with Treat As GDS)

GDS Registration: Register This Client

Creates an application registration in the GDS, assigning it a new application ID. Existing registrations with the same application URI are removed first.

GDS Registration: Update This Client Registration

Updates an application registration in the GDS, keeping its application ID. A new registration is created if the application is not yet registered in the GDS. Preexisting registrations with the same application URI are removed.

GDS Registration: Unregister This Client

Removes an application registration from the GDS, if it exists. All existing registrations for the application URI are removed.

GDS Registration: Find This Client Registrations

Finds and displays all registrations for this application URI in the GDS.

Certificate Manager (CM): Obtain New Certificate

Obtains a new application instance certificate from the certificate manager, and stores it for subsequent usage.

Certificate Manager (CM): Refresh Own Trust Lists

Retrieves the current trust lists for the application from the certificate manager, and refreshes own certificate stores

accordingly.

OPC UA Network Node

Enable Network Discovery

When set, connects to the OPC UA Local Discovery Server with Multicast Extensions (LDS-ME) on the host, and shows the hosts, applications and/or discovery endpoints on the host's network.

Troubleshooting: Ping Host Test

Makes a quick connectivity check to the selected host. Note that the PING command may fail but the host may still be accessible via OPC.

Browse As: Hosts

Groups the information returned from the OPC UA network browsing by hosts.

Browse As: Applications

Groups the information returned from the OPC UA browsing by OPC UA applications.

Browse As: Discovery Endpoints

Shows the individual OPC UA discovery endpoints returned from the OPC UA browsing.

Discovery Endpoints Display: URL, Application Name

The discovery endpoints are displayed using their URL, with an appended application name in parenthesis. For example: opc.tcp://opcua.demo-this.com:51210/UA/SampleServer (UA Sample Server) .

Discovery Endpoints Display: URL

The discovery endpoints are displayed using their URL only. For example: opc.tcp://opcua.demo-this.com:51210/UA/SampleServer .

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

6.1.3.2 Search in Browsing

The Search box in the browsing dialogs and controls allows the user to enter text to search for, choose various search options, and navigate in the tree and/or list to nodes that fulfill the search criteria.

In order to find a node, type the text you want to find into the Search box (over the "Find..." text, if the box is otherwise

empty). The Search box contains following icons:

- **Magnifier**: Clicking on it will search for (and highlight) all matching nodes in the tree/list. The currently selected node is not moved.
- **Left arrow** (***Shift+F3***): Searches for and highlights all matching nodes, and select the previous matching node. When typing, ***Shift+Enter*** has the same meaning.
- **Right arrow** (***F3***): Searches for and highlights all matching nodes, and select the next matching node. When typing, ***Enter*** has the same meaning.
- **Blue cross**: Clears the text in the search box, and removes all node highlighting.

The "*x*/*y*" indicator tells the user how many matching nodes have been found (*y*) and the position of the current node in them (*x*).

The Search box has a drop-down menu with additional commands. You can invoke the drop-down menu by clicking on the down-arrow box, or by pressing a ***down arrow*** key whil typing. The drop-down menu is divided into 3 parts:

1. **Find**. Contains a list of 10 recently used search texts, together with an icon indicating their search type. Selecting any of these commands repeats the corresponding search.
2. **Find options**. Allows to influence how the search is being performed. Available options are Starts with/End with/Contains/Does not contain/Precise text, Match case, and Regular expression.
3. **Nodes**. Contains a list of all nodes that fulfill the search criteria. Selecting any of these commands sets the corresponding node as a currently selected ndoe in the tree/list.

6.1.4 Unsolicited User Interaction

Introduction

OPC Data Client components may, under certain circumstances, display a user interface "by itself", without you having to make a specific method call. Such unsolicited user interaction only happens when the current process is running in user interactive mode, as reported by the Environment.UserInteractive Property. In non-interactive mode, the components assume reasonable default user responses. In addition, you can turn off specific user interactions by setting corresponding parameters in the code.

Usually, these are notifications or confirmations related to conditions that cannot be anticipated upfront, such as the certificate exchange in the OPC UA connection process. As OPC Data Client controls the connections and diconnections automatically, and when subscriptions are in effect, also makes re-connections, you - as a developer - cannot generally predict when such user interaction may be needed.

Interaction Types

There are following situations that may trigger an unsolicited user interaction related to OPC Unified Architecture (OPC UA) operations:

- Issues while checking or creating an application instance certificate. These are rare, and usually only happen when

the application runs for the first time. These can be turned off or on by the AllowClientCertificatePrompt Property of the EngineParameters Property of EasyUAClient.SharedParameters Property. See also **Providing OPC UA Client Instance Certificate (Section 4.13.1.1)**.

- When HTTPS communication is used, failed validation of server's HTTPS certificate. The user is prompted whether he/she wants to accept the certificate anyway. This prompt can be turned off or on by the AllowUserAcceptCertificate Property of HttpsCertificateAcceptancePolicy Property of EngineParameters Property of EasyUAClient.SharedParameters Property. See also **Trusting OPC UA Server HTTPS Certificate (Section 4.13.1.3)**.
- Failed validation of OPC UA server's instance certificate. The user is prompted whether he/she wants to accept the certificate anyway. This prompt can be turned off or on by the AllowUserAcceptCertificate Property of CertificateAcceptancePolicy Property of EngineParameters Property of EasyUAClient.SharedParameters Property. Alternatively, the certificate acceptance policy can be overridden for a specific endpoint by setting it to a non-null value in CertificateAcceptancePolicy Property. See also **Trusting OPC UA Server Instance Certificate (Section 4.13.1.2)**.
- The effective host name in endpoint URL returned by the server does not match any of the domain names in the server certificate. This may be an indication of a spoofing attempt. The user is prompted whether he/she wants to allow the endpoint anyway. This prompt can be turned off or on by the AllowEndpointDomainPrompt Property of UAClientSessionParameters Class.

There is no unsolicited user interaction related to OPC Classic operations.

Interaction Providers

The way OPC Data Client presents the unsolicited user interaction depends on the environment your application is running in. There is an *interaction provider* for each such supported environment. OPC Data Client interrogates the available interaction providers to see which one should perform the interaction, selects the appropriate interaction provider and then uses it to actually communicate with the user. Following interaction providers are available:

- **Console Interaction Provider**. Used in console applications.
- **Windows Forms Interaction Provider (not in .NET Standard)**. Used on Microsoft Windows in applications that have windowed user interface.
- **Null Interaction Provider**. This provider is used if OPC Data Client cannot find any other interaction provider for the current environment.

Each interaction provider (except for the Null Interaction Provider) can be turned on or off, and may have additional configurable parameters.

Interaction and Blocking

All unsolicited user interaction (with exception of rare messages and confirmations related to client application instance certificate checks) has a timeout associated with it, and a default user response is assumed should the user not respond in time. This means that under normal circumstances, the current activity cannot be blocked indefinitely if the user fails to respond. In fact, this behavior also guarantees that the activity will eventually proceed even if the Environment.UserInteractive Property incorrectly claimed that the process is running in user interactive mode. The timeout is configurable by the AcceptNotificationTimeout Property in the UAUserInteractionParameters Class. In order to obtain or modify this parameter, access the UserInteractionParameters Property of EngineParameters Property of EasyUAClient.SharedParameters Property.

Interaction and Licensing

OPC Data Client components do not display, by itself, any visual warning when there is a problem with the license key, or when the trial license runtime expires. Instead, such problems are reported as other errors, using the channels for the programming model used. For example, in **Imperative Programming Model (Section 5.1)**, the single-argument method throws an exception, a multiple-argument method returns the error in the Exception Property of the OperationResult Class, or with subscriptions, the error is reported in the Exception Property of the OperationEventArgs Class. It is up to the calling code to handle such situations.

6.1.4.1 Console Interaction

Introduction

In This Topic

The Console Interaction Provider uses standard console, available on most systems, for user input and output.

OPC Data Client uses the Console Interaction Provider whenever the Environment.UserInteractive Property returns true, and the **Windows Forms Interaction (Section 6.1.4.2)** Provider does not take precedence. Note that is is also possible to use the **Windows Forms Interaction (Section 6.1.4.2)** in console applications **(not in .NET Standard)**, by referencing the **EasyOpcForms** assembly in the console-based project.

Example Interactions

The example below triggers the component to ask the user whether he/she wants to accept the server's HTTPS certificate.

C#

```
// This example shows how in a console application, the user is asked to accept a server HTTPS certificate.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    partial class AcceptCertificate
    {
        public static void Https()
        {
            // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";

            // Instantiate the client object.
            var client = new EasyUAClient();

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// This example shows how in a console application, the user is asked to accept a server HTTPS certificate.
```

```
class procedure AcceptCertificate.Https;
var
  AttributeData: _UAAttributeData;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  EndpointDescriptor: string;
begin
  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
  ClientConfiguration.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

  // Define which server we will work with.
  EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;
  try
    // Obtain attribute data.
    // The component automatically triggers the necessary user interaction during the first operation.
    AttributeData := Client.Read(EndpointDescriptor, 'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results.
  WriteLn('Value: ', AttributeData.Value);
  WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
  WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
  WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);

  FreeAndNil(ClientConfiguration);
end;
```

VB.NET

```
' This example shows how in a console application, the user is asked to accept a server HTTPS certificate.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Interaction
    Partial Friend Class AcceptCertificate
        Public Shared Sub Https()

            ' Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

The program output may look like this:

The example below triggers the component to ask the user whether he/she wants to accept the server's instance certificate.

C#

```
// This example shows how in a console application, the user is asked to accept a server instance certificate.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Engine;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    partial class AcceptCertificate
    {
        public static void Instance()
        {
            // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = UAMessageSecurityModes.Secure;

            // Instantiate the client object.
            var client = new EasyUAClient();

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// This example shows how in a console application, the user is asked to accept a server instance certificate.

class procedure AcceptCertificate.Instance;
var
  Arguments: OleVariant;
  AttributeData: _UAAttributeData;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  EndpointSelectionPolicy: _UAEndpointSelectionPolicy;
  ReadArguments: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
  ClientConfiguration.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

  // Define which server we will work with.
  ReadArguments := CoUAReadArguments.Create;
  ReadArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  // Require secure connection, in order to enforce the certificate check.
  EndpointSelectionPolicy := CoUAEndpointSelectionPolicy.Create;
  EndpointSelectionPolicy.AllowedMessageSecurityModes := UAMessageSecurityModes_Secure;
  ReadArguments.EndpointDescriptor.EndpointSelectionPolicy := EndpointSelectionPolicy;
  ReadArguments.NodeDescriptor.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=10853';

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := ReadArguments;

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;
```

```
  // Obtain attribute data.
  // The component automatically triggers the necessary user interaction during the first operation.
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

  Result := IInterface(Results[0]) as _UAAttributeDataResult;
  if Result.Succeeded then
  begin
    AttributeData := Result.AttributeData;
    // Display results.
    WriteLn('Value: ', AttributeData.Value);
    WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
    WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
    WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);
  end
  else
    WriteLn('*** Failure: ', Result.ErrorMessageBrief);

  FreeAndNil(ClientConfiguration);
end;
```

VB.NET

```
' This example shows how in a console application, the user is asked to accept a server instance certificate.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Engine
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Interaction
    Partial Friend Class AcceptCertificate
        Public Shared Sub Instance()

            ' Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = UAMessageSecurityModes.Secure

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

The program output may look like this:

The example below triggers the component to ask the user whether he/she wants to accept an instance certificate whose domain does not match the URL used to connect to the server.

C#

```
// This example shows how in a console application, the user is asked to allow a server instance certificate with
// mismatched domain name.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
```

```
{
    class AllowEndpointDomain
    {
        public static void Main1()
        {
            // Define which server we will work with.
            // Note that extra '.' at the end of the domain name. For the purpose of this example, it allows us to address
            // the same domain, but cause a mismatch with what the names that are listed in the server instance certificate.
            UAEndpointDescriptor endpointDescriptor = "opc.tcp://opcua.demo-this.com.:51210/UA/SampleServer";

            // Instantiate the client object.
            var client = new EasyUAClient()
            {
                // Enforce the endpoint domain check.
                Isolated = true,
                IsolatedParameters = {SessionParameters = {CheckEndpointDomain = true}}
            };

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// This example shows how in a console application, the user is asked to allow a server instance certificate with
// mismatched domain name.

class procedure AllowEndpointDomain.Main;
var
  AttributeData: _UAAttributeData;
  Client: _EasyUAClient;
  EndpointDescriptor: string;
begin
  // Define which server we will work with.
  // Note that extra '.' at the end of the domain name. For the purpose of this example, it allows us to address
  // the same domain, but cause a mismatch with what the names that are listed in the server instance certificate.
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com.:51210/UA/SampleServer';

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;
  // Enforce the endpoint domain check.
  Client.Isolated := true;
  Client.IsolatedParameters.SessionParameters.CheckEndpointDomain := true;

  try
    // Obtain attribute data.
    // The component automatically triggers the necessary user interaction during the first operation.
    AttributeData := Client.Read(EndpointDescriptor, 'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results.
  WriteLn('Value: ', AttributeData.Value);
  WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
  WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
  WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);
end;
```

VB.NET

```
' This example shows how in a console application, the user is asked to allow a server instance certificate with
' mismatched domain name.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.Engine
```

```
Namespace UADocExamples.Interaction
    Friend Class AllowEndpointDomain
        Public Shared Sub Main1()

            ' Define which server we will work with.
            ' Note that extra '.' at the end of the domain name. For the purpose of this example, it allows us to address
            ' the same domain, but cause a mismatch with what the names that are listed in the server instance certificate.
            Dim endpointDescriptor As UAEndpointDescriptor = "opc.tcp://opcua.demo-this.com.:51210/UA/SampleServer"

            ' Instantiate the client object.
            Dim client = New EasyUAClient() With _
            { _
                .Isolated = True, _
                .IsolatedParameters = New EasyUAAdaptableParameters() With _
                { _
                    .SessionParameters = New UASmartSessionParameters() With _
                    { _
                        .CheckEndpointDomain = True _
                    } _
                } _
            }

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how in a console application, the user is asked to allow a server instance certificate with
Rem mismatched domain name.

Option Explicit

' Define which server we will work with.
' Note that extra '.' at the end of the domain name. For the purpose of this example, it allows us to address
' the same domain, but cause a mismatch with what the names that are listed in the server instance certificate.
Dim endpointDescriptor: endpointDescriptor = "opc.tcp://opcua.demo-this.com.:51210/UA/SampleServer"

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
' Enforce the endpoint domain check.
Client.Isolated = True
Client.IsolatedParameters.SessionParameters.CheckEndpointDomain = True

' Obtain attribute data.
' The component automatically triggers the necessary user interaction during the first operation.
On Error Resume Next
Dim AttributeData: Set AttributeData = Client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Value: " & AttributeData.Value
WScript.Echo "ServerTimestamp: " & AttributeData.ServerTimestamp
WScript.Echo "SourceTimestamp: " & AttributeData.SourceTimestamp
WScript.Echo "StatusCode: " & AttributeData.StatusCode

' Example output:
'
'OPC-UA Endpoint Domain Mismatch
'The effective host name in endpoint URL returned by the server does not match any of the domain names in the server certificate.
'Endpoint URL as returned by the server: opc.tcp://opcua.demo-this.com.:51210/UA/SampleServer
'The server certificate is for following domain names or IP addresses: opcua.demo-this.com
'This may be an indication of a spoofing attempt. Do you want to allow the endpoint anyway? [Y/n]: Y
'Value: -1.285897E+14
'ServerTimestamp: 11/28/2019 1:34:23 PM
'SourceTimestamp: 11/28/2019 1:34:23 PM
'StatusCode: Good
```

The program output may look like this:

Configuration

You can configure parameters of the Console Interaction by finding UAConsoleInteractionPluginParameters in the shared plug-ins configurations, and changing properties of the object obtained.

The example below configures the component to use uncolorized console output, and then triggers some user interaction.

C#

```
// Shows how to configure the OPC UA Console Interaction plug-in by turning off the output colorization.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.Console.Interaction;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Engine;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    partial class ConsoleInteraction
    {
        public static void ColorizeOutput()
        {
            // Configure the shared plug-in.

            // Find the parameters object of the plug-in.
            ConsoleInteractionParameters consoleInteractionPluginParameters =
                EasyUAClient.SharedParameters.PluginConfigurations.Find<ConsoleInteractionParameters>();
            Debug.Assert(consoleInteractionPluginParameters != null);
            // Change the parameter.
            consoleInteractionPluginParameters.ColorizeOutput = false;


            // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = new UAEndpointSelectionPolicy(UAMessageSecurityModes.Secure);

            // Instantiate the client object.
            var client = new EasyUAClient();

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// Shows how to configure the OPC UA Console Interaction plug-in by turning off the output colorization.

class procedure ConsoleInteraction.ColorizeOutput;
var
  Arguments: OleVariant;
  AttributeData: _UAAttributeData;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  ConsoleInteractionParameters: _ConsoleInteractionParameters;
```

```
  EndpointSelectionPolicy: _UAEndpointSelectionPolicy;
  ReadArguments: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  // Configure the shared plug-in.

  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Find the parameters object of the plug-in.
  ConsoleInteractionParameters :=
IUnknown(ClientConfiguration.SharedParameters.PluginConfigurations.Find('OpcLabs.BaseLib.Console.Interaction.ConsoleInteractionParameters'))
as _ConsoleInteractionParameters;
  // Change the parameter.
  ConsoleInteractionParameters.ColorizeOutput := False;

  // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
  ClientConfiguration.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

  // Define which server we will work with.
  ReadArguments := CoUAReadArguments.Create;
  ReadArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  // Require secure connection, in order to enforce the certificate check.
  EndpointSelectionPolicy := CoUAEndpointSelectionPolicy.Create;
  EndpointSelectionPolicy.AllowedMessageSecurityModes := UAMessageSecurityModes_Secure;
  ReadArguments.EndpointDescriptor.EndpointSelectionPolicy := EndpointSelectionPolicy;
  ReadArguments.NodeDescriptor.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=10853';

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := ReadArguments;

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;

  // Obtain attribute data.
  // The component automatically triggers the necessary user interaction during the first operation.
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

  Result := IInterface(Results[0]) as _UAAttributeDataResult;
  if Result.Succeeded then
  begin
    AttributeData := Result.AttributeData;
    // Display results.
    WriteLn('Value: ', AttributeData.Value);
    WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
    WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
    WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);
  end
  else
    WriteLn('*** Failure: ', Result.ErrorMessageBrief);

  FreeAndNil(ClientConfiguration);
end;
```

VB.NET

```
' Shows how to configure the OPC UA Console Interaction plug-in by turning off the output colorization.

Imports OpcLabs.BaseLib.Console.Interaction
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Engine
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Interaction
    Friend Class ConsoleInteraction
        Public Shared Sub ColorizeOutput()

            ' Configure the shared plug-in.

            ' Find the parameters object of the plug-in.
            Dim consoleInteractionPluginParameters =
                EasyUAClient.SharedParameters.PluginConfigurations.Find(Of ConsoleInteractionParameters)()
            Debug.Assert(consoleInteractionPluginParameters IsNot Nothing)
            ' Change the parameter.
            consoleInteractionPluginParameters.ColorizeOutput = False

            ' Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = New UAEndpointSelectionPolicy(UAMessageSecurityModes.Secure)

            ' Instantiate the client object.
            Dim client = New EasyUAClient()
```

```
            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

The program output may look like this:

6.1.4.2 Windows Forms Interaction

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

In This Topic

Introduction

The Windows Forms Interaction Provider uses windowed user interface on Microsoft Windows operating systems for user input and output.

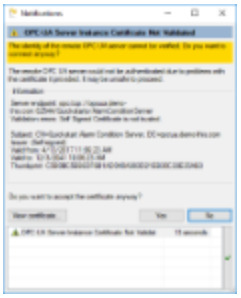
OPC Data Client uses the Windows Forms Interaction Provider whenever the Environment.UserInteractive Property returns true, and the **OpcLabs.EasyOpcForms** assembly can be loaded (which is usually achieved by simply referencing it in your project).

Despite the "Windows Forms" in its name, this interaction provider can also be used in applications that are not based on Windows Forms - for example, in WPF applications, or in applications under COM platform. It is even possible to use it in console applications, in which case the console interaction of your application will be combined with graphical UI pop-ups from the Windows Forms Interaction Provider.

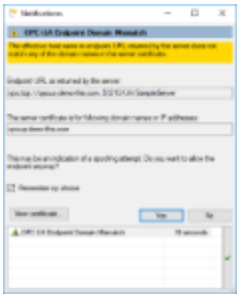
Examples

Accept HTTPS certificate:

Accept instance certificate:



Allow endpoint domain:



Configuration

You can configure parameters of the Windows Forms Interaction by finding UAConsoleInteractionPluginParameters in the shared plug-ins configurations, and changing properties of the object obtained.

6.2 Connectivity Model

The connectivity model is highly generalized, abstract model that allows communication with variuos data sources, without having to know upfront about their implementation details.

If you know ODBC, OLE DB or such technologies for database access, then you can think of Connectivity Model as a similar technology, but for live data, such as in industrial automation or related sectors.

OPC Data Client currently does not make the programmatic access (API) to the Connectivity Model public. The Connectivity Model is, however, on the basis of **Live Binding Model (Section 5.3)**, **Connectivity Explorer (Section 11.2)** tool, and **Excel Option (Section 12.1)** (RTD Server). All these OPC Data Client parts share the features provided by the Connectivity Model, and take advantage from the fact that the model hides the differences between the supported connectivities, such as OPC Classic or OPC Unified Architecture.

6.2.1 Connectivities

The *connectivity* is a component that provides access to a data source of some kind. It can (and usually does) support reading, writing, subscriptions, and browsing.

Three connectivity components are currently provided with OPC Data Client: The DAConnectivity (for OPC Data Access and OPC XML-DA), the UAConnectivity (for OPC Unified Architecture), and the CompositeConnectivity (can be configured to contain and combine any number of other different connectivity components).

6.2.2 Points

Each connectivity must somehow identify the pieces of data it works with. We call them *points*. A connectivity can work with one or more types of points. The composite connectivity merges together the point types of multiple constituent connectivities.

For OPC Data Access (with the DAConnectivity component), following types of points exist:

- DAItemPoint: A point for OPC-DA items.
- DAPropertyPoint: A point for OPC-DA properties (on OPC items).

For OPC Unified Architecture (with the UAConnectivity component), the following point type exists:

- UAAttributePoint: A point for OPC-UA data (node attributes).

> Note: Not all point types have the same capabilities. For example, the most common usage of **Live Binding Model (Section 5.3)**, **Connectivity Explorer (Section 11.2)** tool or the **Excel Option (Section 12.1)** (RTD Server) is with subscriptions (to changing value). Because OPC-DA properties (as opposed to OPC-DA items) do not support subscriptions and we do not emulate them, the subscription operation on OPC-DA property points behaves as a single "get" of the property value.

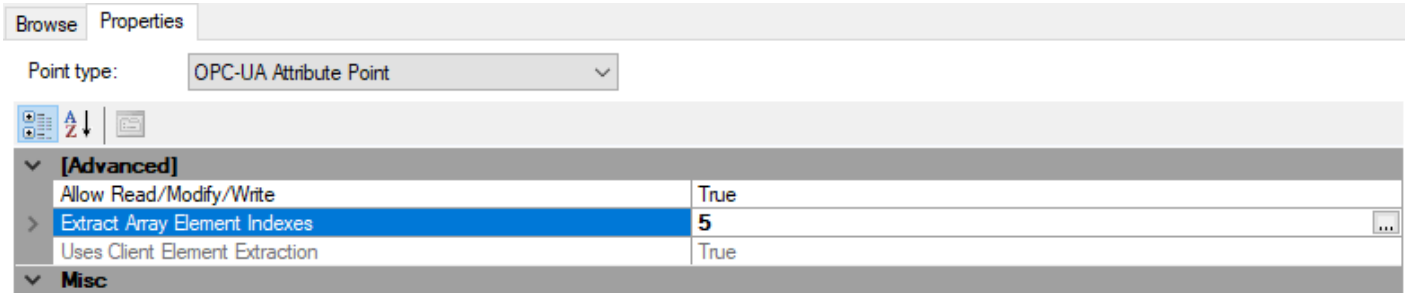
6.2.2.1 Element Extraction

On some points (currently, on all point available: OPC-DA Item Point, OPC-DA Property Point, and OPC-UA Attribute Point), it is possible to use client-side array element extraction.

For array values, this allows you to work with just one element at the given position. With each point, you can specify an index or a list of indices in the ExtractArrayElementIndexes property, and for reads and subscriptions, OPC Data Client will extract the specified element.

In This Topic

Here is how the client-side element extraction is configured in the property grid (the **Properties** tab of the **Point Editor (Section 6.2.3)**).



In the "Extract Array Element Indices" row, you can either manually enter an index or a sequence of indices separated by commas, or you can press the "..." button on the right side to invoke a Collection Editor for the indices.

The client-side array element extraction is mainly useful with OPC Data Access "Classic" servers, because OPC-DA has no corresponding feature in the specification. In OPC Unified Architecture, on the other hand, it is possible to specify Index Range List with each each read, subscribe, or write. If supported, using the Index Range List in OPC UA is recommended over the client-side array element extraction. But not all OPC UA servers support this feature, and in such case you may want to extract the array elements on the client side.

The number of specified indices must match the rank (number of dimensions) of the array, and you can only extract one element (not a range of elements), otherwise an error occurrs.

Reads and Subscriptions

For reads and subscriptions, OPC Data Client will simply read or subscribe to the whole array, and extract the specified element.

Writes

For writes, OPC Data Client will first read the whole array, then modify the element, and then write the whole array back. This read/modify/write algorithm is potentially dangerous if the values are changing in the server or from other clients, and can be disabled using the AllowReadModifyWrite property (defaults to true). When this is disabled and client-side element extraction is specified, the point becomes read-only.

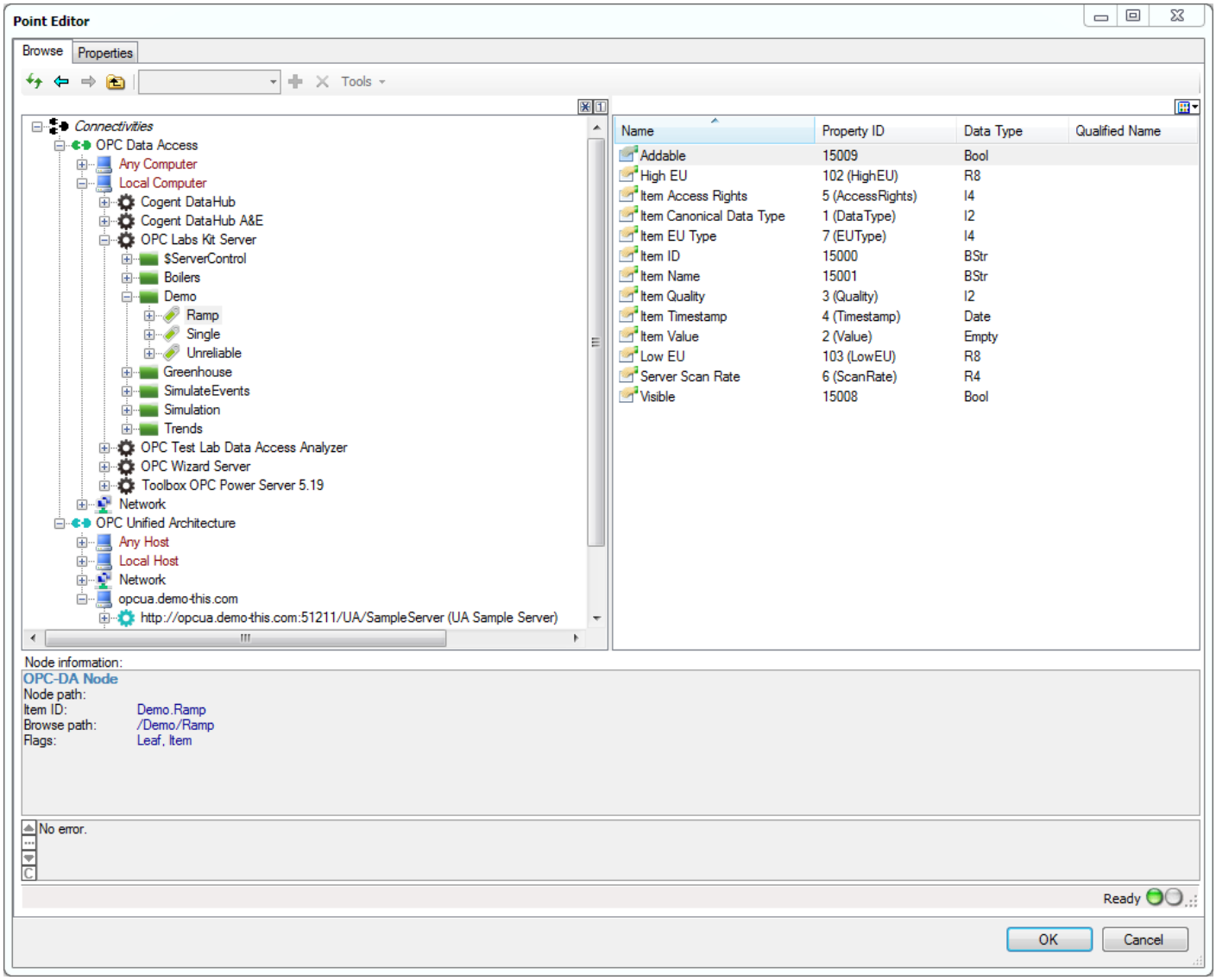
> ⚠ Currently, if you use the element extraction multiple times on the same "underlying" point, OPC Data Client provides no optimization. This means that even if you configure e.g. a point with several different indices, it will be read or subscribed to multiple times. This effect will be mostly noticeable with OPC UA Connectivity, because it currently does not merge together equivalent or similar subscriptions. For writes, each such point will make its own write operation, possibly overwriting the other result(s) if writes are perfomed at once.

6.2.3 Point Editor

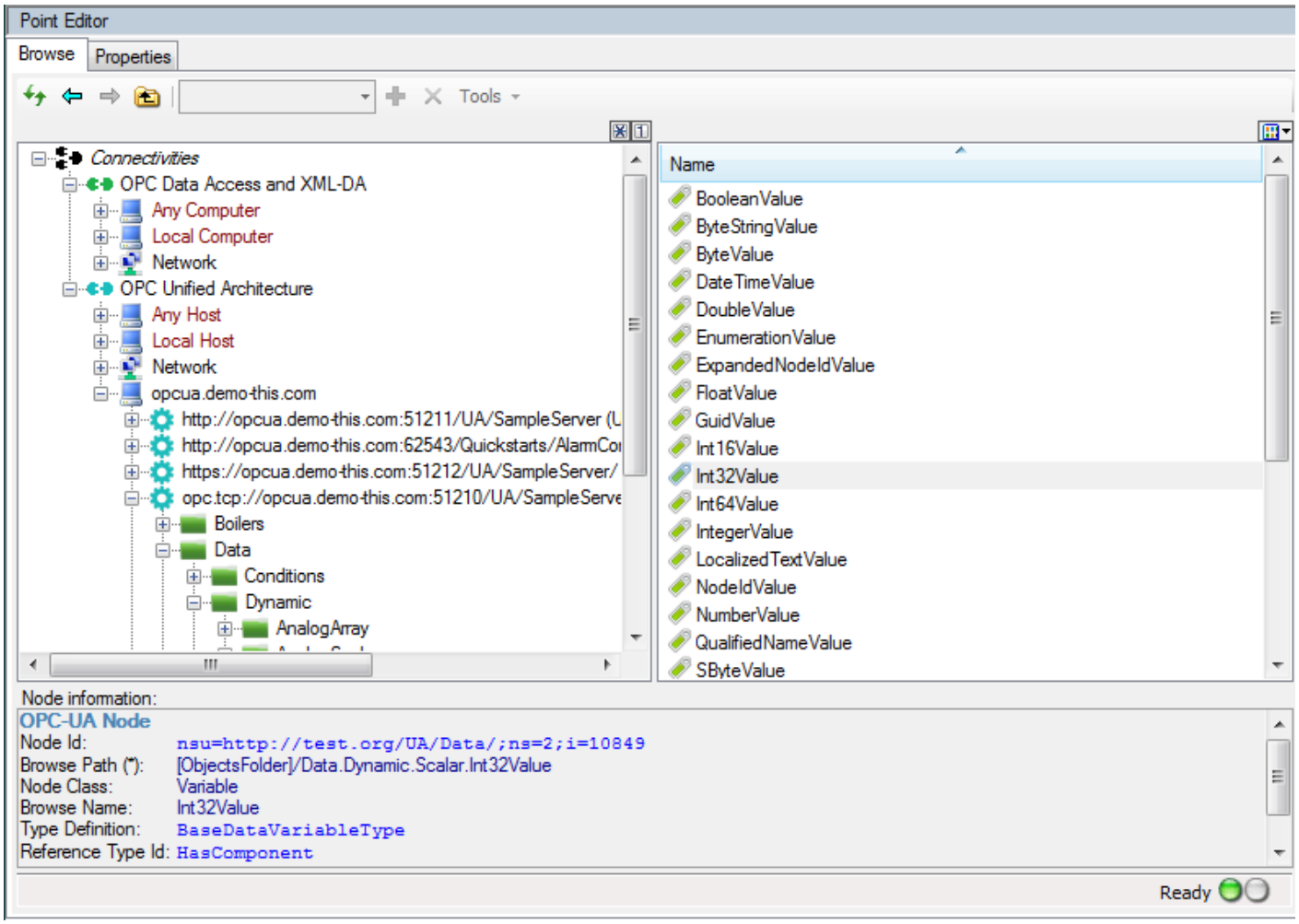
When you are working with the Connectivity Model (such as with the Live Binding in Visual Studio designer, or the Connectivity Explorer tool), you often need to specify the point you'd like to bind to or display.

For this purpose, you will use the **Point Editor**. The Point Editor can be a modal dialog (e.g. in Live Binding), or a subwindow in a larger set (as e.g. in Connectivity Explorer tool), but its main parts are always the same; what differs is the "framing".

In Live Binding, a **Point Editor** dialog is displayed. An example of it is on the following picture:



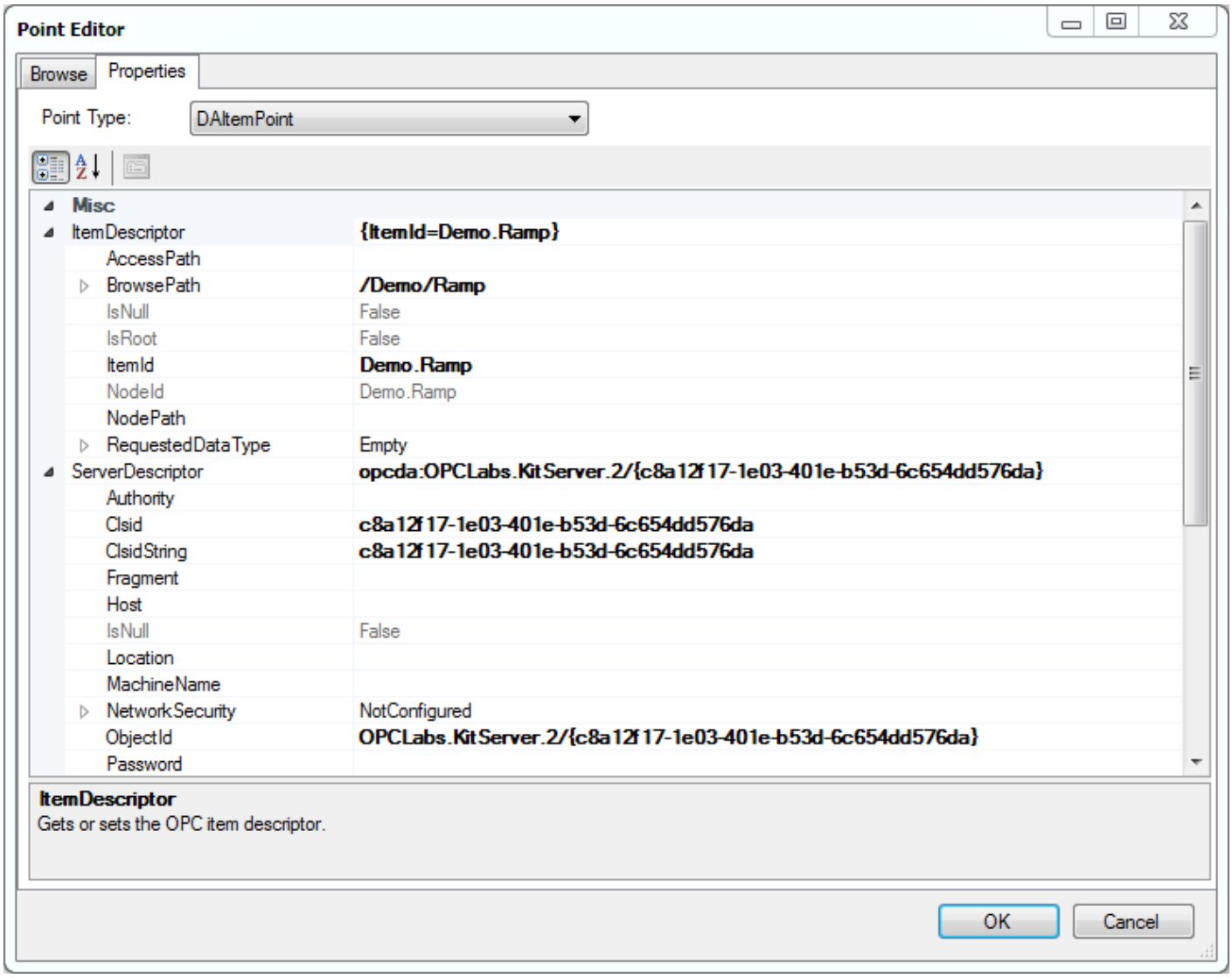
In Connectivity Explorer tool, the main window contains a **Point Editor** subwindow. An example of it is on the following picture:



The actual contents of the dialog or window will of course differ, depending on your configuration. This particular example is with a composite connectivity configured with both OPC Data Access and OPC Unified Architecture, showing multiple binding possibilities in the same dialog.

The point editor has two tabs, labeled **Browse** and **Properties**. Initially, the **Browse** tab is selected, and you are given an opportunity to browse the network, the OPC servers present, and the nodes in them. You can simply select the desired node and press OK (in Live Binding) or double-click on it (in Connectivity Explorer tool).

Sometimes the browsing is not possible, or you may need to view and possibly the details of the selected point. In such cases, you can switch to the **Properties** tab of the dialog:



In the **Properties** tab, the data that describe the point are displayed in a detailed property grid. In addition, in the upper part of the tab, you can select any type of point supported by the connectivity in use, and then fill in its properties as needed.

6.2.3.1 Browse Tab

The **Browse** tab is selected initially (see picture in the **Point Editor (Section 6.2.3)**), and you are given an opportunity to browse the network, the OPC servers present, and the nodes in them. You can simply select the desired node from the tree or list.

If yo do not see the entities you are looking for, you can enter some of them manually. For example, you can enter Computer or Host, and OPC Server or OPC-UA Endpoint.

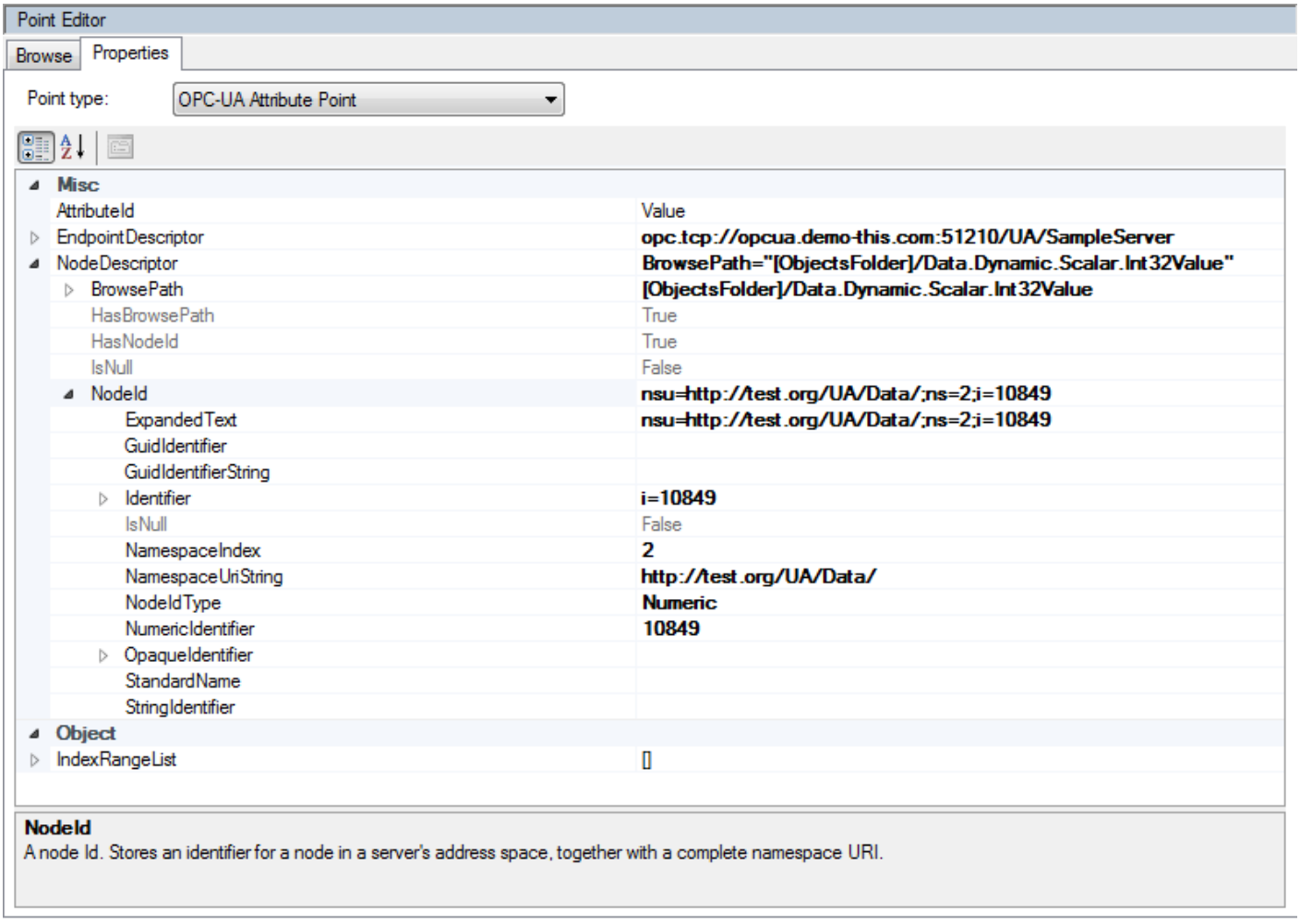
You can also add "Index Range List (OPC-UA)" nodes in the Point Editor. This makes it easier to specify points with index ranges. You do not have to switch into the **Properties** tab, click "..." in the "Index Range List" row, and then "Add" each index range and enter its index values. Instead, simply "Add" the "Index Range List (OPC-UA)" under any OPC UA node,

and then type in the required index range(s), e.g. "5:10".

The **Browse** tab also has a **Legend** pane, allowing the user to view the list of node icons used in the control, and their meaning (descriptions). The **Legend** pane can be turned off and on from the **Layout** menu of the browse control.

6.2.3.2 Properties Tab

In the **Properties** tab, the data that describe the point are displayed in a detailed property grid. In addition, in the upper part of the tab, you can select any type of point supported by the connectivity in use, and then fill in its properties as needed.



6.2.4 Parameters

While the point determines basically the location of the data, *parameters* determine how the data will be accessed. Some operations do not need parameters (in such cases they still exist, but are empty), but most do. For example, a subscription

typically has some kind of update rate as its parameter.

The parameters may depend on the type of point, and the operation involved. A typical connectivity that supports reading, writing and subscription will therefore have up to three types of parameters for each type of point: Read parameters, write parameters, and subscription parameters.

The parameters are not part of the point itself, and are therefore not selected with it in the Point Editor. They are part of the binding (in **Live Binding Model (Section 5.3)**) or data row definitions (in **Connectivity Explorer (Section 11.2)** tool).

6.2.5 Arguments and Results

The *arguments* are the actual data the operation on a connectivity transfers, i.e. the "what". Typically and most importantly, arguments contain the data value being read or written or obtained through subscription.

As with the parameters, they usually depend on the type of point, and the operation involved.

Arguments are represented by an object that has properties, divided into several categories. The categories are as follows:

- **Input**: The data that is taken from the target and provided to the operation (for example, a value to be written).
- **Output**: The data that is obtained by the operation and stored into the target (for example, the value read).
- **Input/Output**: A combination of an input and an output argument.
- **Result**: The result of the operation (for example, the error ID returned).

There is an important distinction between the *outputs*, and the *results*. Each operation can be either successful, or not. When the operation completes, the result arguments are always available. The outputs, however, are only available when the operation succeeded.

6.3 Services

OPC Data Client services are functionality sets that are complementary to the main task of the component. They provide features that are not directly derived from the underlying communication methods of the component, but rather have to do with its integration to the environment, instrumentation (such as configuration), licensing etc.

In This Topic

> ⚠ The term *service* is quite overloaded. It can mean different things in different context. For example, there are *OPC UA services*, which are basically method calls on the OPC UA protocol level. And there are *Windows services*, processes that run in background on the Windows operating system. There are other uses for the term *service* as well. Try not to confuse the different concepts. This article is about *OPC Data Client services*, which are neither OPC UA services nor Windows services nor anything else. In this document, we strive to consistently qualify the term *service* with additional words, wherever needed to avoid ambiguity.

Obtaining Service Interfaces

You can obtain services from a *service provider* object, which is any type that derives, directly or indirectly, from the IServiceProvider interface. The major OPC Data Client components, such as EasyDAClient, EasyAEClient or EasyUAClient, and some other components, are all service providers as well.

The OPC Data Client services are identified by service type, which a .NET Type object, usually an interface. In order to

obtain a service from a service provider, call its GetService method, passing it the type of service of you want to obtain. In most cases, the returned object is of the type you have specified (unless documented otherwise). You would therefore normally cast the received Object to the expected service type. In order to make the coding easier (in .NET), you can use the GetService extension method, which saves you from using the typeof operator (in C#, or its equivalent in other languages) and from casting the result.

When developing on the COM platform, you would have troubles obtaining the .NET Type object to pass into the GetService method. In order to overcome these difficulties, we have provided a GetServiceByName method on each COM interface that represents a service provider. The argument to this method is a string instead. The string represent a fully qualified name of the service type you want to obtain. For example, in order to obtain the _EasyUAClientApplication service from the _EasyUAClient, call is GetServiceByName method, passing it a string **"OpcLabs.EasyOpc.UA.Application.ComTypes._EasyUAClientApplication"**.

The GetService (or GetServiceByName) method returns a null reference when the specified interface type is not available. The set of services may depend on the various factors, such as the client object used, its configuration, software version etc. Unless you are sure that the service is available, your code should be able to handle the null result well, e.g. by gracefully degrading its functionality.

Service Catalog

Different objects provide different types of services. The following paragraphs provide an overview of the types of services available from each major OPC Data Client component.

EasyDAClient Services

The EasyDAClient object can provide following services:

Service Type (.NET)	Service Type (COM)	Title/Purpose	Description
IDisposable	IDisposable	Disposable Object	Provides a mechanism for releasing unmanaged resources. Useful when you deal with a client interface already downcasted to IServiceProvider.
ILicenseInfoProvider	×	License Info Provider	Provides information about the licensed object. Example: **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**.
IEasyDAClient	_EasyDAClient	OPC-DA Client	The client itself. Useful when you deal with a client interface already downcasted to IServiceProvider.
IEasyDAClientSettings	×	OPC-DA Client Settings	Provides access to settings for IEasyDAClient objects.

EasyAEClient Services

The EasyAEClient object can provide following services:

Service Type (.NET)	Service Type (COM)	Title/Purpose	Description
IDisposable	IDisposable	Disposable Object	Provides a mechanism for releasing unmanaged resources. Useful when you deal with a client interface already downcasted to IServiceProvider.
ILicenseInfoProvider	×	License Info Provider	Provides information about the licensed object. Example: **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**.
IEasyAEClient	_EasyAEClient	OPC A&E Client	The client itself. Useful when you deal with a client interface already downcasted to IServiceProvider.
IEasyAEClientSettings	×	OPC A&E Client Settings	Provides access to settings for IEasyAEClient objects.

EasyUAClient Services

The EasyUAClient object can provide following services:

Service Type (.NET)	Service Type (COM)	Title/Purpose	Description
IDisposable	IDisposable	Disposable Object	Provides a mechanism for releasing unmanaged resources. Useful when you deal with a client interface already downcasted to IServiceProvider.
ILicenseInfoProvider	×	License Info Provider	Provides information about the licensed object. Example: **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**.
IEasyUAClientApplication	_EasyUAClientApplication	OPC UA Client Application	Allows management of the **OPC UA Client Application Service (Section 6.3.1)**.
IEasyUAClient	_EasyUAClient	OPC UA Client	The client itself. Useful when you deal with a client interface already downcasted

			to IServiceProvider.
IEasyUAClientSettings	×	OPC UA Client Settings	Provides access to settings for IEasyUAClient objects.
IEasyUAClientComplexData	_EasyUAClientComplexData	OPC UA Complex Data	Gives advanced access to **OPC UA Complex Data Extension (Section 7.2.1)**. Only available if the corresponding plug-in is enabled. For more information, see also **Navigation in the OPC UA data model (Section 7.2.1.4.3)** and **Adding or removing data type dictionaries to or from the OPC UA data type system (Section 7.2.1.4.4)** in **Advanced OPC UA Complex Data Tasks (Section 7.2.1.4)**.

EasyUAAlarmsAndConditionsClient Services

The EasyUAAlarmsAndConditionsClient object can provide following services:

Service Type (.NET)	Service Type (COM)	Title/Purpose	Description
IDisposable	IDisposable	Disposable Object	Provides a mechanism for releasing unmanaged resources. Useful when you deal with a client interface already downcasted to IServiceProvider.
IEasyUAAlarmsAndConditionsClient	_EasyUAAlarmsAndConditionsClient	OPC UA Alarms and Conditions Client	The specialized client itself. Useful when you deal with a client interface already downcasted to IServiceProvider.
ILicenseInfoProvider	×	License Info Provider	Provides information about the licensed object. Example: **Examples - Licensing -**

			Obtain serial number (Section 13.2.1.1).

EasyUACertificateManagementClient Services

The EasyUACertificateManagementClient object can provide following services:

Service Type (.NET)	Service Type (COM)	Title/Purpose	Description
IDisposable	IDisposable	Disposable Object	Provides a mechanism for releasing unmanaged resources. Useful when you deal with a client interface already downcasted to IServiceProvider.
IEasyUACertificateManagementClient	_EasyUACertificateManagementClient	OPC UA Certificate Management Client	The specialized client itself. Useful when you deal with a client interface already downcasted to IServiceProvider.
ILicenseInfoProvider	×	License Info Provider	Provides information about the licensed object. Example: **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**.

EasyUAGlobalDiscoveryClient Services

The EasyUAGlobalDiscoveryClient object can provide following services:

Service Type (.NET)	Service Type (COM)	Title/Purpose	Description
IDisposable	IDisposable	Disposable Object	Provides a mechanism for releasing unmanaged resources. Useful when you deal with

			a client interface already downcasted to IServiceProvider.
IEasyUAGlobalDiscoveryClient	_EasyUAGlobalDiscoveryClient	OPC UA Global Discovery Client	The specialized client itself. Useful when you deal with a client interface already downcasted to IServiceProvider.
ILicenseInfoProvider	×	License Info Provider	Provides information about the licensed object. Example: **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**.

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

6.3.1 OPC UA Client Application Service

This OPC Data Client service allows management of OPC UA client application. The OPC UA Client Application Service has high-level properties and methods for tasks related to maintaining application registrations in the GDS servers, obtaining instance certificate from GDS, refreshing application trust lists from GDS, and similar tasks.

If you can, prefer use of the OPC UA Client Application Service over implementing the same functionality yourself, e.g. with the help of **OPC UA Global Discovery Client (Section 6.4.1)** and/or **OPC UA Certificate Management Client (Section 6.4.2)**. The OPC UA Client Application Service already has all the necessary logic needed to perform the typical client-side application tasks properly and easily.

In This Topic

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

Specifically, methods in this service that work with the Global Discovery Server (GDS) are not licensed for use in lower editions of OPC Data Client.

Exporting

In order to export the registration data of the OPC UA application into an XML file, call the ExportRegisteredApplication Method. This method exports the security settings in the format used by the GDS client from OPC Foundation (Windows desktop sample). It uses the **RegisteredApplication** schema (which is not part of the OPC UA standard).

In order to export the security settings of the OPC UA application into an XML file, call the ExportSecuredApplication Method. This method exports the security settings in the format described in OPC UA Specification Part 6, Annex "Security settings management". The file conforms to the **SecuredApplication** schema.

Managing GDS Registrations

The OPC UA Client Application Service is capable of maintaining client application registrations with one or more GDS servers simultaneously. The methods described further below allow you to control these registrations.

The service keeps track of the registrations (and unregistrations) performed. You can use the ApplicationIdDictionary Property to figure out the application IDs assigned to the current client application by the OPC UA Global Discovery Servers.

> Registration of OPC UA clients in the GDS is not "as important" as registration of OPC UA servers, because (except for so-called Reverse Connect) it is the clients that need to discover the servers, and not vice versa. The application registration is, however, a prerequisite for certificate management (also typically provided by the GDS), and for this reason, you may find yourself dealing with registration of OPC UA client application with the GDS anyway.

Registering to GDS

In order to register the current client application with the OPC UA Global Discovery Server (GDS), call the RegisterToGds Method. This method creates an application registration in the GDS, assigning it a new application ID. Existing registrations with the same application URI are removed first.

The difference between the RegisterToGds and UpdateGdsRegistration methods is that the RegisterToGds Method always obtains a new application ID, whereas the UpdateGdsRegistration Method attempts to reuse the existing application ID, if known. Neither of these methods requires the GDS registration be in any particular state for it to succeed, and both methods assure that after a successful execution, there is one and only one registration in the given GDS for this application.

The identity of the application itself is carried by its application URI, as given by the ApplicationUriString returned by the GetApplicationElement Method. Methods on this interface that manipulate or inspect the GDS registration use the application URI to identify the registration records belonging to this application.

For ane example to this method, see: **Examples - OPC UA Application - Register to GDS (Section 13.2.7.7)**.

Updating GDS Registration

In order to update the registration of the current client application within the OPC UA Global Discovery Server (GDS), call the UpdateGdsRegistration Method. This method updates an application registration in the GDS, keeping its application ID. A new registration is created if the application is not yet registered in the GDS. Preexisting registrations with the same application URI are removed.

The difference between the RegisterToGds and UpdateGdsRegistration methods is that the RegisterToGds Method always obtains a new application ID, whereas the UpdateGdsRegistration Method attempts to reuse the existing application ID, if known. Neither of these methods requires the GDS registration be in any particular state for it to succeed, and both methods assure that after a successful execution, there is one and only one registration in the given GDS for this application.

The identity of the application itself is carried by its application URI, as given by the ApplicationUriString returned by the GetApplicationElement Method. Methods on this interface that manipulate or inspect the GDS registration use the application URI to identify the registration records belonging to this application.

C#

```
// Shows how to update an application registration in the GDS, keeping its application ID
if possible.
```

```
using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class UpdateGdsRegistration
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Update an application registration in the GDS, keeping its application ID if
possible.
            UANodeId applicationId;
            try
            {
                applicationId =
clientApplication.UpdateGdsRegistration(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Application ID: {0}", applicationId);
        }
    }
}
```

Object Pascal

```
// Shows how to update an application registration in the GDS, keeping its application ID
if possible.

class procedure UpdateGdsRegistration.Main;
var
  ApplicationId: _UANodeId;
  Client: _EasyUAClient;
```

```
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Update an application registration in the GDS, keeping its application ID if possible.
  try
    ApplicationId := ClientApplication.UpdateGdsRegistration(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Application ID: ', ApplicationId.ToString);
end;
```

PHP

```
// Shows how to update an application registration in the GDS, keeping its application ID
if possible.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

// Update an application registration in the GDS, keeping its application ID if possible.
try
{
```

```
    $ApplicationId = $ClientApplication->UpdateGdsRegistration($GdsEndpointDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("Application ID: %s\n", $ApplicationId);
```

VB.NET

```
' Shows how to update an application registration in the GDS, keeping its application ID if
possible.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class UpdateGdsRegistration
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Update an application registration in the GDS, keeping its application ID if
possible.
            Dim applicationId As UANodeId
            Try
                applicationId =
clientApplication.UpdateGdsRegistration(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Application ID: {0}", applicationId)
        End Sub
    End Class
End Namespace
```

Unregistering from GDS

In order to remove the registration of the current client application from the OPC UA Global DIscovery Server (GDS), call the UnregisterFromGds Method. All existing registrations for the application URI are removed.

This method does not return an error or throw an exception if the application is not currently registered in the GDS.

C#

```
// Shows how to delete an application registration from the GDS.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class UnregisterFromGds
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Delete an application registration from the GDS.
            try
            {
                clientApplication.UnregisterFromGds(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
        }
    }
}
```

Object Pascal

```
// Shows how to delete an application registration from the GDS.
```

```
class procedure UnregisterFromGds.Main;
var
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Delete an application registration from the GDS.
  try
    ClientApplication.UnregisterFromGds(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;
end;
```

PHP

```
// Shows how to delete an application registration from the GDS.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

  // Delete an application registration from the GDS.
try
{
    $ClientApplication->UnregisterFromGds($GdsEndpointDescriptor);
```

```
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}
```

VB.NET

```
' Shows how to delete an application registration from the GDS.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class UnregisterFromGds
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Delete an application registration from the GDS.
            Try
                clientApplication.UnregisterFromGds(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

Finding GDS Registrations

In order to find all registrations for the current client application (using its application URI) in the OPC UA Global Discovery Server (GDS), call the FindGdsRegistrations Method. The method returns a dictionary of UAApplicationElement objects with application registration information, keyed by a UANodeId which is the application ID identifying the registration in the GDS.

The identity of the application itself is carried by its application URI, as given by the ApplicationUriString returned by the GetApplicationElement Method. Methods on this interface that manipulate or inspect the GDS registration use the application URI to identify the registration records belonging to this application.

For ane example to this method, see: **Examples - OPC UA Application - Find all our registrations in GDS (Section 13.2.7.1)**.

Getting Application Element

"Application element" is a data structure that contains registration information about an OPC UA application, i.e. how the application is configured to register itself in OPC UA ecosystem. In order to get the registration information for the current client application, call the GetApplicationElement Method.

The data contained the application element is implementation-dependent. For the OPC UA client application service on the EasyUAClient Class implemented with the OPC Foundation's UA SDK, it is determined by the settings in the UAClientApplicationParameters Class. In order to modify these parameters, access the ApplicationParameters Property of EngineParameters Property of static SharedParameters Property.

For ane example to this method, see: **Examples - OPC UA Application - Get registration information (Section 13.2.7.3)**.

Managing Certificates

Some methods in this group work with the OPC UA Global Discovery Server (GDS) in the role of Certificate Manager (CM). Use of the certificate manager can significantly simplify the management of OPC UA security. Only use these methods if your OPC UA setup includes such Global Discovery Server.

Obtaining a New Certificate

In order to begin an asynchronous operation that obtains a enw application instance certificate from the certificate manager and stores it for subsequent usage, call the BeginObtainNewCertificate Method.

The operation is asynchronous, because it involves multiple steps, and waiting for a finalization of the request by the GDS (see the IEasyUACertificateManagementClient.FinishRequest method). End of the operation is handled by the IEasyUAClientApplication.EndObtainNewCertificate method. The operation can be cancelled using the IEasyUAClientApplication.CancelObtainNewCertificate method.

For a synchronous alternative, see the ObtainNewCertificate Extension Method, with its overloads.

For a task-based asynchronous programming pattern alternative, see the ObtainNewCertificateAsync Extension Method, with its overloads.

C#

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.Security.Cryptography.PkiCertificates;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Application.Extensions;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    partial class ObtainNewCertificate
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
```

```
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.
            PkiCertificate certificate;
            try
            {
                certificate =
clientApplication.ObtainNewCertificate(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Certificate: {0}", certificate);
        }
    }
}
```

Object Pascal

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
// and store it for subsequent usage.

class procedure ObtainNewCertificate.Main;
var
  ApplicationElement: _UAApplicationElement;
  Certificate: _X509Certificate;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  Parameters: _UAObtainNewCertificateParameters;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;
```

```
  // Display which application we are about to work with.
  ApplicationElement := ClientApplication.GetApplicationElement;
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Obtain a new application certificate from the certificate manager (GDS), and store it
for subsequent usage.
  Parameters := CoUAObtainNewCertificateParameters.Create;

  try
    Certificate := ClientApplication.ObtainNewCertificate(GdsEndpointDescriptor,
Parameters);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Certificate: ', (Certificate as _PKICertificate).ToString);
end;
```

PHP

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
// and store it for subsequent usage.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
$ApplicationElement = $ClientApplication->GetApplicationElement;
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

// Obtain a new application certificate from the certificate manager (GDS), and store it
for subsequent usage.
$Parameters = new COM("OpcLabs.EasyOpc.UA.Application.UAObtainNewCertificateParameters");

try
{
    $Certificate = $ClientApplication->ObtainNewCertificate($GdsEndpointDescriptor,
$Parameters);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
```

```
    exit();
}

// Display results
printf("Certificate: %s\n", $Certificate);
```

VB.NET

```
' Shows how to obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.Security.Cryptography.PkiCertificates
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Application.Extensions
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Partial Friend Class ObtainNewCertificate
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.
            Dim certificate As PkiCertificate
            Try
                certificate = clientApplication.ObtainNewCertificate(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Certificate: {0}", certificate)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage.

Option Explicit
```

```
' Define which GDS we will work with.
Dim GdsEndpointDescriptor: Set GdsEndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
GdsEndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName = "appadmin"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password = "demo"

' Obtain the client application service.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ClientApplication: Set ClientApplication =
Client.GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA")

' Display which application we are about to work with.
Dim ApplicationElement: Set ApplicationElement = ClientApplication.GetApplicationElement
WScript.Echo "Application URI string: " &
ClientApplication.GetApplicationElement.ApplicationUriString

Rem Obtain a new application certificate from the certificate manager (GDS), and store it
for subsequent usage.
Dim Parameters: Set Parameters =
CreateObject("OpcLabs.EasyOpc.UA.Application.UAObtainNewCertificateParameters")
On Error Resume Next
Dim Certificate: Set Certificate =
ClientApplication.ObtainNewCertificate(GdsEndpointDescriptor, Parameters)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Certificate: " & Certificate
```

This method can also be provided a callback, in form of the IProgress Interface, through which it reports its progress.

C#

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage,
// with progress reporting.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.Security.Cryptography.PkiCertificates;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Application.Extensions;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    partial class ObtainNewCertificate
    {
        public static void Progress()
```

```
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.
            PkiCertificate certificate;
            try
            {
                certificate = clientApplication.ObtainNewCertificate(gdsEndpointDescriptor,
                    new Progress<string>(s => Console.WriteLine("Progress: {0}", s)));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Certificate: {0}", certificate);
        }
    }
}
```

VB.NET

```
' Shows how to obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage,
' with progress reporting.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.Security.Cryptography.PkiCertificates
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Application.Extensions
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Partial Friend Class ObtainNewCertificate
        Public Shared Sub Progress()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
```

```
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.
            Dim certificate As PkiCertificate
            Try
                certificate = clientApplication.ObtainNewCertificate(gdsEndpointDescriptor,
                    New Progress(Of String)(Sub(s) Console.WriteLine("Progress: {0}", s)))
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Certificate: {0}", certificate)
        End Sub
    End Class
End Namespace
```

Refreshing Trust Lists

In order to retrieve the current trust lists for the application from the certificate manager, and refresh own certificate stores accordingly, call the RefreshTrustLists Method.

The operation first updates the application's registration in the GDS, before retrieving the trust lists from the certificate manager and adding them to the certificate stores

C#

```
// Shows how to refresh own certificate stores using current trust lists for the
application from the certificate manager.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class RefreshTrustLists
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
```

```
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Refresh own certificate stores using current trust lists for the application
from the certificate manager.
            UATrustListMasks refreshedTrustLists;
            try
            {
                refreshedTrustLists =
clientApplication.RefreshTrustLists(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Refreshed trust lists: {0}", refreshedTrustLists);
        }
    }
}
```

Object Pascal

```
// Shows how to refresh own certificate stores using current trust lists
// for the application from the certificate manager.

class procedure RefreshTrustLists.Main;
var
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  RefreshedTrustLists: UATrustListMasks;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
```

```
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Refresh own certificate stores using current trust lists for the application from the
certificate manager.
  try
    RefreshedTrustLists := clientApplication.RefreshTrustLists(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('Refreshed trust lists: ', RefreshedTrustLists);
end;
```

PHP

```
// Shows how to refresh own certificate stores using current trust lists
// for the application from the certificate manager.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

// Refresh own certificate stores using current trust lists for the application from the
certificate manager.
try
{
    $RefreshedTrustLists = $ClientApplication->RefreshTrustLists($GdsEndpointDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("Refreshed trust lists: %s\n", $RefreshedTrustLists);
```

VB.NET

```
' Shows how to refresh own certificate stores using current trust lists for the application
from the certificate manager.
```

```
Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class RefreshTrustLists
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Refresh own certificate stores using current trust lists for the application
from the certificate manager.
            Dim refreshedTrustLists As UATrustListMasks
            Try
                refreshedTrustLists =
clientApplication.RefreshTrustLists(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Refreshed trust lists: {0}", refreshedTrustLists)
        End Sub
    End Class
End Namespace
```

Getting Instance Certificate

In order to get the instance certificate the application is currently configured to use, call the GetInstanceCertificate Method.

The method returns the instance certificate, if found. It returns null if the instance certificate cannot be found.

Getting Certificate Subject Name

In order to get the the subject distinguished name the application is configured to use for its certificates, call the GetCertificateSubjectName Method.

The value returned is implementation-dependent. For the OPC UA client application service on the EasyUAClient Class implemented with the OPC Foundation's UA SDK, it is determined by the settings in the UAClientApplicationParameters Class, as described here: **Providing OPC UA Client Instance Certificate (Section 4.13.1.1)**,

For ane example to this method, see: **Examples - OPC UA Application - Get certificate subject name (Section 13.2.7.2)** .

6.4 Specialized Client Objects

A specialized client object builds on top of a general-purpose client object (such as EasyUAClient) and provides functionality that is relevant for specific problem domain. It contains, in a sense, pre-packaged pieces of reusable code that the developer would typically need to write, whenever he needs to develop a solution for that problem domain.

In OPC UA, the problem domain addressed by the specialized client object is often a selected part of the OPC UA specification. This is due to the layered architecture of OPC UA, where additional informational models can be used to enhance the functionality and address new problem domains, while the basic set of services (covered by the general-purpose client object) remains static and relatively small.

In This Topic

Specialized Client Derivation

A specialized client object can be created from an existing general-purpose client (such as when you have the IEasyUAClient Interface) using so-called *derivation*. For derivation, there is a method (or an extension method) on the general-purpose client, that returns the specialized client needed. For example, you can call the AsGlobalDiscoveryClient (Extension) Method on the IEasyUAClient Interface, and you will receive back a specialized client object for OPC UA global discovery.

A specialized client object created using derivation performs its operations using the general-purpose client object that it was derived from, and therefore also uses all its settings, always shares the same connections, etc. You can then think of the specialized client simply as a functionality extension of the general-purpose client object.

It is recommended that you use the specialized client derivation whenever you have a suitable general-purpose client object available.

Standalone Specialized Clients

A specialized client object can also be created standalone, without having a general-purpose client object first. This is done simply by instantiating the specialized client object, e.g. EasyUAGlobalDiscoveryClient. When created in this way, the specialized client object will use an internal general-purpose client to perform its operations. The parameters of such general-purpose client object, if they are of interest to you, are available in the ClientSelector Property of the specialized client. Note that if you want to change any of these parameters, you must do so before you perform the first operation on the specialized client object. Also note that the internal general-purpose client object may be used for multiple purposes by OPC Data Client.

The example below uses a standalone specialized client object to perform OPC UA global discovery operation.

C#

```
// Shows how to unregister all clients from a GDS.

using System;
using System.Collections.Generic;
using System.Linq;
using OpcLabs.BaseLib.Collections.Generic.Extensions;
```

```
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
{
    class UnregisterApplication
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Instantiate the global discovery client object
            var globalDiscoveryClient = new EasyUAGlobalDiscoveryClient();

            // Find application IDs of all client applications registered in the GDS.
            var clientApplicationIds = new HashSet<UANodeId>();
            try
            {
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor: gdsEndpointDescriptor,
                    startingRecordId: 0,
                    maximumRecordsToReturn: 0,
                    applicationName: "",
                    applicationUriString: "",
                    applicationTypes: UAApplicationTypes.Client,
                    productUriString: "",
                    serverCapabilities: new string[0],
                    lastCounterResetTime: out _,
                    nextRecordId: out _,
                    applications: out UAApplicationDescription[]
applicationDescriptionArray);

                foreach (UAApplicationDescription applicationDescription in
applicationDescriptionArray)
                {
                    var applicationRecordArray =
globalDiscoveryClient.FindApplications(
                        gdsEndpointDescriptor,
                        applicationDescription.ApplicationUriString);

clientApplicationIds.AddRange(applicationRecordArray.Select(description =>
description.ApplicationId));
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
```

```
            }

            // Unregister all client applications found.
            foreach (UANodeId applicationId in clientApplicationIds)
            {
                Console.WriteLine();
                Console.WriteLine("Application ID: {0}", applicationId);

                try
                {
                    globalDiscoveryClient.UnregisterApplication(gdsEndpointDescriptor,
applicationId);
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                    continue;
                }
                Console.WriteLine("Unregistered.");
            }
        }
    }
}
```

Object Pascal

```
// Shows how to unregister all clients from a GDS.

class procedure UnregisterApplication.Main;
var
  ApplicationDescription: _UAApplicationDescription;
  ApplicationDescriptionArray: OleVariant;
  ApplicationId: _UANodeId;
  ApplicationName: WideString;
  ApplicationRecord: _UAApplicationRecordDataType;
  ApplicationRecordArray: OleVariant;
  ApplicationUriString: WideString;
  ClientApplicationIds: TList<_UANodeID>;
  GlobalDiscoveryClient: OpcLabs_EasyOpcUA_TLB._EasyUAGlobalDiscoveryClient;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  I, J: integer;
  LastCounterResetTime: TDateTime;
  MaximumRecordsToReturn: Integer;
  NextRecordId: Integer;
  ProductUriString: WideString;
  ServerCapabilities: array of string;
  StartingRecordId: Integer;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Instantiate the global discovery client object
```

```
  GlobalDiscoveryClient := CoEasyUAGlobalDiscoveryClient.Create;

  // Find application IDs of all client applications registered in the GDS.
  ClientApplicationIds := TList<_UANodeID>.Create();
  StartingRecordId := 0;
  MaximumRecordsToReturn := 0;
  ApplicationName := '';
  ApplicationUriString := '';
  ProductUriString := '';
  try
    GlobalDiscoveryClient.QueryApplications(
      GdsEndpointDescriptor,
      StartingRecordId,
      MaximumRecordsToReturn,
      ApplicationName,
      ApplicationUriString,
      UAApplicationTypes_Client,
      ProductUriString,
      ServerCapabilities,
      LastCounterResetTime,
      NextRecordId,
      ApplicationDescriptionArray);

    for I := VarArrayLowBound(ApplicationDescriptionArray,1) to
VarArrayHighBound(ApplicationDescriptionArray,1) do
    begin
      ApplicationDescription := IUnknown(ApplicationDescriptionArray[I]) as
_UAApplicationDescription;
      TVarData(ApplicationRecordArray).VType := varArray or varVariant;
      TVarData(ApplicationRecordArray).VArray :=
PVarArray(GlobalDiscoveryClient.FindApplications(
        GdsEndpointDescriptor, ApplicationDescription.ApplicationUriString));
      for J := VarArrayLowBound(ApplicationRecordArray,1) to
VarArrayHighBound(ApplicationRecordArray,1) do
      begin
        ApplicationRecord := IUnknown(ApplicationRecordArray[J]) as
_UAApplicationRecordDataType;
        ClientApplicationIds.Add(ApplicationRecord.ApplicationId);
      end;
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Unregister all client applications found.
  for ApplicationId in ClientApplicationIds do
  begin
    WriteLn;
    WriteLn('Application ID: ', ApplicationId.ToString);
    try
      GlobalDiscoveryClient.UnregisterApplication(
        GdsEndpointDescriptor, ApplicationId);
    except
      on E: EOleException do
```

```
      begin
        WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
        Continue;
      end;
    end;
    WriteLn('Unregistered.');
  end;
  FreeAndNil(ClientApplicationIds);
end;
```

VB.NET

```
' Shows how to unregister all clients from a GDS.

Imports System.Linq
Imports OpcLabs.BaseLib.Collections.Generic.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
    Friend Class UnregisterApplication
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Instantiate the global discovery client object
            Dim globalDiscoveryClient = New EasyUAGlobalDiscoveryClient()

            ' Find application IDs of all client applications registered in the GDS.
            Dim clientApplicationIds = New HashSet(Of UANodeId)
            Try
                Dim lastCounterResetTime As DateTime
                Dim nextRecordId As Long
                Dim applicationDescriptionArray() As UAApplicationDescription = Nothing
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor:=gdsEndpointDescriptor,
                    startingRecordId:=0,
                    maximumRecordsToReturn:=0,
                    applicationName:="",
                    applicationUriString:="",
                    applicationTypes:=UAApplicationTypes.Client,
                    productUriString:="",
                    serverCapabilities:=New String() {},
                    lastCounterResetTime:=lastCounterResetTime,
                    nextRecordId:=nextRecordId,
                    applications:=applicationDescriptionArray)

                For Each applicationDescription As UAApplicationDescription In
applicationDescriptionArray
```

```
                    Dim applicationRecordArray =
globalDiscoveryClient.FindApplications(
                        gdsEndpointDescriptor,
                        applicationDescription.ApplicationUriString)

clientApplicationIds.AddRange(applicationRecordArray.Select(Function(description)
description.ApplicationId))
                Next applicationDescription
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Unregister all client applications found.
            For Each applicationId As UANodeId In clientApplicationIds
                Console.WriteLine()
                Console.WriteLine("Application ID: {0}", applicationId)

                Try
                    globalDiscoveryClient.UnregisterApplication(gdsEndpointDescriptor,
applicationId)
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Continue For
                End Try
                Console.WriteLine("Unregistered.")
            Next applicationId
        End Sub
    End Class
End Namespace
```

Visual Studio Toolbox

The standalone specialized clients are also available as icons in the Visual Studio Toolbox. This means that in certain types of projects, they can be dragged from the Toolbox to the design surface, and Visual Studio will create code to instantiate and configure the specialized client object. Component properties are then also editable from the Visual Studio (in the Properties tool window).

Reverting to the General-Purpose Client Object

In a similar as you can derive a specialized client object from a general-purpose client, you can go in the opposite direction. If you have a specialized client object at hand, you can use its AsClient method to obtain the client object upon which the specialized client is built. If the specialized client object has been derived from a general-purpose client, this method returns the original general-purpose client object used in derivation. If the specialized client object us standalone, this method returns the internal general-purpose client object used by this specialized client.

The IsDerived Property determines whether a given specialized client object has been derived from a general-purpose client object.

Specialized Clients Catalog

There are currently following specialized client objects available:

- OPC UA Alarms and Conditions Client. See **Modifying Information (OPC UA Alarms & Conditions) (Section 5.1.4.2)** under **Imperative Programming Model for OPC UA Alarms & Conditions (Section 5.1.4)**.
- **OPC UA Global Discovery Client (Section 6.4.1)**. Provides access to information model of an OPC UA Global Discovery Server (GDS).
- **OPC UA Certificate Management Client (Section 6.4.2)**. Provides access to certificate management (CM) information model of an OPC UA Global Discovery Server (GDS).
- **OPC UA Publish-Subscribe Client (Section 6.4.3)**. Provides access to information model in OPC UA PubSub.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

6.4.1 OPC UA Global Discovery Client

In This Topic

Example

The OPC UA Global Discovery Client is a specialized OPC UA client object for global discovery. In the OPC UA specifications, the functionality covered by this type is described in Part 12: Discovery and Global Services, in the Global Discovery Server section.

This specialized client object can be derived from IEasyUAClient Interface using the AsGlobalDiscoveryClient Extension Method, or it can be created standalone as EasyUAGlobalDiscoveryClient Component.

This interface contains methods to communicate with the OPC UA Global Discovery Server (GDS) using its information model for global discovery, making the work of assembling the proper OPC UA method parameters and retrieving the results. The methods allow querying for applications and/or servers, and registering, unregistering or updating application registrations in the GDS. Only the global discovery aspect of the GDS information is covered by this interface. Certificate management (CM), key credential management, and authorization services are treated separately.

> ⚠ Before deciding to use methods provided by this specialized client, it is recommended that you have a look at the IEasyUAClientApplication service (see **OPC UA Client Application Service (Section 6.3.1)**), available from IServiceProvider.GetService method from the IEasyUAClient interface. The client application service works on higher level, and conveniently combines several methods together to achieve the typically needed functionality related to GDS and certificate management in OPC UA.

Available methods are:

- FindApplications Method. Finds the application records for an OPC UA application known to the GDS.
- GetApplication Method. Gets a specific application record for an OPC UA application known to GDS.
- QueryApplications Method. Finds client or server applications that meet the specified filters.
- QueryServers Method. (Depreciated) Finds server applications that meet the specified filters.
- RegisterApplication Method. Registers a new application instance with a Global Directory Server.
- UnregisterApplication Method. Removes an application from a Global Discovery Server.
- UpdateApplication Method. Updates an existing application in a Global Discovery Server.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

Example

C#

```
// Shows how to check if an application needs to update its certificate.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Gds._EasyUACertificateManagementClient
{
    class GetCertificateStatus
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Register our client application with the GDS, so that we obtain an
application ID that we need later.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();
            UANodeId applicationId;
            try
            {
                applicationId = clientApplication.RegisterToGds(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine("Application ID: {0}", applicationId);

            // Instantiate the certificate management client object
            var certificateManagementClient = new EasyUACertificateManagementClient();

            // Check if the application needs to update its certificate.
            bool updateRequired;
            try
            {
                updateRequired =
certificateManagementClient.GetCertificateStatus(gdsEndpointDescriptor, applicationId,
                    UANodeId.Null, UANodeId.Null);
            }
            catch (UAException uaException)
            {
```

```
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Update required: {0}", updateRequired);
        }


        // Example output:
        //Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=aec94459-f513-4979-8619-8383555fca61
        //Update required: False
    }
}
```

Object Pascal

```
// Shows how to check if an application needs to update its certificate.

class procedure GetCertificateStatus.Main;
var
  ApplicationId: _UANodeId;
  CertificateManagementClient: OpcLabs_EasyOpcUA_TLB._EasyUACertificateManagementClient;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  NullNodeId: _UANodeId;
  UpdateRequired: boolean;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Register our client application with the GDS, so that we obtain an application ID that
we need later.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Create an application registration in the GDS, assigning it a new application ID.
  try
    ApplicationId := ClientApplication.RegisterToGds(GdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Application ID: ', ApplicationId.ToString);

  // Instantiate the certificate management client object
  CertificateManagementClient := CoEasyUACertificateManagementClient.Create;
```

```
  // Check if the application needs to update its certificate.
  NullNodeId := CoUANodeId.Create;
  UpdateRequired := false;
  try
    UpdateRequired :=
CertificateManagementClient.GetCertificateStatus(GdsEndpointDescriptor, ApplicationId,
NullNodeId, NullNodeId);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Update required: ', UpdateRequired);

  // Example output:
  //Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=aec94459-
f513-4979-8619-8383555fca61
  //Update required: FALSE

end;
```

VB.NET

```
' Shows how to check if an application needs to update its certificate.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUACertificateManagementClient
    Friend Class GetCertificateStatus
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Register our client application with the GDS, so that we obtain an
application ID that we need later.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Create an application registration in the GDS, assigning it a new application
ID.
            Dim applicationId As UANodeId
            Try
                applicationId = clientApplication.RegisterToGds(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
```

```
            Exit Sub
        End Try

        ' Instantiate the certificate management client object
        Dim certificateManagementClient = New EasyUACertificateManagementClient()

        ' Check if the application needs to update its certificate.
        Dim updateRequired As Boolean
        Try
            updateRequired =
certificateManagementClient.GetCertificateStatus(gdsEndpointDescriptor, applicationId,
                UANodeId.Null, UANodeId.Null)
        Catch uaException As UAException
            Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
            Exit Sub
        End Try

        ' Display results
        Console.WriteLine("Update required: {0}", updateRequired)
    End Sub
End Class
End Namespace
```

VBScript

```
Rem Shows how to check if an application needs to update its certificate.

Option Explicit

' Define which GDS we will work with.
Dim GdsEndpointDescriptor: Set GdsEndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
GdsEndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName = "appadmin"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password = "demo"

' Register our client application with the GDS, so that we obtain an application ID that we
need later.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ClientApplication: Set ClientApplication = _
    Client.GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA")
On Error Resume Next
Dim ApplicationId: Set ApplicationId =
ClientApplication.RegisterToGds(GdsEndpointDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "Application ID: " & ApplicationId

' Instantiate the certificate management client object
Dim CertificateManagementClient: Set CertificateManagementClient = _
    CreateObject("OpcLabs.EasyOpc.UA.Gds.EasyUACertificateManagementClient")

' Check if the application needs to update its certificate.
Dim NullNodeId: Set NullNodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
```

```
On Error Resume Next
Dim UpdateRequired: UpdateRequired = CertificateManagementClient.GetCertificateStatus( _
    GdsEndpointDescriptor, ApplicationId, NullNodeId, NullNodeId)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Update required: " & UpdateRequired


' Example output:
'Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=aec94459-f513-
4979-8619-8383555fca61
'Update required: False
```

6.4.2 OPC UA Certificate Management Client

The OPC UA Certificate Management Client is a specialized OPC UA client object providing access to certificate management (CM) information model of an OPC UA Global Discovery Server (GDS). In the OPC UA specifications, the functionality covered by this type is described in Part 12: Discovery and Global Services, in the section Certificate Management Overview.

This specialized client object can be derived from IEasyUAClient Interface using the AsCertificateManagementClient Extension Method, or it can be created standalone as EasyUACertificateManagementClient Component.

This interface contains methods to communicate with the OPC UA Global Discovery Server (GDS) using its information model for certificate management, making the work of assembling the proper OPC UA method parameters and retrieving the results. The methods allow getting certificate groups, requesting key pairs or certificate signing and obtaining the results, retrieving trust lists, and more.

> ⚠ Before deciding to use methods provided by this specialized client, it is recommended that you have a look at the IEasyUAClientApplication service (see **OPC UA Client Application Service (Section 6.3.1)**), available from IServiceProvider.GetService method from the IEasyUAClient interface. The client application service works on higher level, and conveniently combines several methods together to achieve the typically needed functionality related to GDS and certificate management in OPC UA.

Available methods are:

- FinishRequest Method. Finishes a certificate request started with a call to StartNewKeyPairRequest Method or StartSigningRequest Method.
- GetCertificateGroups Method. Returns the certificate groups assigned to application.
- GetCertificateStatus Method. Checks if an application needs to update its certificate.
- GetTrustList Method. Retrieves the node ID of a trust list assigned to an application.
- StartNewKeyPairRequest Method. Starts a request for a new certificate and private key. The certificate and private key are returned in the FinishRequest response.
- StartSigningRequest Method. Initiates a request to create a certificate which uses the private key which the caller currently has. The new certificate is returned in the FinishRequest response.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product

> Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

6.4.3 OPC UA Publish-Subscribe Client

Introduction

In This Topic

The OPC UA Publish-Subscribe Client is a specialized client object providing access to PubSub information model in OPC UA.

Note that although the OPC UA PubSub exchanges the data using its own protocols and can stay on its own, the information model provides additional useful integration with OPC UA infrastructure. The information model for OPC UA PubSub is accessed using "normal" OPC UA Client-Server technologies. This means that the OPC UA Publish-Subscribe Client accesses data in some "live" OPC UA server that must be available at the time your program makes the operations. Additionally, the OPC UA Publish-Subscribe Client gives you access to OPC UA PubSub configuration data stored in files.

This specialized client object can be derived from IEasyUAClient Interface using the AsPublishSubscribeClient Extension Method, or it can be created standalone as EasyUAPublishSubscribeClient Component.

> ⚠ The EasyUAPublishSubscribeClient Class class does **not** have OPC UA Subscriber functionality; if you need that, see EasyUASubscriber Class instead.

> 💡 If your intent is to use the PubSub configuration stored in the OPC UA information model to find data that would allow you to set up a PubSub subscription, you can use the built-in **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** mechanism instead. This will save you lots of work.

> Currently, the OPC UA Publish-Subscribe Client only provides a read-only view of the information model. That is, you cannot use it to change anything. In addition, only the parts relevant to **OPC UA PubSub logical resolution (Section 5.1.5.1.4)** are available (this means that the parts related to reader group and dataset reader configuration are left out).

Functionality

In order to access the PublishSubscribe element, which is at the top of the PubSub information model, use the GetPublishSubscribeElement Method. The PublishSubscribe element allows you to find out which transport profiles are supported by the publisher or subscriber associated with the server, using the SupportedTransportProfileUriStringSet Property.

The HasConfigurationModel Method determines whether the specified OPC UA server provides a PubSub configuration model. This allows you to make such test cleanly, without possibly causing an error (exception being thrown).

The "main" method of the OPC UA Publish-Subscribe client is the AccessReadOnlyConfiguration Method returns the IUAReadOnlyPubSubConfiguration Interface which provides read-only access to PubSub configuration provided by the specified server. If, instead of accessing the "live" configuration in the OPC server, you want to load the PubSub configuration from a specified file (in UABinary format), use the LoadReadOnlyConfiguration Method. The PubSub configuration interface then provides members described in **Accessing OPC UA PubSub Configuration Model (Section 5.1.5.2)**.

If your PubSub configuration is in the UABinary format, but not stored in a file, you can use

the UABinaryReadOnlyConfigurationLoader Property. It returns an instance of StreamLoader Class which you can use to load the configuration from an array of bytes, a Stream, a managed resource, and so on.

For more functionality, and detailed information, see the Reference: IEasyUAPublishSubscribeClient Interface, EasyUAPublishSubscribeClient Class and IEasyUAPublishSubscribeClientExtension Class.

Advanced: Loading PubSub configuration from a file node

Some OPC UA servers offer the PubSub configuration in form of a node representing the OPC UA file object. The file contains the PubSub configuration in the UABinary file, same as it would appear in a regular file (file on disk). This can be the only way the OPC UA server offers the PubSub configuration (possibly because the server is a small embedded server), or it can be in addition to the standard way of providing the PubSub configuration (which involves many related OPC UA nodes in a complex mesh). Representing the PubSub configuration is for many uses cases practical, but nevertheless non-standard practice. It appears mainly in servers built with Unified Automation toolkit(s).

OPC Data Client has ability to load such PubSub configuration and work with it. If you know that the PubSub configuration is stored in this way in the server, you can use the LoadReadOnlyConfigurationFromFileNode Method to load it. There is also the HasConfigurationFileNode Method to test whether the "well-known" node with PubSub configuration is present in the server.

For generic case (when you do not know how the PubSub configuration is stored in the server), we recommend using the AccessOrLoadReadOnlyConfiguration Method. This method can decide dynamically, based on the actual OPC UA server contents, which approach to use to provide the PubSub configuration interface to you. Note the possible differences in behavior between the PubSub configuration interfaces returned in each case, described further in this article.

Differences between "live" and loaded PubSub configurations

Accessing a PubSub configuration that is "live" in the OPC UA server always gives you the current status of the configuration, at every moment you call some operation on the PubSub configuration interface. This may be a good thing (if you are interested in up-to-date state), but also a bad thing (because OPC UA does not provide transactional access to the PubSub configuration, the information may change while you are working with different parts of it, and your code needs to be able to deal with it). Accessing a "live" PubSub configuration is generally slow (each operation requires one or more OPC UA service calls), and it may fail at any time (any operation may return an error caused by inability to perform the intended OPC UA service on the server).

In contrast, when you load the PubSub configuration from a file (or OPC UA file node in the server), it represents a one-time snapshot that does not change by itself. It resides in memory, and accessing it is fast. If used correctly, it also generally does not return any errors.

Examples

For easy set up, the examples provided here access OPC UA PubSub configuration data stored in a pre-made binary configuration file. The same principle can, however, be used to access configuration data provided by an OPC server. There are commented statements in the examples that show how to do it.

The example below starts at the "root" of the PubSub configuration, and first obtains names of all PubSub connections available (they are at the 1st level). For each PubSub connection, given its name, it obtains names of writer groups configured on that PubSub connection (they are at the 2nd level). And, for each such writer group, given its name, it obtains names of all dataset writers configured on that writer group (they are at the 3rd level). Besides the PubSub object names, the commented parts also show how to obtain more detailed information about each PubSub object.

C#

```
// This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
// OPC UA PubSub configuration.

using System;
using OpcLabs.BaseLib.Collections.Specialized;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions;

namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
{
    partial class GetMethods
    {
        public static void PubSubComponents()
        {
            // Instantiate the publish-subscribe client object.
            var publishSubscribeClient = new EasyUAPublishSubscribeClient();

            try
            {
                Console.WriteLine("Loading the configuration...");
                // Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                // specified that it has to be copied to the project's output
directory.
                IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary");

                // Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA Server
                // with appropriate information model.
                //IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                //
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010");

                // Get the names of PubSub connections in the configuration, regardless
of the folder they reside in.
                StringCollection pubSubConnectionNames =
pubSubConfiguration.ListPubSubConnectionNames();
                foreach (string pubSubConnectionName in pubSubConnectionNames)
                {
                    Console.WriteLine($"PubSub connection: {pubSubConnectionName}");

                    // You can use the statement below to obtain parameters of the
PubSub connection.
                    //UAPubSubConnectionElement connectionElement =
                    //
pubSubConfiguration.GetConnectionElement(pubSubConnectionName);

                    // Get names of the writer groups on this PubSub connection.
                    StringCollection writerGroupNames =
```

```
pubSubConfiguration.ListWriterGroupNames(pubSubConnectionName);
                    foreach (string writerGroupName in writerGroupNames)
                    {
                        Console.WriteLine($"  Writer group: {writerGroupName}");

                        // You can use the statement below to obtain parameters of the
writer group.
                        //UAWriterGroupElement writerGroupElement =
                        //
pubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName);

                        // Get names of the dataset writers on this writer group.
                        StringCollection dataSetWriterNames =

pubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName);
                        foreach (string dataSetWriterName in dataSetWriterNames)
                        {
                            Console.WriteLine($"    Dataset writer:
{dataSetWriterName}");

                            // You can use the statement below to obtain parameters of
the dataset writer.
                            //UADataSetWriterElement dataSetWriterElement =
pubSubConfiguration.GetDataSetWriterElement(
                            //    pubSubConnectionName, writerGroupName,
dataSetWriterName);
                        }
                    }
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine($"*** Failure:
{uaException.InnerException.Message}");
            }

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Loading the configuration...
        //PubSub connection: FixedLayoutConnection
        //  Writer group: FixedLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: AllTypesWriter
        //    Dataset writer: MassTestWriter
        //PubSub connection: DynamicLayoutConnection
        //  Writer group: DynamicLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: MassTestWriter
        //    Dataset writer: AllTypes-DynamicWriter
        //    Dataset writer: EventSimpleWriter
        //PubSub connection: FlexibleLayoutConnection
        //  Writer group: FlexibleLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: MassTestWriter
```

```
        //    Dataset writer: AllTypes-DynamicWriter
        //Finished.
    }
}
```

VB.NET

```
' This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
' OPC UA PubSub configuration.

Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
'Imports OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions

Namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
    Partial Friend Class GetMethods
        Public Shared Sub PubSubComponents()

            ' Instantiate the publish-subscribe client object.
            Dim publishSubscribeClient = New EasyUAPublishSubscribeClient()

            Try
                Console.WriteLine("Loading the configuration...")
                ' Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                ' specified that it has to be copied to the project's output directory.
                Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary")

                ' Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                ' Server with appropriate information model.
                'Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                '
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010")

                ' Get the names of PubSub connections in the configuration, regardless
of the folder they reside in.
                Dim pubSubConnectionNames =
pubSubConfiguration.ListPubSubConnectionNames()
                For Each pubSubConnectionName As String In pubSubConnectionNames
                    Console.WriteLine($"PubSub connection: {pubSubConnectionName}")

                    ' You can use the statement below to obtain parameters of the
PubSub connection.
                    'Dim connectionElement As UAPubSubConnectionElement =
                    '    pubSubConfiguration.GetConnectionElement(pubSubConnectionName)

                    ' Get names of the writer groups on this PubSub connection.
                    Dim writerGroupNames =
pubSubConfiguration.ListWriterGroupNames(pubSubConnectionName)
                    For Each writerGroupName As String In writerGroupNames
                        Console.WriteLine($"  Writer group: {writerGroupName}")
```

```
                        ' You can use the statement below to obtain parameters of the
writer group.
                        'Dim writerGroupElement As UAWriterGroupElement =
                        '
pubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName)

                        ' Get names of the dataset writers on this writer group.
                        Dim dataSetWriterNames =

pubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName)
                        For Each dataSetWriterName As String In dataSetWriterNames
                            Console.WriteLine($"    Dataset writer:
{dataSetWriterName}")

                            ' You can use the statement below to obtain parameters of
the dataset writer.
                            'Dim dataSetWriterElement As UADataSetWriterElement =
pubSubConfiguration.GetDataSetWriterElement(
                            '    pubSubConnectionName, writerGroupName,
dataSetWriterName)
                        Next dataSetWriterName
                    Next writerGroupName
                Next pubSubConnectionName
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Loading the configuration...
    'PubSub connection FixedLayoutConnection
    '  Writer group: FixedLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: AllTypesWriter
    '    Dataset writer: MassTestWriter
    'PubSub connection: DynamicLayoutConnection
    '  Writer group: DynamicLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: MassTestWriter
    '    Dataset writer: AllTypes-DynamicWriter
    '    Dataset writer: EventSimpleWriter
    'PubSub connection: FlexibleLayoutConnection
    '  Writer group: FlexibleLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: MassTestWriter
    '    Dataset writer: AllTypes-DynamicWriter
    'Finished.

End Namespace
```

Object Pascal

```
// This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
// OPC UA PubSub configuration.

class procedure GetMethods.PubSubComponents;
var
  //DataSetWriterElement: _UADataSetWriterElement;
  DataSetWriterName: string;
  DataSetWriterNames: _StringCollection;
  //EndpointDescriptor: _UAEndpointDescriptor;
  I, J, K: Integer;
  //PubSubConnectionElement: _UAPubSubConnectionElement;
  PubSubConnectionName: string;
  PubSubConnectionNames: _StringCollection;
  PublishSubscribeClient: _EasyUAPublishSubscribeClient;
  PubSubConfiguration: _UAReadOnlyPubSubConfiguration;
  //WriterGroupElement: _UAWriterGroupElement;
  WriterGroupName: string;
  WriterGroupNames: _StringCollection;
begin
  // Instantiate the publish-subscribe client object.
  PublishSubscribeClient := CoEasyUAPublishSubscribeClient.Create;

  try
    WriteLn('Loading the configuration...');
    // Load the PubSub configuration from a file. The file itself is included alongside
the script.
    PubSubConfiguration :=
PublishSubscribeClient.LoadReadOnlyConfiguration('UADemoPublisher-Default.uabinary');

    // Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    // with appropriate information model.
    //EndpointDescriptor := CoUAEndpointDescriptor.Create;
    //EndpointDescriptor.UrlString := 'opc.tcp://localhost:48010';
    //PubSubConfiguration :=
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor);

    // Get the names of PubSub connections in the configuration.
    PubSubConnectionNames := PubSubConfiguration.ListPubSubConnectionNames;

    for I := 0 to PubSubConnectionNames.Count-1 do
    begin
      PubSubConnectionName := PubSubConnectionNames[I];
      WriteLn('PubSub connection: ', PubSubConnectionName);

      // You can use the statement below to obtain parameters of the PubSub connection.
      //PubSubConnectionElement :=
PubSubConfiguration.GetConnectionElement(PubSubConnectionName);

      // Get names of the writer groups on this PubSub connection.
      WriterGroupNames :=
PubSubConfiguration.ListWriterGroupNames(PubSubConnectionName);
      for J := 0 to WriterGroupNames.Count-1 do
```

```
      begin
        WriterGroupName := WriterGroupNames[J];
        WriteLn('  Writer group: ', WriterGroupName);

        // You can use the statement below to obtain parameters of the writer group.
        //WriterGroupElement :=
PubSubConfiguration.GetWriterGroupElement(PubSubConnectionName, WriterGroupName);

        // Get names of the dataset writers on this writer group.
        DataSetWriterNames :=
PubSubConfiguration.ListDataSetWriterNames(PubSubConnectionName, WriterGroupName);
        for K := 0 to DataSetWriterNames.Count-1 do
        begin
          DataSetWriterName := DataSetWriterNames[K];
          WriteLn('    Dataset writer: ', DataSetWriterName);

          // You can use the statement below to obtain parameters of the dataset
writer.
          //DataSetWriterElement :=
PubSubConfiguration.GetDataSetWriterElement(PubSubConnectionName, WriterGroupName,
DataSetWriterName);
        end;
      end;
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Finished.');
end;

// Example output:
//
//Loading the configuration...
//PubSub connection: FixedLayoutConnection
//  Writer group: FixedLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: AllTypesWriter
//    Dataset writer: MassTestWriter
//PubSub connection: DynamicLayoutConnection
//  Writer group: DynamicLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: MassTestWriter
//    Dataset writer: AllTypes-DynamicWriter
//    Dataset writer: EventSimpleWriter
//PubSub connection: FlexibleLayoutConnection
//  Writer group: FlexibleLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: MassTestWriter
//    Dataset writer: AllTypes-DynamicWriter
//Finished.
```

VBScript

```
Rem This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
Rem OPC UA PubSub configuration.

Option Explicit

' Instantiate the publish-subscribe client object.
Dim PublishSubscribeClient: Set PublishSubscribeClient =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.InformationModel.EasyUAPublishSubscribeClient")

On Error Resume Next
DumpPubSubComponents
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Finished."



Sub DumpPubSubComponents()
    WScript.Echo "Loading the configuration..."
    ' Load the PubSub configuration from a file. The file itself is included alongside
the script.
    Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-Default.uabinary")

    ' Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    ' with appropriate information model.
    'Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
    'EndpointDescriptor.UrlString = "opc.tcp://localhost:48010"
    'Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor)

    ' Get the names of PubSub connections in the configuration.
    Dim PubSubConnectionNames: Set PubSubConnectionNames =
PubSubConfiguration.ListPubSubConnectionNames
    Dim pubSubConnectionName: For Each pubSubConnectionName In PubSubConnectionNames
        WScript.Echo "PubSub connection: " & pubSubConnectionName

        ' You can use the statement below to obtain parameters of the PubSub
connection.
        'Dim PubSubConnectionElement: Set PubSubConnectionElement =
PubSubConfiguration.GetConnectionElement(pubSubConnectionName)

        ' Get names of the writer groups on this PubSub connection.
        Dim WriterGroupNames: Set WriterGroupNames =
PubSubConfiguration.ListWriterGroupNames(pubSubConnectionName)
        Dim writerGroupName: For Each writerGroupName In WriterGroupNames
            WScript.Echo "  Writer group: " & writerGroupName

            ' You can use the statement below to obtain parameters of the writer group.
            'Dim WriterGroupElement: Set WriterGroupElement =
```

```
PubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName)

            ' Get names of the dataset writers on this writer group.
            Dim DataSetWriterNames: Set DataSetWriterNames =
PubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName)
            Dim dataSetWriterName: For Each dataSetWriterName In DataSetWriterNames
                WScript.Echo "    Dataset writer: " & dataSetWriterName

                ' You can use the statement below to obtain parameters of the dataset
writer.
                'Dim DataSetWriterElement: Set DataSetWriterElement = _
                '    PubSubConfiguration.GetDataSetWriterElement(pubSubConnectionName,
writerGroupName, dataSetWriterName)
            Next
        Next
    Next
End Sub



' Example output:
'
'Loading the configuration...
'PubSub connection: FixedLayoutConnection
'  Writer group: FixedLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: AllTypesWriter
'    Dataset writer: MassTestWriter
'PubSub connection: DynamicLayoutConnection
'  Writer group: DynamicLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: MassTestWriter
'    Dataset writer: AllTypes-DynamicWriter
'    Dataset writer: EventSimpleWriter
'PubSub connection: FlexibleLayoutConnection
'  Writer group: FlexibleLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: MassTestWriter
'    Dataset writer: AllTypes-DynamicWriter
'Finished.
```

The example below uses the ListAllPublishedDataSetNames Method to retrieve all published dataset from the PubSub configuration. The published datasets are actually organized in the PubSub configuration using a hierarchical structure of dataset folders. The word "all" in the method name denotes that the method will truly return all published datasets in the configuration, not just those at the "root" of the dataset folder structure, or in any specified dataset folder. The method will act recursively in the dataset folder structure, if needed. There are also methods that allow you to work with contents of individual dataset folders. Note that the published dataset names are unique across the PubSub configuration as a whole (regardless of their location in the dataset folder structure), and therefore a published dataset name is sufficient to identify the published dataset, and the dataset folder path is not strictly necessary.

C#

```
// This example obtains and prints out information about all published datasets in the
```

```
OPC UA PubSub configuration.

using System;
using OpcLabs.BaseLib.Collections.Specialized;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions;

namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
{
    partial class GetMethods
    {
        public static void PublishedDataSets()
        {
            // Instantiate the publish-subscribe client object.
            var publishSubscribeClient = new EasyUAPublishSubscribeClient();

            try
            {
                Console.WriteLine("Loading the configuration...");
                // Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                // specified that it has to be copied to the project's output
directory.
                IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary");

                // Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                // Server with appropriate information model.
                //IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                //
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010");

                // Get the names of all published datasets in the PubSub configuration.
                StringCollection publishedDataSetNames =
pubSubConfiguration.ListAllPublishedDataSetNames();

                foreach (string publishedDataSetName in publishedDataSetNames)
                {
                    Console.WriteLine($"Published dataset: {publishedDataSetName}");

                    // You can use the statement below to obtain parameters of the
published dataset.
                    //UAPublishedDataSetElement publishedDataSetElement =
                    //
pubSubConfiguration.GetPublishedDataSetElement(publishedDataSetName);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine($"*** Failure:
{uaException.InnerException.Message}");
            }
```

```
            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Loading the configuration...
        //Published dataset: Simple
        //Published dataset: AllTypes
        //Published dataset: MassTest
        //Published dataset: AllTypes-Dynamic
        //Finished.
    }
}
```

VB.NET

```
' This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions

Namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
    Partial Friend Class GetMethods
        Public Shared Sub PublishedDataSets()

            ' Instantiate the publish-subscribe client object.
            Dim publishSubscribeClient = New EasyUAPublishSubscribeClient()

            Try
                Console.WriteLine("Loading the configuration...")
                ' Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                ' specified that it has to be copied to the project's output directory.
                Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary")

                ' Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                ' Server with appropriate information model.
                'Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                '
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010")

                ' Get the names of all published datasets in the PubSub configuration.
                Dim publishedDataSetNames =
pubSubConfiguration.ListAllPublishedDataSetNames()

                For Each publishedDataSetName As String In publishedDataSetNames
                    Console.WriteLine($"Published dataset: {publishedDataSetName}")

                    ' You can use the statement below to obtain parameters of the
```

```
published dataset.
                    'Dim publishedDataSetElement As UAPublishedDataSetElement =
                    '
pubSubConfiguration.GetPublishedDataSetElement(publishedDataSetName)
                Next publishedDataSetName
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Loading the configuration...
    'Published dataset Simple
    'Published dataset: AllTypes
    'Published dataset: MassTest
    'Published dataset: AllTypes-Dynamic
    'Finished.

End Namespace
```

Object Pascal

```
// This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

class procedure GetMethods.PublishedDataSets;
var
  //EndpointDescriptor: _UAEndpointDescriptor;
  I: Integer;
  PublishedDataSetName: string;
  PublishedDataSetNames: _StringCollection;
  //PublishedDataSetElement: _UAPublishedDataSetElement;
  PublishSubscribeClient: _EasyUAPublishSubscribeClient;
  PubSubConfiguration: _UAReadOnlyPubSubConfiguration;
begin
  // Instantiate the publish-subscribe client object.
  PublishSubscribeClient := CoEasyUAPublishSubscribeClient.Create;

  try
    WriteLn('Loading the configuration...');
    // Load the PubSub configuration from a file. The file itself is included alongside
the script.
    PubSubConfiguration :=
PublishSubscribeClient.LoadReadOnlyConfiguration('UADemoPublisher-Default.uabinary');

    // Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    // with appropriate information model.
    //EndpointDescriptor := CoUAEndpointDescriptor.Create;
    //EndpointDescriptor.UrlString := 'opc.tcp://localhost:48010';
    //PubSubConfiguration :=
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor);
```

```
    // Get the names of PubSub connections in the configuration, regardless of the
folder they reside in.
    PublishedDataSetNames := PubSubConfiguration.ListAllPublishedDataSetNames;

    for I := 0 to PublishedDataSetNames.Count-1 do
    begin
      PublishedDataSetName := PublishedDataSetNames[I];
      WriteLn('Published dataset: ', PublishedDataSetName);

      // You can use the statement below to obtain parameters of the published dataset.
      //PublishedDataSetElement :=
PubSubConfiguration.GetPublishedDataSetElement(Unassigned, PublishedDataSetName);
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Finished.');
end;

// Example output:
//
//Loading the configuration...
//Published dataset: Simple
//Published dataset: AllTypes
//Published dataset: MassTest
//Published dataset: AllTypes-Dynamic
//Finished.
```

VBScript

```
Rem This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

Option Explicit

' Instantiate the publish-subscribe client object.
Dim PublishSubscribeClient: Set PublishSubscribeClient =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.InformationModel.EasyUAPublishSubscribeClient")

On Error Resume Next
DumpPublishedDataSets
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Finished."



Sub DumpPublishedDataSets()
```

```
    WScript.Echo "Loading the configuration..."
    ' Load the PubSub configuration from a file. The file itself is included alongside
the script.
    Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-Default.uabinary")

    ' Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    ' with appropriate information model.
    'Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
    'EndpointDescriptor.UrlString = "opc.tcp://localhost:48010"
    'Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor)

    ' Get the names of PubSub connections in the configuration, regardless of the
folder they reside in.
    Dim PublishedDataSetNames: Set PublishedDataSetNames =
PubSubConfiguration.ListAllPublishedDataSetNames

    Dim publishedDataSetName: For Each publishedDataSetName In PublishedDataSetNames
        WScript.Echo "Published dataset: " & publishedDataSetName

        ' You can use the statement below to obtain parameters of the published
dataset.
        'Dim PublishedDataSetElement: Set PublishedDataSetElement =
PubSubConfiguration.GetPublishedDataSetElement(Nothing, publishedDataSetName)
    Next
End Sub



' Example output:
'
'Loading the configuration...
'Published dataset: Simple
'Published dataset: AllTypes
'Published dataset: MassTest
'Published dataset: AllTypes-Dynamic
'Finished.
```

⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

7 Extensions

7.1 Layered Extensions for .NET

EasyOPC Extensions for .NET is a pack of .NET classes that adds more functionality to the EasyOPC.NET and EasyOPC-UAcomponents. It is built upon and relies on EasyOPC.NET and EasyOPC-UA, so in fact you could build all these extensions yourself, but it is meant to provide a ready-made, verified code for useful “shortcut” methods to achieve common tasks.

Technically, **Specialized Client Objects (Section 6.4)** are also similar to layered extensions.

7.1.1 Usage

A significant part of the EasyOPC Extensions for .NET functionality is provided in form of so-called Extension Methods. In languages that support them (including C#, VB.NET), extension methods will appear as additional methods on the classes that are being extended. For example, the EasyDAClient class appears as having many more new methods that you can choose from. This way, you can write a code that call e.g. GetDataTypePropertyValue (extension) method on the EasyDAClient object, although the method is actually located in the IEasyDAClientExtension2 class which you do not even have to be aware of.

In languages that do not support extension method syntax, it is still possible to use them, but they need to be called as static method on the extension class, and you provide the object reference as an additional argument. In the above example, you would call IEasyDAClientExtension2**.**GetDataTypePropertyValue instead, and pass it the EasyDAClient object as the first argument.

7.1.1.1 Generic Types

Some methods (described further below) that provide generic access return values (or arrays of values) of type ValueResult<T>. ValueResult<T> is a type derived from ValueResult that holds returns with values of type T. Attempt to store a value of incompatible type causes an exception. This means that you specify the type of the value you expect from OPC, and safely assume that the value of given type will be returned.

Further generic types exist for various OPC specifications. See e.g. Generic Types for OPC Data Access.

7.1.2 OPC Data Access Extensions

This chapter describes EasyOPC.NET extensions for OPC Data Access and OPC XML-DA.

7.1.2.1 Generic Types for OPC-DA

Following generic types are available for use with OPC Data Access and OPC XML-DA:

- DAVtq<T>: Holds typed data value (of type T), timestamp, and quality.
- DAVtqResult<T>: Holds result of an operation in form of a typed DAVtq<T> (value, timestamp, quality).
- DAItemValueArguments<T>: Arguments for an operation on an OPC item. Carries a value of type T.
- DAItemVtqArguments<T>: Arguments for an operation on an OPC item. Carries a DAVtq<T> object.

- EasyDAItemSubscriptionArguments<T>: Arguments for subscribing to an OPC item of type T.
- EasyDAItemChangedEventArgs<T>: Data of an event or callback for a significant change in OPC item of type T.

These types are used with generic access extensions for OPC items (see Generic Access) and with generic access extensions for OPC properties (see Generic Access). They are also use with the Live Mapping Model.

For example, you can call the EasyDAClient.ReadItem<T> extension method with a given type T, and receive back a DAVtq<T> object with a TypedValue already properly typed as T.

Noted that if the untyped property on the base (non-generic) type has a name like **XXXX**, the corresponding typed property on the generic type has a name like TypedXXXX. For example, for an untyped property DAVtqResult.Vtq, there is a corresponding typed property DAVtqResult<T>.TypedVtq.

7.1.2.2 Extensions for OPC Items

7.1.2.2.1 Type-safe Access

With EasyOPC.NET Extensions, you can use type-safe methods that allow reading an item value already converted to the specified type, with methods such as EasyDAClient.ReadItemValueInt32. There is such a method for each primitive type, named ReadItemValueXXXX, where **XXXX** is the name of the type. Using these methods allows your code be free of unnecessary conversions. The methods also take care of passing a proper requested data type to the OPC server.

A corresponding set of methods also exists for one-dimensional arrays of primitive types. Such methods are named ReadItemValueArrayOfXXXX, where **XXXX** is the name of the element type. For example, EasyDAClient.ReadItemValueArrayOfInt32 will read from an item as an array of 32-bit signed integers.

You can also use type-safe methods that allow writing an item value of a specified type, with methods such as EasyDAClient.WriteItemValueInt32. There is such a method for each primitive type, named WriteItemValueXXXX, where **XXXX** is the name of the type. Using these methods allows your code be free of unnecessary conversions. The methods also take care of passing a proper requested data type to the OPC server.

A corresponding set of methods also exists for one-dimensional arrays of primitive types. Such methods are named WriteItemValueArrayOfXXXX, where **XXXX** is the name of the element type. For example, EasyDAClient.WriteItemValueArrayOfInt32 will write into an item as an array of 32-bit signed integers.

7.1.2.2.2 Generic Access

The EasyDAClient.ReadItem<T> extension method reads an OPC item of a given type T, and return the result as DAVtq<T> (typed value/timestamp/quality) object.

The EasyDAClient.ReadItemValue<T> extension method reads an OPC item of a given type T, and return the actual item's value typed as T.

The EasyDAClient.WriteItem<T> extension method writes a typed value/timestamp/quality into an OPC item, using DAVtq<T> object.

The EasyDAClient.WriteItemValue<T> extension method writes a typed value of type T into an OPC item.

There are several overloads of all the above described methods, with different structure of arguments.

7.1.2.3 Extensions for OPC Properties

7.1.2.3.1 Type-safe Access

With EasyOPC.NET Extensions, you can use type-safe methods that allow obtaining a value of an OPC property value already converted to the specified type, with methods such as EasyDAClient.GetPropertyValueInt32. There is such a method for each primitive type, named GetPropertyValueXXXX, where **XXXX** is the name of the type. Using these methods allows your code be free of unnecessary conversions.

A corresponding set of methods also exists for one-dimensional arrays of primitive types. Such methods are named GetPropertyValueArrayOfXXXX, where **XXXX** is the name of the element type. For example, EasyDAClient.GetPropertyValueArrayOfString will obtain a value of a property as an array of strings.

7.1.2.3.2 Generic Access

The EasyDAClient.GetPropertyValue<T> extension method gets a value of OPC property of a given type T, and return the actual value typed as T. There are several overloads of this method with different structure of arguments.

7.1.2.3.3 Well-known Properties

A common scenario is to work with well-known OPC properties. With EasyOPC.NET Extensions, you can quickly obtain a value of any well-known OPC property of a given OPC item, with methods such as EasyDAClient.GetDataTypePropertyValue. All these methods are named GetXXXXPropertyValue, where **XXXX** is the name of the property. The methods also check the value type and convert it to the type that corresponds to the property. For example, GetDataTypePropertyValue method returns a VarType, GetScanRatePropertyValue method returns a float, and GetDescriptionPropertyValue method returns a string.

C#

```
// This example shows how to obtain a data type of an OPC item.

using System;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Extensions;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientExtension
{
    class GetDataTypePropertyValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            // Get the DataType property value, already converted to VarType
            VarType varType;
            try
            {
                varType = client.GetDataTypePropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random");
            }
```

```
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            // Display the obtained data type
            Console.WriteLine("VarType: {0}", varType); // Display data type
symbolically
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of an OPC item.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Extensions
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientExtension
    Friend Class GetDataTypePropertyValue
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            ' Get the DataType property value, already converted to VarType
            Dim varType As VarType
            Try
                varType = client.GetDataTypePropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Display the obtained data type
            Console.WriteLine("VarType: {0}", varType) ' Display data type symbolically
        End Sub
    End Class
End Namespace
```

7.1.2.3.4 Alternate Access Methods

The GetPropertyValueDictionary method allows you to obtain a dictionary of property values for a given OPC item, where a key to the dictionary is the property Id. You can pass in a set of property Ids that you are interested in, or have the method obtain all well-known OPC properties. You can then easily extract the value of any property by looking it up in a dictionary (as opposed to having to numerically index into an array, as with the base GetMultiplePropertyValues method).

C#

```
// This example shows how to obtain a dictionary of OPC property values for an OPC
item, and extract property values.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Extensions;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientExtension
{
    class GetPropertyValueDictionary
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            // Get dictionary of property values, for all well-known properties
            DAPropertyValueDictionary propertyValueDictionary;
            try
            {
                propertyValueDictionary = client.GetPropertyValueDictionary("",
"OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            // Display some of the obtained property values
            // The production code should also check for the .Exception first, before
getting .Value

Console.WriteLine("propertyValueDictionary[DAPropertyId.AccessRights].Value: {0}",
                propertyValueDictionary[DAPropertyIds.AccessRights].Value);
            Console.WriteLine("propertyValueDictionary[DAPropertyId.DataType].Value:
{0}",
                propertyValueDictionary[DAPropertyIds.DataType].Value);
            Console.WriteLine("propertyValueDictionary[DAPropertyId.Timestamp].Value:
{0}",
                propertyValueDictionary[DAPropertyIds.Timestamp].Value);
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a dictionary of OPC property values for an OPC item,
and extract property values.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Extensions
Imports OpcLabs.EasyOpc.OperationModel
```

```
Namespace DocExamples.DataAccess._EasyDAClientExtension
    Friend Class GetPropertyValueDictionary
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            ' Get dictionary of property values, for all well-known properties
            Dim propertyValueDictionary As DAPropertyValueDictionary
            Try
                propertyValueDictionary = client.GetPropertyValueDictionary("",
"OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Display some of the obtained property values
            ' The production code should also check for the .Exception first, before
getting .Value

Console.WriteLine("propertyValueDictionary[DAPropertyId.AccessRights].Value: {0}",
propertyValueDictionary(DAPropertyIds.AccessRights).Value)
            Console.WriteLine("propertyValueDictionary[DAPropertyId.DataType].Value:
{0}", propertyValueDictionary(DAPropertyIds.DataType).Value)
            Console.WriteLine("propertyValueDictionary[DAPropertyId.Timestamp].Value:
{0}", propertyValueDictionary(DAPropertyIds.Timestamp).Value)
        End Sub
    End Class
End Namespace
```

The GetItemPropertyRecord method allows you to obtain a structure filled in with property values for a given OPC item. It can retrieve all well-known properties at once, or you can pass in a set of property Ids that you are interested in. You can then simply use the properties in the resulting DAItemPropertyRecord structure, without further looking up the values in any way.

C#

```
// This example shows how to obtain a structure containing property values for an OPC
item, and display some property values.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Extensions;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientExtension
{
    class GetItemPropertyRecord
    {
        public static void Main1()
        {
            var client = new EasyDAClient();
```

```
            // Get a structure containing values of all well-known properties
            DAItemPropertyRecord itemPropertyRecord;
            try
            {
                itemPropertyRecord = client.GetItemPropertyRecord("",
"OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            // Display some of the obtained property values
            Console.WriteLine("itemPropertyRecord.AccessRights: {0}",
itemPropertyRecord.AccessRights);
            Console.WriteLine("itemPropertyRecord.DataType: {0}",
itemPropertyRecord.DataType);
            Console.WriteLine("itemPropertyRecord.Timestamp: {0}",
itemPropertyRecord.Timestamp);
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a structure containing property values for an OPC
item, and display some property values.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Extensions
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientExtension
    Friend Class GetItemPropertyRecord
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            ' Get a structure containing values of all well-known properties
            Dim itemPropertyRecord As DAItemPropertyRecord
            Try
                itemPropertyRecord = client.GetItemPropertyRecord("",
"OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Display some of the obtained property values
            Console.WriteLine("itemPropertyRecord.AccessRights: {0}",
itemPropertyRecord.AccessRights)
            Console.WriteLine("itemPropertyRecord.DataType: {0}",
itemPropertyRecord.DataType)
            Console.WriteLine("itemPropertyRecord.Timestamp: {0}",
```

```
itemPropertyRecord.Timestamp)
        End Sub
    End Class
End Namespace
```

The static DAPropertyIDSet class gives you an easy way to provide pre-defined sets of properties to the above methods. There are well-known sets such as the Basic property set, Extension set, or Alarms and Events property set. It also allows you to combine the property sets together (a union operation), with the Add method or the '+' operator.

7.1.3 OPC Alarms&Events Extensions

This chapter describes EasyOPC.NET extensions for OPC Alarms&Events.

> This functionality is not available under (or the text does not apply to) .NET Standard development platform.

7.1.3.1 OPC-A&E Queries

The extension method EasyAEClient.FindEventCategory attempts to find an event category given by its numerical category ID in the OPC server. If successful, it provides the AECategoryElement filled in with details about the event category.

The extension method EasyAEClient.FindEventCondition attempts to find an event condition in the OPC server, given the numerical category ID, and event condition name. If successful, it provides the AEConditionElement filled in with details about the event condition.

7.1.4 OPC Unified Architecture Extensions

This chapter describes EasyOPC extensions for OPC Unified Architecture (OPC-UA).

7.1.4.1 Extensions on Helper Types

The UAApplicationTypes.IsServer extension method allows you to determine whether a UA application (as returned e.g. by server discovery) includes OPC-UA server functionality. This is a shortcut for testing whether the application type is either Server or ClientOrServer.

The extensions on the UANodeElement (as returned e.g. by browsing for OPC-UA nodes) allow finer categorization of the nodes over what is provided just by the NodeClass property. For example, you can test whether a node is a Data Variable or a Property, by extensions methods IsDataVariable and IsProperty.

7.1.4.2 Extensions for OPC UA reading

The ReadDataType Method reads a value of DataType attribute of a node.

The ReadMultipleDataTypes Method reads value of DataType attribute of multiple nodes, using array of argument objects

as an input.

7.2 Integrated Extensions

Integrated extensions are extensions whose fuctions resides inside the objects you use. That is, they modify or extend the behavior of existing objects. The advantage of such approach is that the new functionality can be added transparently, and the code that uses them does have (in many cases) be aware of their existence or look any different.

7.2.1 OPC UA Complex Data Extension

In OPC UA, servers can stick to the standard types defined in the OPC UA specifications, or they can define their own types. The "non-standard" data types can be completely server-specific, or can conform to some industry standard. Usage of these data types is commonly referred to as *OPC UA complex data*. With complex data, it is possible that the data type is not even know at compile time; instead, it can be discovered, constructed or processed during runtime, without prior knowledge of it.

Use of OPC UA complex data poses a new range of interoperability problems. When the server uses some new data type, how can the client know what the data type is, and how is the data encoded? The data is basically a binary blob that is transferred as a block between the server and the client, and who knows what it means? In some cases, the client can be hard-coded with upfront knowledge of what this binary blob of data (called *extension object*) contains and how it is encoded. But that is only when the client can be written with specific knowledge about the server it connects to, and even then decoding of the data might be a tedious programming task. Thankfully, OPC UA specifications provide ways for servers to describe the data types and their encoding to the clients, using information that can reside in the server itself. This is done

- using so-called *data type dictionaries*, and
- in OPC UA version 1.04 and later also using so-called *data type definitions*.

Knowledge Base: Comparison of OPC UA complex data models .

In OPC Data Client, both models work together and in a unified way from developer's perspective. OPC client can then extract the data type descriptions from these dictionaries or definitions, and be written in a generic way that adjusts to the data types that exist in the server.

Retrieving the associated metadata from the server's information model, interpreting the data type dictionaries or data type definitions, and decoding and encoding data according to data type descriptions is quite complicated. OPC Data Client can do it for you, with the help of the OPC UA Complex Data extension, described in this chapter. With the OPC UA Complex Data extension, you do not have to deal with binary blobs; instead, you work with meaningful structures that represent the data types and the data itself logically.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

The OPC UA Complex Data extension integrates into the EasyUAClient Class pipeline, and its functionality is thus transparently available both to .NET and COM programming languages and tools.

7.2.1.1 Basic OPC UA Complex Data operations

The OPC UA Complex Data extension does not require you to perform OPC operations any differently when complex data is involved. You use the same objects and methods as with the regular data. The only difference is in the data that is being passed to or from the component. When the OPC UA Complex Data extension is enabled (which is by default), complex data returned to you by the methods on the EasyUAClient Class is represented by instances of the UAGenericObject Class, and conversely, if you want to pass complex data to some method on the EasyUAClient Class, you need to prepare an instance of UAGenericObject Class.

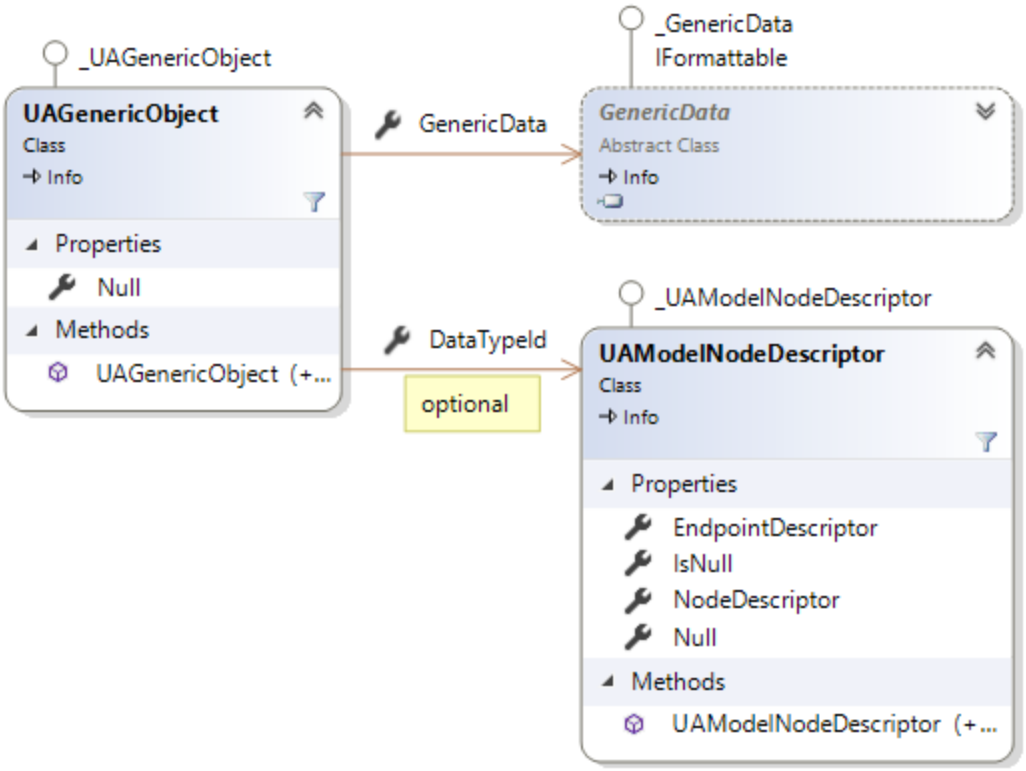
In This Topic

If the node being read, written or subscribed to represents an array (one- or multi-dimensional), the value passed to or from the component is then an array of UAGenericObject-s. Note that this is different from having a scalar node, and a UAGenericObject with generic data in that itself is an array (a sequence data). Both these options are possible (and sometimes a bit confusing).

The UAGenericObject class

The UAGenericObject Class is used to represent complex data on its way from the server or to the server. It has two basic pieces of information:

- The generic data itself, in the GenericData Property. This can be, for example, a primitive type (such as an integer), or a sequence (array), or a structured type with several fields of various types. The OPC UA Complex Data extension gives you methods to process generic data, and to create it. For more information, see **Generic data and data types (Section 7.2.1.2)**.
- An optional data type ID, in the DataTypeId Property. In most cases you do not need to be concerned about the data type ID. It is filled in when an instance of the UAGenericObject Class is returned from the server by the component. When you are sending generic data to the server through the component, you can usually leave this property unfilled, except in some situations where sub-types are involved. For more information, see **Role of the data type ID (Section 7.2.1.1.4)**.



Steps to use complex data

Here is what you need to do in order to work with OPC UA complex data.

- To read complex data information: Call one of the **Read** methods on the EasyUAClient Class as usual. When complex data is returned by the client, you will receive a value that is of type UAGenericObject Class, and you will process its contents. For more information, see **OPC UA Complex Data reading (Section 7.2.1.1.1)**.
- To subscribe to complex data information: Call one of the **Subscribe** methods on the EasyUAClient Class as usual. When complex data is returned by the client, a DataChangeNotification Event or EventNotification Event will be raised (or a corresponding callback will be made). The value in the event arguments will be of type UAGenericObject Class, and you will process its contents. For more information, see **OPC UA Complex Data subscribing (Section 7.2.1.1.2)**.
- To write a complex data information: Prepare an value of the UAGenericObject Class, and call one of the **Write** methods on the EasyUAClient Class as usual. For more information, see **OPC UA Complex Data writing (Section 7.2.1.1.3)**.

With the OPC UA Complex Data extension, it is also possible to call OPC UA methods that have complex data in their input arguments, or that return complex data in their output arguments. You will use a usual call to one of the **Call** methods on the EasyUAClient Class again.

7.2.1.1.1 OPC UA Complex Data reading

To read complex data information, call one of the **Read** methods on the EasyUAClient Class as usual. When complex data is returned by the client, you will receive a value that is of type UAGenericObject Class, and you will process its contents. The actual generic data is in its GenericData Property.

How precisely you process the received generic data depends on the application. In general, there can be two usage cases:

- Your application knows the type of the data to expect. In this case, you can hard-code this knowledge into the application. For example, if you know that the received data is a structure that has a field called "Temperature". You would type-cast the received data to StructuredData, and from the FieldData collection, you will get an element indexed by a string "Temperature". This can then be an instance of PrimitiveData Class, containing the temperature value in its Value Property. For more information, see **Working with generic data (Section 7.2.1.2.1)**.
- Your application does not know the type of the data to expect. This requires more coding, but is also perfectly doable. You will be accessing the generic data information similarly to the above case, but in a more generic way. The GenericData Class gives you a DataType Property, and the DataType object contained in this property gives you the metadata needed. For more information, see **Working with data types (Section 7.2.1.2.2)**.

The example below shows how to read and display OPC UA complex data.

C#

```
// Shows how to read complex data with OPC UA Complex Data plug-in.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._EasyUAClient
{
    class ReadValue
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; // [ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node which returns complex data. This is done in the same way as regular reads - just the data
            // returned is different.
            object value;
            try
            {
                value = client.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display basic information about what we have read.
            Console.WriteLine(value);


            // We know that this node returns complex data, so we can type cast to UAGenericObject.
            var genericObject = (UAGenericObject) value;

            // The actual data is in the GenericData property of the UAGenericObject.
            //
            // If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
            // specifier. In the debugger, you can view the same by displaying the private DebugView property.
            Console.WriteLine();
            Console.WriteLine("{0:V}", genericObject.GenericData);

            // For processing the internals of the data, refer to examples for GenericData and DataType classes.


            // Example output (truncated):
            //
            //(ScalarValueDataType) structured
            //
            //(ScalarValueDataType) structured
            //   [BooleanValue] (Boolean) primitive; True {System.Boolean}
            //   [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
            //   [ByteValue] (Byte) primitive; 153 {System.Byte}
            //   [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
            //   [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
            //   [EnumerationValue] (Int32) primitive; 0 {System.Int32}
            //   [ExpandedNodeIdValue] (ExpandedNodeId) structured
            //     [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
            //     [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
```

```
            //    [NodeIdType] (NodeIdType) enumeration; 3 (String)
            //    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
            //    [String] (StringNodeId) structured
            //      [Identifier] (CharArray) primitive; "????" {System.String}
            //      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
            //  [FloatValue] (Float) primitive; 78.37176 {System.Single}
            //  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
            //  [Int16Value] (Int16) primitive; 2793 {System.Int16}
            //  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
            //  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
            //  [Integer] (Variant) structured
            //    [ArrayDimensionsSpecified] sequence[1]
            //      [0] (Bit) primitive; False {System.Boolean}
            //    [ArrayLengthSpecified] sequence[1]
            //      [0] (Bit) primitive; False {System.Boolean}
            //    [Int64] sequence[1]
            //      [0] (Int64) primitive; 0 {System.Int64}
            //    [VariantType] sequence[6]
            //      [0] (Bit) primitive; False {System.Boolean}
            //      [1] (Bit) primitive; False {System.Boolean}
            //      [2] (Bit) primitive; False {System.Boolean}
            //      [3] (Bit) primitive; True {System.Boolean}
            //      [4] (Bit) primitive; False {System.Boolean}
            //      [5] (Bit) primitive; False {System.Boolean}
            //  [LocalizedTextValue] (LocalizedText) structured
            //    [Locale] (CharArray) primitive; "ko" {System.String}
            //    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
            //    [Reserved1] sequence[6]
            //      [0] (Bit) primitive; False {System.Boolean}
            //      [1] (Bit) primitive; False {System.Boolean}
            //      [2] (Bit) primitive; False {System.Boolean}
            //      [3] (Bit) primitive; False {System.Boolean}
            //      [4] (Bit) primitive; False {System.Boolean}
            //      [5] (Bit) primitive; False {System.Boolean}
            //    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
            //    [TextSpecified] (Bit) primitive; True {System.Boolean}
            //  [NodeIdValue] (NodeId) structured
        }
    }
}
```

Object Pascal

```
// Shows how to read complex data with OPC UA Complex Data plug-in.

class procedure ReadValue.Main;
var
  Client: _EasyUAClient;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
  Value: OleVariant;
begin
  // Define which server and node we will work with.
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  // [ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node which returns complex data. This is done in the same way as regular reads - just the data
  // returned is different.

  try
    Value := Client.ReadValue(EndpointDescriptor, NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display basic information about what we have read.
  WriteLn('value: ', Value);

  // We know that this node returns complex data, so it is a UAGenericObject.
  GenericObject := IUnknown(Value) as _UAGenericObject;

  // The actual data is in the GenericData property of the UAGenericObject.
  //
```

```
  // If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
  // specifier. In the debugger, you can view the same by displaying the private DebugView property.
  WriteLn;
  WriteLn(GenericObject.GenericData.ToString_2['V', nil]);

  // For processing the internals of the data, refer to examples for GenericData and DataType classes.

  // Example output (truncated):
  //
  //(ScalarValueDataType) structured
  //
  //(ScalarValueDataType) structured
  //  [BooleanValue] (Boolean) primitive; True {System.Boolean}
  //  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
  //  [ByteValue] (Byte) primitive; 153 {System.Byte}
  //  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
  //  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
  //  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
  //  [ExpandedNodeIdValue] (ExpandedNodeId) structured
  //    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
  //    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
  //    [NodeIdType] (NodeIdType) enumeration; 3 (String)
  //    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
  //    [String] (StringNodeId) structured
  //      [Identifier] (CharArray) primitive; "????" {System.String}
  //      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
  //  [FloatValue] (Float) primitive; 78.37176 {System.Single}
  //  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
  //  [Int16Value] (Int16) primitive; 2793 {System.Int16}
  //  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
  //  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
  //  [Integer] (Variant) structured
  //    [ArrayDimensionsSpecified] sequence[1]
  //      [0] (Bit) primitive; False {System.Boolean}
  //    [ArrayLengthSpecified] sequence[1]
  //      [0] (Bit) primitive; False {System.Boolean}
  //    [Int64] sequence[1]
  //      [0] (Int64) primitive; 0 {System.Int64}
  //    [VariantType] sequence[6]
  //      [0] (Bit) primitive; False {System.Boolean}
  //      [1] (Bit) primitive; False {System.Boolean}
  //      [2] (Bit) primitive; False {System.Boolean}
  //      [3] (Bit) primitive; True {System.Boolean}
  //      [4] (Bit) primitive; False {System.Boolean}
  //      [5] (Bit) primitive; False {System.Boolean}
  //  [LocalizedTextValue] (LocalizedText) structured
  //    [Locale] (CharArray) primitive; "ko" {System.String}
  //    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
  //    [Reserved1] sequence[6]
  //      [0] (Bit) primitive; False {System.Boolean}
  //      [1] (Bit) primitive; False {System.Boolean}
  //      [2] (Bit) primitive; False {System.Boolean}
  //      [3] (Bit) primitive; False {System.Boolean}
  //      [4] (Bit) primitive; False {System.Boolean}
  //      [5] (Bit) primitive; False {System.Boolean}
  //    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
  //    [TextSpecified] (Bit) primitive; True {System.Boolean}
  //  [NodeIdValue] (NodeId) structured

end;
```

PHP

```
// Shows how to read complex data with OPC UA Complex Data plug-in.

// Define which server and node we will work with.
$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
$NodeDescriptor = "nsu=http://test.org/UA/Data/;i=10239";  // [ObjectsFolder]/Data.Static.Scalar.StructureValue

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Read a node which returns complex data. This is done in the same way as regular reads - just the data
// returned is different.

try
{
    $Value = $Client->ReadValue($EndpointDescriptor, $NodeDescriptor);
}
catch (com_exception $e)
{
```

```
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display basic information about what we have read.
printf("Value: %s\n", $Value);

// We know that this node returns complex data, so it is a UAGenericObject.
$GenericObject = $Value;

// The actual data is in the GenericData property of the UAGenericObject.
//
// If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
// specifier. In the debugger, you can view the same by displaying the private DebugView property.
printf("\n");
printf("%s\n", $GenericObject->GenericData->ToString_3("V"));

// For processing the internals of the data, refer to examples for GenericData and DataType classes.

// Example output (truncated):
//
//(ScalarValueDataType) structured
//
//(ScalarValueDataType) structured
//  [BooleanValue] (Boolean) primitive; True {System.Boolean}
//  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
//  [ByteValue] (Byte) primitive; 153 {System.Byte}
//  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
//  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
//  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
//  [ExpandedNodeIdValue] (ExpandedNodeId) structured
//    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
//    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
//    [NodeIdType] (NodeIdType) enumeration; 3 (String)
//    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
//    [String] (StringNodeId) structured
//      [Identifier] (CharArray) primitive; "????" {System.String}
//      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
//  [FloatValue] (Float) primitive; 78.37176 {System.Single}
//  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
//  [Int16Value] (Int16) primitive; 2793 {System.Int16}
//  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
//  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
//  [Integer] (Variant) structured
//    [ArrayDimensionsSpecified] sequence[1]
//      [0] (Bit) primitive; False {System.Boolean}
//    [ArrayLengthSpecified] sequence[1]
//      [0] (Bit) primitive; False {System.Boolean}
//    [Int64] sequence[1]
//      [0] (Int64) primitive; 0 {System.Int64}
//    [VariantType] sequence[6]
//      [0] (Bit) primitive; False {System.Boolean}
//      [1] (Bit) primitive; False {System.Boolean}
//      [2] (Bit) primitive; False {System.Boolean}
//      [3] (Bit) primitive; True {System.Boolean}
//      [4] (Bit) primitive; False {System.Boolean}
//      [5] (Bit) primitive; False {System.Boolean}
//  [LocalizedTextValue] (LocalizedText) structured
//    [Locale] (CharArray) primitive; "ko" {System.String}
//    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
//    [Reserved1] sequence[6]
//      [0] (Bit) primitive; False {System.Boolean}
//      [1] (Bit) primitive; False {System.Boolean}
//      [2] (Bit) primitive; False {System.Boolean}
//      [3] (Bit) primitive; False {System.Boolean}
//      [4] (Bit) primitive; False {System.Boolean}
//      [5] (Bit) primitive; False {System.Boolean}
//    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
//    [TextSpecified] (Bit) primitive; True {System.Boolean}
//  [NodeIdValue] (NodeId) structured
```

VB.NET

```
' Shows how to read complex data with OPC UA Complex Data plug-in.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._EasyUAClient

    Friend Class ReadValue
```

```
Public Shared Sub Main1()

    ' Define which server we will work with.
    Dim endpointDescriptor As UAEndpointDescriptor =
            "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
    ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
    ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

    ' Define which node we will work with.
    Dim nodeDescriptor As UANodeDescriptor =
        "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue

    ' Instantiate the client object.
    Dim client = New EasyUAClient

    ' Read a node which returns complex data. This is done in the same way as regular reads - just the data
    ' returned is different.
    Dim value As Object
    Try
        value = client.ReadValue(endpointDescriptor, nodeDescriptor)
    Catch uaException As UAException
        Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
        Exit Sub
    End Try

    ' Display basic information about what we have read.
    Console.WriteLine(value)


    ' We know that this node returns complex data, so we can type cast to UAGenericObject.
    Dim genericObject = CType(value, UAGenericObject)

    ' The actual data is in the GenericData property of the UAGenericObject.
    '
    ' If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
    ' specifier. In the debugger, you can view the same by displaying the private DebugView property.
    Console.WriteLine()
    Console.WriteLine("{0:V}", genericObject.GenericData)

    ' For processing the internals of the data, refer to examples for GenericData and DataType classes.


    ' Example output (truncated):
    '
    '(ScalarValueDataType) structured
    '
    '(ScalarValueDataType) structured
    '   [BooleanValue] (Boolean) primitive; True {System.Boolean}
    '   [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
    '   [ByteValue] (Byte) primitive; 153 {System.Byte}
    '   [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
    '   [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
    '   [EnumerationValue] (Int32) primitive; 0 {System.Int32}
    '   [ExpandedNodeIdValue] (ExpandedNodeId) structured
    '     [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
    '     [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
    '     [NodeIdType] (NodeIdType) enumeration; 3 (String)
    '     [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
    '     [String] (StringNodeId) structured
    '       [Identifier] (CharArray) primitive; "????" {System.String}
    '       [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
    '   [FloatValue] (Float) primitive; 78.37176 {System.Single}
    '   [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
    '   [Int16Value] (Int16) primitive; 2793 {System.Int16}
    '   [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
    '   [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
    '   [Integer] (Variant) structured
    '     [ArrayDimensionsSpecified] sequence[1]
    '       [0] (Bit) primitive; False {System.Boolean}
    '     [ArrayLengthSpecified] sequence[1]
    '       [0] (Bit) primitive; False {System.Boolean}
    '     [Int64] sequence[1]
    '       [0] (Int64) primitive; 0 {System.Int64}
    '     [VariantType] sequence[6]
    '       [0] (Bit) primitive; False {System.Boolean}
    '       [1] (Bit) primitive; False {System.Boolean}
    '       [2] (Bit) primitive; False {System.Boolean}
    '       [3] (Bit) primitive; True {System.Boolean}
    '       [4] (Bit) primitive; False {System.Boolean}
    '       [5] (Bit) primitive; False {System.Boolean}
    '   [LocalizedTextValue] (LocalizedText) structured
    '     [Locale] (CharArray) primitive; "ko" {System.String}
```

```
            '    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
            '    [Reserved1] sequence[6]
            '      [0] (Bit) primitive; False {System.Boolean}
            '      [1] (Bit) primitive; False {System.Boolean}
            '      [2] (Bit) primitive; False {System.Boolean}
            '      [3] (Bit) primitive; False {System.Boolean}
            '      [4] (Bit) primitive; False {System.Boolean}
            '      [5] (Bit) primitive; False {System.Boolean}
            '    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
            '    [TextSpecified] (Bit) primitive; True {System.Boolean}
            '  [NodeIdValue] (NodeId) structured
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to read and display data of an attribute (value, timestamps, and status code).

Option Explicit

' Define which server and node we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
Dim nodeDescriptor: nodeDescriptor = _
    "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Read a node which returns complex data. This is done in the same way as regular reads - just the data
' returned is different.
On Error Resume Next
Dim Value: Set Value = Client.ReadValue(endpointDescriptor, nodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display basic information about what we have read.
WScript.Echo Value


' We know that this node returns complex data, so it is a UAGenericObject.
Dim GenericObject: Set GenericObject = Value

' The actual data is in the GenericData property of the UAGenericObject.
'
' If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
' specifier. In the debugger, you can view the same by displaying the private DebugView property.
WScript.Echo
WScript.Echo GenericObject.GenericData.ToString_2("V", Nothing)

' For processing the internals of the data, refer to examples for GenericData and DataType classes.


' Example output (truncated):
'
'(ScalarValueDataType) structured
'
'(ScalarValueDataType) structured
'  [BooleanValue] (Boolean) primitive; True {System.Boolean}
'  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
'  [ByteValue] (Byte) primitive; 153 {System.Byte}
'  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
'  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
'  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
'  [ExpandedNodeIdValue] (ExpandedNodeId) structured
'    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
'    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
'    [NodeIdType] (NodeIdType) enumeration; 3 (String)
'    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
'    [String] (StringNodeId) structured
'      [Identifier] (CharArray) primitive; "????" {System.String}
'      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
'  [FloatValue] (Float) primitive; 78.37176 {System.Single}
'  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
'  [Int16Value] (Int16) primitive; 2793 {System.Int16}
'  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
'  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
```

```
'  [Integer] (Variant) structured
'    [ArrayDimensionsSpecified] sequence[1]
'      [0] (Bit) primitive; False {System.Boolean}
'    [ArrayLengthSpecified] sequence[1]
'      [0] (Bit) primitive; False {System.Boolean}
'    [Int64] sequence[1]
'      [0] (Int64) primitive; 0 {System.Int64}
'    [VariantType] sequence[6]
'      [0] (Bit) primitive; False {System.Boolean}
'      [1] (Bit) primitive; False {System.Boolean}
'      [2] (Bit) primitive; False {System.Boolean}
'      [3] (Bit) primitive; True {System.Boolean}
'      [4] (Bit) primitive; False {System.Boolean}
'      [5] (Bit) primitive; False {System.Boolean}
'  [LocalizedTextValue] (LocalizedText) structured
'    [Locale] (CharArray) primitive; "ko" {System.String}
'    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
'    [Reserved1] sequence[6]
'      [0] (Bit) primitive; False {System.Boolean}
'      [1] (Bit) primitive; False {System.Boolean}
'      [2] (Bit) primitive; False {System.Boolean}
'      [3] (Bit) primitive; False {System.Boolean}
'      [4] (Bit) primitive; False {System.Boolean}
'      [5] (Bit) primitive; False {System.Boolean}
'    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
'    [TextSpecified] (Bit) primitive; True {System.Boolean}
'  [NodeIdValue] (NodeId) structured
```

7.2.1.1.2 OPC UA Complex Data subscribing

To subscribe to complex data information, call one of the **Subscribe** methods on the EasyUAClient Class as usual. When complex data is returned by the client, a DataChangeNotification Event or EventNotification Event will be raised (or a corresponding callback will be made). The value in the event arguments will be of type UAGenericObject Class, and you will process its contents.

How precisely you process the received generic data depends on the application. In general, there can be two usage cases:

- Your application knows the type of the data to expect. In this case, you can hard-code this knowledge into the application. For example, if you know that the received data is a structure that has a field called "Temperature". You would type-cast the received data to StructuredData, and from the FieldData collection, you will get an element indexed by a string "Temperature". This can then be an instance of PrimitiveData Class, containing the temperature value in its Value Property. For more information, see **Working with generic data (Section 7.2.1.2.1)**.
- Your application does not know the type of the data to expect. This requires more coding, but is also perfectly doable. You will be accessing the generic data information similarly to the above case, but in a more generic way. The GenericData Class gives you a DataType Property, and the DataType object contained in this property gives you the metadata needed. For more information, see **Working with data types (Section 7.2.1.2.2)**.

The example below shows how to subscribe to OPC UA complex data and display the incoming values.

C#

```
// Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._EasyUAClient
{
```

```
    class SubscribeDataChange
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10867"; //
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

            // Instantiate the client object and hook the event handler.
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            // Subscribe to a node which returns complex data. This is done in the same
way as regular subscribes - just
            // the data returned is different.
            Console.WriteLine("Subscribing...");
            client.SubscribeDataChange(endpointDescriptor, nodeDescriptor,
samplingInterval:1000);

            Console.WriteLine("Processing data change events for 20 seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            // Unhook the event handler.
            client.DataChangeNotification -= client_DataChangeNotification;


            // Example output:
            //
            //Subscribing...
            //Processing data change events for 20 seconds...
            //(ScalarValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ScalarValueDataType) structured
            //(ScalarValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ScalarValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
```

```
            //(ScalarValueDataType) structured
            //(ScalarValueDataType) structured
            //(ScalarValueDataType) structured
            //Unsubscribing...
            //Waiting for 5 seconds...
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("Value: {0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);

            // For processing the internals of the data, refer to examples for
GenericData and DataType classes.
        }
    }
}
```

Object Pascal

```
// Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

type
  TClientEventHandlers33 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers33.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display value
  if eventArgs.Succeeded then
      WriteLn('Value: ', eventArgs.AttributeData.Value.ToString)
  else
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);

  // For processing the internals of the data, refer to examples for GenericData and
DataType classes.
end;

class procedure SubscribeDataChange.Main;
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers33;
  EndpointDescriptor: string;
  NodeDescriptor: string;
begin
  // Define which server and node we will work with.
```

```
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10867';  //
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers33.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  // Subscribe to a node which returns complex data. This is done in the same way as
regular subscribes - just
  // the data returned is different.
  WriteLn('Subscribing...');
  Client.SubscribeDataChange(EndpointDescriptor, NodeDescriptor, 1000);

  WriteLn('Processing data change events for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);

  // Example output:
  //
  //Subscribing...
  //Processing data change events for 20 seconds...
  //(ScalarValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ScalarValueDataType) structured
  //(ScalarValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ScalarValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ScalarValueDataType) structured
  //(ScalarValueDataType) structured
  //(ScalarValueDataType) structured
  //Unsubscribing...
  //Waiting for 5 seconds...
end;
```

PHP

```
// Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display value
        if ($E->Succeeded)
        printf("Value: %s\n", $E->AttributeData->Value);
        else
        printf("*** Failure : %s\n", $E->ErrorMessageBrief);
        // For processing the internals of the data, refer to examples for GenericData
and DataType classes.
    }
}

// Define which server and node we will work with.
$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
$NodeDescriptor = "nsu=http://test.org/UA/Data/;i=10867";  //
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

// Subscribe to a node which returns complex data. This is done in the same way as
regular subscribes - just
// the data returned is different.
printf("Subscribing...\n");
$Client->SubscribeDataChange($EndpointDescriptor, $NodeDescriptor, 1000);

printf("Processing data change events for 20 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 20);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

VB.NET

```
' Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._EasyUAClient

    Friend Class SubscribeDataChange
```

```
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10867"  '
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

            ' Instantiate the client object and hook the event handler.
            Dim client = New EasyUAClient
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            ' Subscribe to a node which returns complex data. This is done in the same
way as regular subscribes - just
            ' the data returned is different.
            Console.WriteLine("Subscribing...")
            client.SubscribeDataChange(endpointDescriptor, nodeDescriptor,
samplingInterval:=1000)

            Console.WriteLine("Processing data change events for 20 seconds...")
            Threading.Thread.Sleep((20 * 1000))

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep((5 * 1000))

            ' Unhook the event handler.
            RemoveHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification


            ' Example output:
            '
            'Subscribing...
            'Processing data change events for 20 seconds...
            '(ScalarValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ScalarValueDataType) structured
            '(ScalarValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ScalarValueDataType) structured
```

```
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ScalarValueDataType) structured
            '(ScalarValueDataType) structured
            '(ScalarValueDataType) structured
            'Unsubscribing...
            'Waiting for 5 seconds...
        End Sub

        Private Shared Sub client_DataChangeNotification(sender As Object, e As
EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If

            ' For processing the internals of the data, refer to examples for
GenericData and DataType classes.
        End Sub
    End Class
End Namespace
```

Complex data is also supported and can also be returned in notifications for OPC UA Alarms&Conditions, i.e. in event arguments of type EasyUAEventNotificationEventArgs, used in the EventNotification Event .

7.2.1.1.3 OPC UA Complex Data writing

To write a complex data information, prepare a value of the UAGenericObject Class, and call one of the **Write** methods on the EasyUAClient Class as usual. In the UAGenericObject:

- Fill in the GenericData Property by the generic data you want to write.
- Optionally, fill in the DataTypeId Property by the data type ID specifying the type of data to be written. In some cases, this information is necessary for the Write to succeed. Refer to **Role of the data type ID (Section 7.2.1.1.4)** for related discussion.

For discussion about creating the generic data, refer to **Creating generic data (Section 7.2.1.2.1.2)**. If the GenericData object you have created has a non-null DataType Property, the generic data will be first validated against this associated data type. For details on the validation process, refer to **Generic data validation (Section 7.2.1.2.1.4)**. The validation is optional, because the presence of the data type in the GenericData object is also optional. If no validation takes place, same errors may be discovered later during the encoding process. The validation has the advantage that it is done upfront, without a need to contact the server at all. It is therefore recommended to use it, if the data type is available somehow (see also "Resolving the data type" in **Navigation in the OPC UA data model (Section 7.2.1.4.3)**).

In some cases, you do not have to provide all data that are part of the data type. For example, it may be possible to omit the value for a field that determines a length of sequence in the same structure. For details, see **Generic data auto-complete (Section 7.2.1.3.1)**.

The example below first reads a structured value from the server, and then manipulates (changes) its elements. The modified value is then written to the server. In this case, the data type ID (and also the data type) can be taken over from

what has been provided by the Read.

C#

```
// Shows how to write complex data with OPC UA Complex Data plug-in.

using System;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._EasyUAClient
{
    class WriteValue
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node which returns complex data.
            // We know that this node returns complex data, so we can type cast to
UAGenericObject.
            Console.WriteLine("Reading...");
            UAGenericObject genericObject;
            try
            {
                genericObject = (UAGenericObject)client.ReadValue(endpointDescriptor,
nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }


            // Modify the data read.
            // This node returns one of the two data types, randomly (this is not
common, usually the type is fixed). The
            // data types are sub-types of one common type which the data type of the
node. We therefore use the data type
            // ID in the returned UAGenericObject to detect which data type has been
returned.
```

```
            // For processing the internals of the data, refer to examples for
GenericData and DataType classes.
            // We know how the data is structured, and have hard-coded a logic that
modifies certain values inside. It is
            // also possible to discover the structure of the data type in the program,
and write generic clients that can
            // cope with any kind of complex data.
            //
            // Note that the code below is not fully robust - it will throw an
exception if the data is not as expected.
            Console.WriteLine("Modifying...");
            Console.WriteLine(genericObject.DataTypeId);
            if
(genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9440"))
// ScalarValueDataType
            {
                // Negate the byte in the "ByteValue" field.
                var structuredData = (StructuredData)genericObject.GenericData;
                var byteValue = (PrimitiveData)structuredData.FieldData["ByteValue"];
                byteValue.Value = (Byte)~((Byte)byteValue.Value);
                Console.WriteLine(byteValue.Value);
            }
            else if
(genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9669"))
// ArrayValueDataType
            {
                // Negate bytes at indexes 0 and 1 of the array in the "ByteValue"
field.
                var structuredData = (StructuredData)genericObject.GenericData;
                var byteValue = (SequenceData)structuredData.FieldData["ByteValue"];
                var element0 = (PrimitiveData)byteValue.Elements[0];
                var element1 = (PrimitiveData)byteValue.Elements[1];
                element0.Value = (Byte)~((Byte)element0.Value);
                element1.Value = (Byte)~((Byte)element1.Value);
                Console.WriteLine(element0.Value);
                Console.WriteLine(element1.Value);
            }


            // Write the modified complex data back to the node.
            // The data type ID in the UAGenericObject is borrowed without change from
what we have read, so that the server
            // knows which data type we are writing. The data type ID not necessary if
writing precisely the same data type
            // as the node has (not a subtype).
            Console.WriteLine("Writing...");
            try
            {
                client.WriteValue(endpointDescriptor, nodeDescriptor, genericObject);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
```

```
    }
}
```

Object Pascal

```
// Shows how to write complex data with OPC UA Complex Data plug-in.

class procedure WriteValue.Main;
var
  ArrayValueDataType: _UANodeDescriptor;
  ByteValue: _PrimitiveData;
  ByteValue2: _SequenceData;
  Client: _EasyUAClient;
  Element0, Element1: _PrimitiveData;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
  ScalarValueDataType: _UANodeDescriptor;
  StructuredData: _StructuredData;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
   'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node which returns complex data.
  // We know that this node returns complex data, so we can type cast to
UAGenericObject.
  WriteLn('Reading...');

  try
    GenericObject := _UAGenericObject(IUnknown(Client.ReadValue(EndpointDescriptor,
NodeDescriptor)));
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Modify the data read.
  // This node returns one of the two data types, randomly (this is not common, usually
the type is fixed). The
  // data types are sub-types of one common type which the data type of the node. We
therefore use the data type
  // ID in the returned UAGenericObject to detect which data type has been returned.

  // For processing the internals of the data, refer to examples for GenericData and
DataType classes.
  // We know how the data is structured, and have hard-coded a logic that modifies
```

```
certain values inside. It is
  // also possible to discover the structure of the data type in the program, and write
generic clients that can
  // cope with any kind of complex data.
  //
  // Note that the code below is not fully robust - it will throw an exception if the
data is not as expected.

  WriteLn('Modifying...');
  WriteLn(GenericObject.DataTypeId.ToString);
  ScalarValueDataType := CoUANodeDescriptor.Create;
  ScalarValueDataType.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=9440'; //
ScalarValueDataType
  if GenericObject.DataTypeId.NodeDescriptor.Match(ScalarValueDataType) then
  begin
    // Negate the byte in the "ByteValue" field.
    StructuredData := IUnknown(GenericObject.GenericData) as _StructuredData;
    ByteValue := IUnknown(StructuredData.FieldData['ByteValue']) as _PrimitiveData;
    ByteValue.Value := Byte(not (Byte(byteValue.Value)));
    WriteLn(ByteValue.Value);
  end
  else
  begin
    ArrayValueDataType := CoUANodeDescriptor.Create;
    ArrayValueDataType.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=9669'; //
ArrayValueDataType
    if GenericObject.DataTypeId.Nodedescriptor.Match(ArrayValueDataType) then
    begin
      // Negate bytes at indexes 0 and 1 of the array in the "ByteValue" field.
      StructuredData := IUnknown(GenericObject.GenericData) as _StructuredData;
      ByteValue2 := IUnknown(StructuredData.FieldData['ByteValue']) as _SequenceData;
      Element0 := IUnknown(ByteValue2.Elements[0]) as _PrimitiveData;
      Element1 := IUnknown(ByteValue2.Elements[1]) as _PrimitiveData;
      Element0.Value := Byte(not (Byte(element0.Value)));
      Element1.Value := Byte(not (Byte(element1.Value)));
      WriteLn(Element0.Value);
      WriteLn(Element1.Value);
    end;
  end;

  // Write the modified complex data back to the node.
  // The data type ID in the UAGenericObject is borrowed without change from what we
have read, so that the server
  // knows which data type we are writing. The data type ID not necessary if writing
precisely the same data type
  // as the node has (not a subtype).
  WriteLn('Writing...');
  try
    Client.WriteValue(EndpointDescriptor, NodeDescriptor, GenericObject);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
```

```
end;
```

PHP

```
// Shows how to write complex data with OPC UA Complex Data plug-in.

// Define which server and node we will work with.
$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
$NodeDescriptor = "nsu=http://test.org/UA/Data/;i=10239";  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Read a node which returns complex data.
// We know that this node returns complex data, so we can type cast to UAGenericObject.
printf("Reading...\n");

try
{
    $GenericObject = $Client->ReadValue($EndpointDescriptor, $NodeDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Modify the data read.
// This node returns one of the two data types, randomly (this is not common, usually
the type is fixed). The
// data types are sub-types of one common type which the data type of the node. We
therefore use the data type
// ID in the returned UAGenericObject to detect which data type has been returned.

// For processing the internals of the data, refer to examples for GenericData and
DataType classes.
// We know how the data is structured, and have hard-coded a logic that modifies
certain values inside. It is
// also possible to discover the structure of the data type in the program, and write
generic clients that can
// cope with any kind of complex data.
//
// Note that the code below is not fully robust - it will throw an exception if the
data is not as expected.

printf("Modifying...\n");
printf("%s\n", $GenericObject->DataTypeId);
$ScalarValueDataType = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
$ScalarValueDataType->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=9440"; //
ScalarValueDataType
if ($GenericObject->DataTypeId->NodeDescriptor->Match($ScalarValueDataType)) {
    // Negate the byte in the "ByteValue" field.
    $StructuredData = $GenericObject->GenericData->AsStructuredData();
    $ByteValue = $StructuredData->FieldData["ByteValue"]->AsPrimitiveData();
```

```
    $ByteValue->Value = ~($ByteValue->Value) & 255;
    printf("%s\n", $ByteValue->Value);
}
else {
    $ArrayValueDataType = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
    $ArrayValueDataType->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=9669";
// ArrayValueDataType
    if ($GenericObject->DataTypeId->Nodedescriptor->Match($ArrayValueDataType)) {
        // Negate bytes at indexes 0 and 1 of the array in the "ByteValue" field.
        $StructuredData = $GenericObject->GenericData->AsStructuredData();
        $ByteValue2 = $StructuredData->FieldData["ByteValue"]->AsSequenceData();
        $Element0 = $ByteValue2->Elements[0]->AsPrimitiveData();
        $Element1 = $ByteValue2->Elements[1]->AsPrimitiveData();
        $Element0->Value = ~($Element0->Value) & 255;
        $Element1->Value = ~($Element1->Value) & 255;
        printf("%s\n", $Element0->Value);
        printf("%s\n", $Element1->Value);
    }
}

// Write the modified complex data back to the node.
// The data type ID in the UAGenericObject is borrowed without change from what we have
read, so that the server
// knows which data type we are writing. The data type ID not necessary if writing
precisely the same data type
// as the node has (not a subtype).
printf("Writing...\n");
try
{
    $Client->WriteValue($EndpointDescriptor, $NodeDescriptor, $GenericObject);
}
catch (com_exception $e)
{
    printf("Failure: %s\n", $e->getMessage());
    Exit();
}
```

VB.NET

```
' Shows how to write complex data with OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._EasyUAClient

    Friend Class WriteValue

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
```

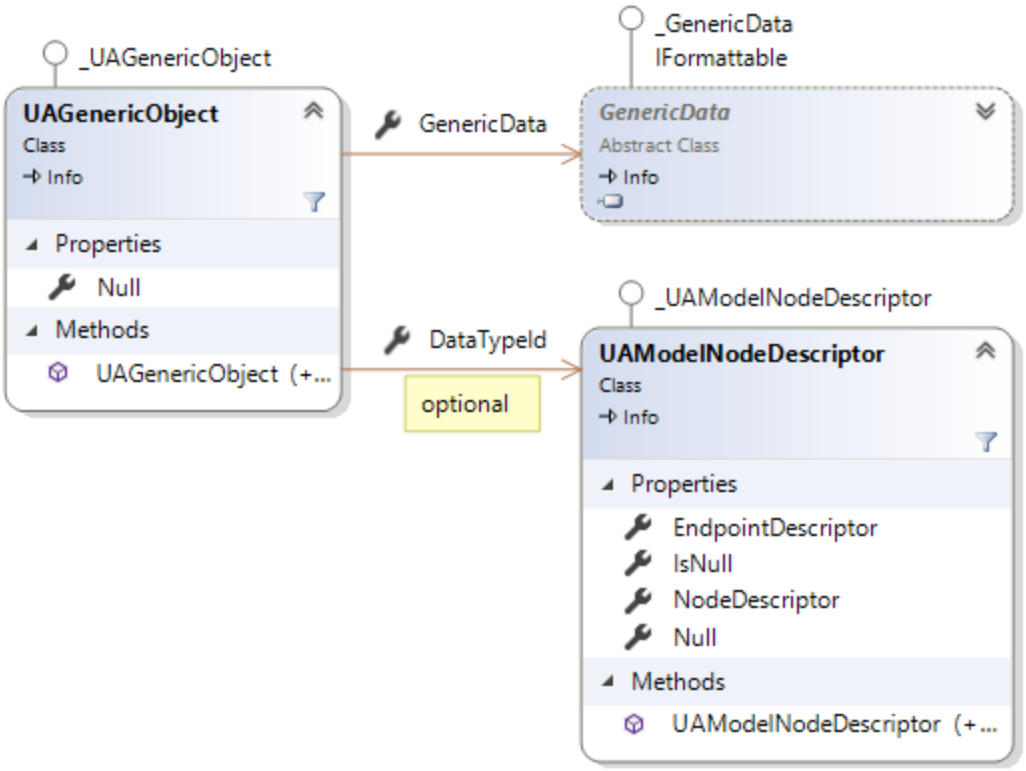
```
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node which returns complex data.
            ' We know that this node returns complex data, so we can type cast to
UAGenericObject.
            Console.WriteLine("Reading...")
            Dim genericObject As UAGenericObject
            Try
                genericObject = CType(client.ReadValue(endpointDescriptor,
nodeDescriptor), UAGenericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try


            ' Modify the data read.
            ' This node returns one of the two data types, randomly (this is not
common, usually the type is fixed). The
            ' data types are sub-types of one common type which the data type of the
node. We therefore use the data type
            ' ID in the returned UAGenericObject to detect which data type has been
returned.
            ' For processing the internals of the data, refer to examples for
GenericData and DataType classes.
            ' We know how the data is structured, and have hard-coded a logic that
modifies certain values inside. It is
            ' also possible to discover the structure of the data type in the program,
and write generic clients that can
            ' cope with any kind of complex data.
            '
            ' Note that the code below is not fully robust - it will throw an exception
if the data is not as expected.
            Console.WriteLine("Modifying...")
            Console.WriteLine(genericObject.DataTypeId)
            If
genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9440")
Then    ' ScalarValueDataType
                ' Negate the byte in the "ByteValue" field.
                Dim structuredData = CType(genericObject.GenericData, StructuredData)
                Dim byteValue = CType(structuredData.FieldData("ByteValue"),
PrimitiveData)
                byteValue.Value = CType(Not CType(byteValue.Value, Byte), Byte)
                Console.WriteLine(byteValue.Value)
            ElseIf
genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9669")
Then    ' ArrayValueDataType
```

```
                ' Negate bytes at indexes 0 and 1 of the array in the "ByteValue"
field.
                Dim structuredData = CType(genericObject.GenericData, StructuredData)
                Dim byteValue = CType(structuredData.FieldData("ByteValue"),
SequenceData)
                Dim element0 = CType(byteValue.Elements(0), PrimitiveData)
                Dim element1 = CType(byteValue.Elements(1), PrimitiveData)
                element0.Value = CType(Not CType(element0.Value, Byte), Byte)
                element1.Value = CType(Not CType(element1.Value, Byte), Byte)
                Console.WriteLine(element0.Value)
                Console.WriteLine(element1.Value)
            End If


            ' Write the modified complex data back to the node.
            ' The data type ID in the UAGenericObject is borrowed without change from
what we have read, so that the server
            ' knows which data type we are writing. The data type ID not necessary if
writing precisely the same data type
            ' as the node has (not a subtype).
            Console.WriteLine("Writing...")
            Try
                client.WriteValue(endpointDescriptor, nodeDescriptor, genericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

7.2.1.1.4 Role of the data type ID

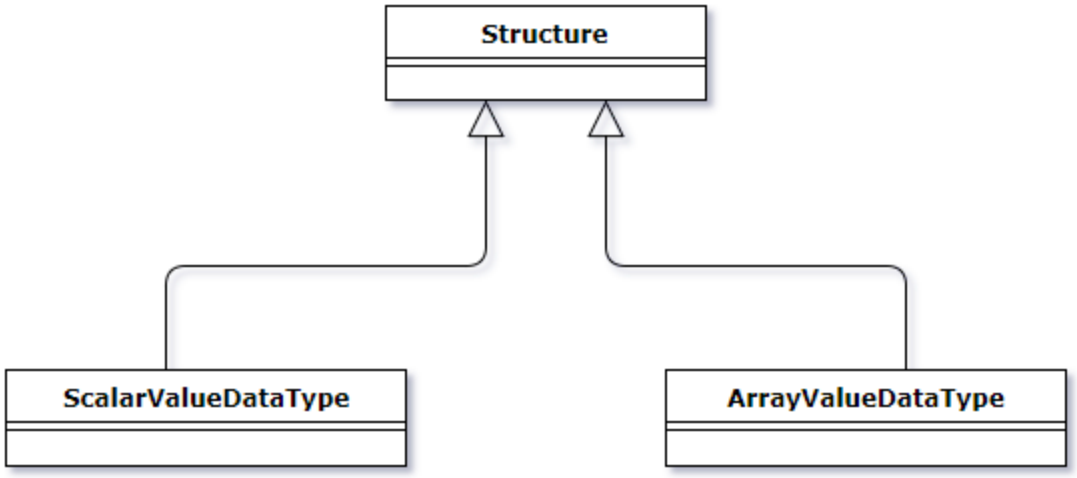
The data type ID (in DataTypeId Property) is an optional information in the UAGenericObject Class (it equals to UAModelNodeDescriptor.Null when "not present"). It is basically a node ID of data type that this UAGenericObject represents.

Show UAGenericObject class diagram



The data type ID of certain node can be determined from the information model of the server (by reading the DataType attribute of that node). That is, however, the **declared** data type for that node. As the OPC UA information model for data types follows the classic paradigms of OOP (Object Oriented Programming), the **actual** data type at runtime can also be any direct or indirect sub-type of the declared data type (in fact, when the declared data type is abstract, the actual data type must be some sub-type, different from the declared type - this is again the same as in classic OOP, because instances of abstract types cannot be created).

In many cases, the node in the OPC UA server you are dealing with has a data type that has no sub-types. In such case, you are lucky, because you always know upfront the precise data type that is to be used for the node's value. But in some cases, the situation is more complicated, and the problem domain calls for sub-types, a hierarchical structrue of data types. The picture below is a class diagram that shows example relations between three data types. The ScalarValueDataType and ArrayValueDataType inherit from a base data type, Structure.



When a node in such server then specifies its data type as Structure, there are three possible data types that the value can actually have: Structure, ScalarValueDataType, or ArrayValueDataType.

When the generic object is obtained from the server (such as by reads and subscriptions), the component always fills in the with a data type ID that is not UAModelNodeDescriptor.Null. When the generic object is to be transferred to the server (such as by writing), the component will use the data type ID provided, if it is not equal to UAModelNodeDescriptor.Null. Otherwise, the component will try to determine the data type (ID) itself, which may not work in case the actual data to be written are a sub-type of the node's data type (and which is always the case if the node's data type is abstract).

What do you need to take out of this discussion? That's actually easy:

- If you are just obtaining information from the server (reads, subscriptions), you need not to worry about the data type ID.
- If you are writing to the server, and the target node only allows value of single data type (no sub-types), you also do not need to worry about the data type ID.
- If you are writing to the server and the target node allows sub-types, you **have to** provide proper data type ID with the generic object, by filling in the DataTypeId Property.

How you obtain the data type ID depends on the application. For example

- You may know which server you are connecting to, and you will hard-code the necessary data type IDs into the program.
- You have previously obtained the generic data from the server, together with the its associated data type ID. In this case, you can simply pass that data type ID into the Write call.
- You can read the DataType attribute of the given node, either by regular Read methods on EasyUAClient Class, or using a handy extension method ReadDataType (or ReadMultipleDataTypes). As discussed above, this will only work if the actual data type you want is not a sub-type of the node's data type.
- You can navigate through the information model in the server, obtain available sub-types, and then have your application select from them.

The example below determines sub-types of a a given data type:

C#

```
// This example shows how to retrieve all sub-types of a given data type, recursively.

using System;
using OpcLabs.BaseLib.OperationModel.Generic;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.Extensions;

namespace UADocExamples._EasyUAClient
{
    class RetrieveAllDataTypes
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Retrieve all sub-types of the 'Structure' data type.
            ValueResult<UANodeDescriptor[]> result = client.RetrieveAllDataTypes(
                endpointDescriptor, UADataTypeIds.Structure);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead
if you want exception be thrown.
            if (!result.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief);
                return;
            }

            // Display results. Note that all node descriptors have node IDs in them;
but the default display format shows
            // the browse paths, which are more readable, when they are available.
            foreach (UANodeDescriptor nodeDescriptor in result.Value)
                Console.WriteLine(nodeDescriptor);

            // Example output:
            //
            //NodeId="Structure"
            //BrowsePath="[Structure]/Argument"
            //BrowsePath="[Structure]/StatusResult"
            //BrowsePath="[Structure]/UserTokenPolicy"
            //BrowsePath="[Structure]/ApplicationDescription"
            //BrowsePath="[Structure]/EndpointDescription"
            //BrowsePath="[Structure]/UserIdentityToken"
            //BrowsePath="[Structure]/EndpointConfiguration"
            //BrowsePath="[Structure]/SupportedProfile"
            //BrowsePath="[Structure]/BuildInfo"
            //BrowsePath="[Structure]/SoftwareCertificate"
            //BrowsePath="[Structure]/SignedSoftwareCertificate"
```

```
            //BrowsePath="[Structure]/AddNodesItem"
            //BrowsePath="[Structure]/AddReferencesItem"
            //BrowsePath="[Structure]/DeleteNodesItem"
            //BrowsePath="[Structure]/DeleteReferencesItem"
            //BrowsePath="[Structure]/ScalarTestType"
            //BrowsePath="[Structure]/ArrayTestType"
            //BrowsePath="[Structure]/CompositeTestType"
            //BrowsePath="[Structure]/RegisteredServer"
            //BrowsePath="[Structure]/ContentFilterElement"
            //BrowsePath="[Structure]/ContentFilter"
            //BrowsePath="[Structure]/FilterOperand"
            //BrowsePath="[Structure]/HistoryEvent"
            //BrowsePath="[Structure]/MonitoringFilter"
            //BrowsePath="[Structure]/RedundantServerDataType"
            //BrowsePath="[Structure]/SamplingIntervalDiagnosticsDataType"
            //BrowsePath="[Structure]/ServerDiagnosticsSummaryDataType"
            //BrowsePath="[Structure]/ServerStatusDataType"
            //BrowsePath="[Structure]/SessionDiagnosticsDataType"
            //BrowsePath="[Structure]/SessionSecurityDiagnosticsDataType"
            //BrowsePath="[Structure]/ServiceCounterDataType"
            //BrowsePath="[Structure]/SubscriptionDiagnosticsDataType"
            //BrowsePath="[Structure]/ModelChangeStructureDataType"
            //BrowsePath="[Structure]/Range"
            //BrowsePath="[Structure]/EUInformation"
            //BrowsePath="[Structure]/Annotation"
            //BrowsePath="[Structure]/ProgramDiagnosticDataType"
            //BrowsePath="[Structure]/SemanticChangeStructureDataType"
            //BrowsePath="[Structure]/HistoryEventFieldList"
            //BrowsePath="[Structure]/AggregateConfiguration"
            //BrowsePath="[Structure]/EnumValueType"
            //BrowsePath="[Structure]/TimeZoneDataType"
            //BrowsePath="[Structure]/EndpointUrlListDataType"
            //BrowsePath="[Structure]/NetworkGroupDataType"
            //BrowsePath="[Structure]/AxisInformation"
            //BrowsePath="[Structure]/XVType"
            //BrowsePath="[Structure]/ComplexNumberType"
            //BrowsePath="[Structure]/DoubleComplexNumberType"
            //BrowsePath="[Structure]/ScalarValueDataType"
            //BrowsePath="[Structure]/ArrayValueDataType"
            //BrowsePath="[Structure]/UserScalarValueDataType"
            //BrowsePath="[Structure]/UserArrayValueDataType"
            //BrowsePath="[Structure]/UserIdentityToken/AnonymousIdentityToken"
            //BrowsePath="[Structure]/UserIdentityToken/UserNameIdentityToken"
            //BrowsePath="[Structure]/UserIdentityToken/X509IdentityToken"
            //BrowsePath="[Structure]/UserIdentityToken/IssuedIdentityToken"
            //BrowsePath="[Structure]/FilterOperand/ElementOperand"
            //BrowsePath="[Structure]/FilterOperand/LiteralOperand"
            //BrowsePath="[Structure]/FilterOperand/AttributeOperand"
            //BrowsePath="[Structure]/FilterOperand/SimpleAttributeOperand"
            //BrowsePath="[Structure]/MonitoringFilter/EventFilter"
        }
    }
}
```

VB.NET

```
' This example shows how to retrieve all sub-types of a given data type, recursively.
```

```
Imports OpcLabs.BaseLib.OperationModel.Generic
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.Extensions

Namespace UADocExamples._EasyUAClient
    Friend Class RetrieveAllDataTypes
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Retrieve all sub-types of the 'Structure' data type.
            Dim result As ValueResult(Of UANodeDescriptor()) =
client.RetrieveAllDataTypes( _
                endpointDescriptor, UADataTypeIds.Structure)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if
you want exception be thrown.
            If Not result.Succeeded Then
                Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief)
            End If

            ' Display results. Note that all node descriptors have node IDs in them;
but the default display format shows
            ' the browse paths, which are more readable, when they are available.
            For Each nodeDescriptor As UANodeDescriptor In result.Value
                Console.WriteLine(nodeDescriptor)
            Next nodeDescriptor

            ' Example output:
            '
            'NodeId="Structure"
            'BrowsePath="[Structure]/Argument"
            'BrowsePath="[Structure]/StatusResult"
            'BrowsePath="[Structure]/UserTokenPolicy"
            'BrowsePath="[Structure]/ApplicationDescription"
            'BrowsePath="[Structure]/EndpointDescription"
            'BrowsePath="[Structure]/UserIdentityToken"
            'BrowsePath="[Structure]/EndpointConfiguration"
            'BrowsePath="[Structure]/SupportedProfile"
            'BrowsePath="[Structure]/BuildInfo"
            'BrowsePath="[Structure]/SoftwareCertificate"
            'BrowsePath="[Structure]/SignedSoftwareCertificate"
            'BrowsePath="[Structure]/AddNodesItem"
            'BrowsePath="[Structure]/AddReferencesItem"
            'BrowsePath="[Structure]/DeleteNodesItem"
            'BrowsePath="[Structure]/DeleteReferencesItem"
            'BrowsePath="[Structure]/ScalarTestType"
            'BrowsePath="[Structure]/ArrayTestType"
            'BrowsePath="[Structure]/CompositeTestType"
```

```
            'BrowsePath="[Structure]/RegisteredServer"
            'BrowsePath="[Structure]/ContentFilterElement"
            'BrowsePath="[Structure]/ContentFilter"
            'BrowsePath="[Structure]/FilterOperand"
            'BrowsePath="[Structure]/HistoryEvent"
            'BrowsePath="[Structure]/MonitoringFilter"
            'BrowsePath="[Structure]/RedundantServerDataType"
            'BrowsePath="[Structure]/SamplingIntervalDiagnosticsDataType"
            'BrowsePath="[Structure]/ServerDiagnosticsSummaryDataType"
            'BrowsePath="[Structure]/ServerStatusDataType"
            'BrowsePath="[Structure]/SessionDiagnosticsDataType"
            'BrowsePath="[Structure]/SessionSecurityDiagnosticsDataType"
            'BrowsePath="[Structure]/ServiceCounterDataType"
            'BrowsePath="[Structure]/SubscriptionDiagnosticsDataType"
            'BrowsePath="[Structure]/ModelChangeStructureDataType"
            'BrowsePath="[Structure]/Range"
            'BrowsePath="[Structure]/EUInformation"
            'BrowsePath="[Structure]/Annotation"
            'BrowsePath="[Structure]/ProgramDiagnosticDataType"
            'BrowsePath="[Structure]/SemanticChangeStructureDataType"
            'BrowsePath="[Structure]/HistoryEventFieldList"
            'BrowsePath="[Structure]/AggregateConfiguration"
            'BrowsePath="[Structure]/EnumValueType"
            'BrowsePath="[Structure]/TimeZoneDataType"
            'BrowsePath="[Structure]/EndpointUrlListDataType"
            'BrowsePath="[Structure]/NetworkGroupDataType"
            'BrowsePath="[Structure]/AxisInformation"
            'BrowsePath="[Structure]/XVType"
            'BrowsePath="[Structure]/ComplexNumberType"
            'BrowsePath="[Structure]/DoubleComplexNumberType"
            'BrowsePath="[Structure]/ScalarValueDataType"
            'BrowsePath="[Structure]/ArrayValueDataType"
            'BrowsePath="[Structure]/UserScalarValueDataType"
            'BrowsePath="[Structure]/UserArrayValueDataType"
            'BrowsePath="[Structure]/UserIdentityToken/AnonymousIdentityToken"
            'BrowsePath="[Structure]/UserIdentityToken/UserNameIdentityToken"
            'BrowsePath="[Structure]/UserIdentityToken/X509IdentityToken"
            'BrowsePath="[Structure]/UserIdentityToken/IssuedIdentityToken"
            'BrowsePath="[Structure]/FilterOperand/ElementOperand"
            'BrowsePath="[Structure]/FilterOperand/LiteralOperand"
            'BrowsePath="[Structure]/FilterOperand/AttributeOperand"
            'BrowsePath="[Structure]/FilterOperand/SimpleAttributeOperand"
            'BrowsePath="[Structure]/MonitoringFilter/EventFilter"
        End Sub
    End Class
End Namespace
```

Note that this example shows both abstract and concrete data types. Only the concrete (= non-abstract) data types can be used for data transfer. You can determine whether a data type is abstract by reading the IsAbstract attribute of its data type ID node.

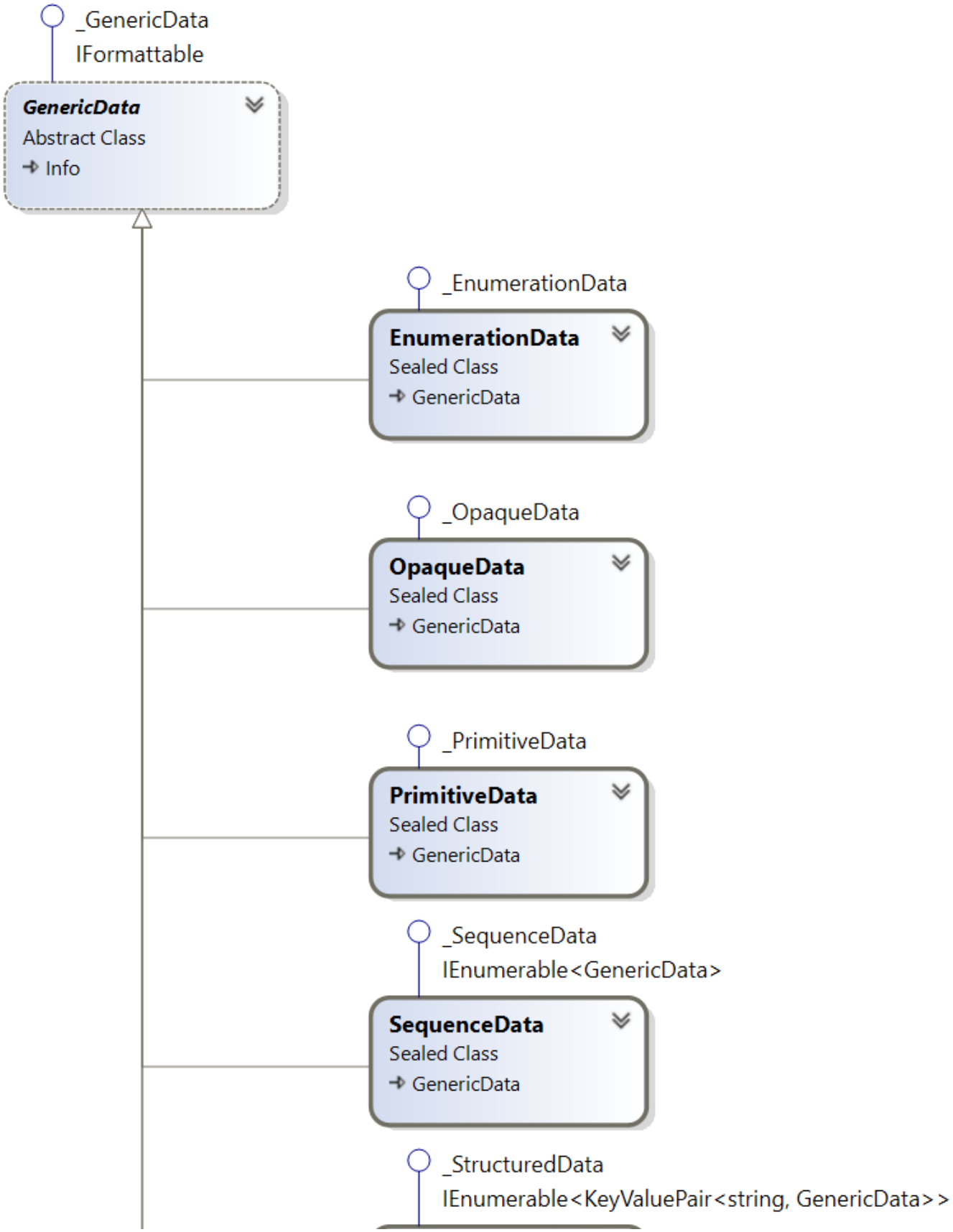
7.2.1.2 Generic data and data types

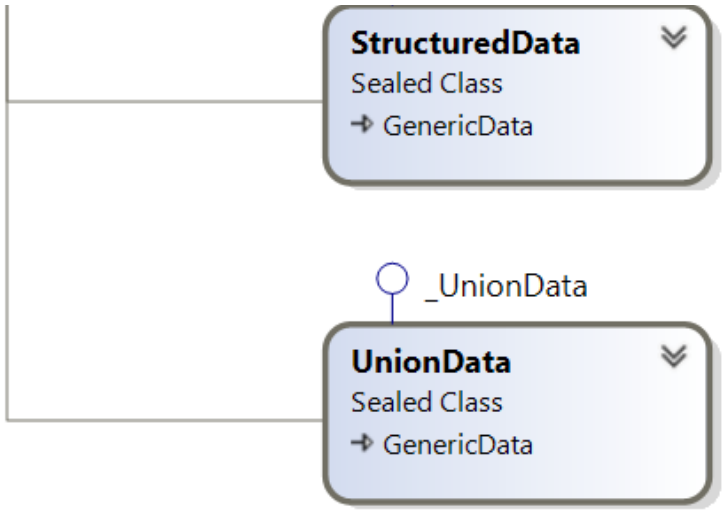
Generic data and data types is a set of classes that can be used to describe various simple and structured types commonly used in today's programming, and create objects that hold values of these types. The classes are designed so that you can easily create and process such generic data and data types, without them being tied to concrete platform or implementation.

7.2.1.2.1 Working with generic data

Generic data is represented by instances of types derived from the GenericData Class. The GenericData Class is a base abstract class that is concretized by its subtypes to enumerations, sequences, structures etc., as described in **Generic data kinds (Section 7.2.1.2.1.1)**.

In This Topic





Data type

Generic data can optionally have a *data type* associated with in, in its DataType Property. The data type provides metadata about what is contained in the generic data value itself. In addition, it provides a means to validate the generic data. Refer to **Generic data validation (Section 7.2.1.2.1.4)** for details on the validation.

> The data type assigned to the DataType Property of any generic value must always be the same as the kind of the generic data itself. For example, only enumeration data type can be assigned to enumeration generic data.
> The GenericData Class and its derivatives check the kind of data type being assigned automatically, and it is not possible to assign a data type of an incorrect kind to the DataType Property (an attempt to do so throws an exception). For this reason, checking the match of data type kinds versus is not necessary during and is not part of generic data validation.

Displaying generic data

When you invoke the ToString Method on some generic data object, or view it in the debugger, you will get a short, on-line rendering of the generic data. It contains the name of its data type (if available), its kind (such as 'structured' or 'enumerable'), and possibly some details specific to the generic data kind (such as the value of primitive data and the .NET type associated with ir, or a length of a sequence). You may get, for example:

```
(ScalarValueDataType) structured
```

If you want to see the information about the internals of the data type, you can use the "V" (verbose) format specifier. For example, this code (C#):

```
Console.WriteLine("{0:V}", genericData);
```

may produce following output (truncated):

```
(ScalarValueDataType) structured
  [BooleanValue] (Boolean) primitive; True {System.Boolean}
  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
  [ByteValue] (Byte) primitive; 153 {System.Byte}
```

```
  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
  [ExpandedNodeIdValue] (ExpandedNodeId) structured
    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance"
{System.String}
    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
    [NodeIdType] (NodeIdType) enumeration; 3 (String)
    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
    [String] (StringNodeId) structured
      [Identifier] (CharArray) primitive; "????" {System.String}
      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
  [FloatValue] (Float) primitive; 78.37176 {System.Single}
  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
  [Int16Value] (Int16) primitive; 2793 {System.Int16}
  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
  [Integer] (Variant) structured
    [ArrayDimensionsSpecified] sequence[1]
      [0] (Bit) primitive; False {System.Boolean}
    [ArrayLengthSpecified] sequence[1]
      [0] (Bit) primitive; False {System.Boolean}
    [Int64] sequence[1]
      [0] (Int64) primitive; 0 {System.Int64}
    [VariantType] sequence[6]
      [0] (Bit) primitive; False {System.Boolean}
      [1] (Bit) primitive; False {System.Boolean}
      [2] (Bit) primitive; False {System.Boolean}
      [3] (Bit) primitive; True {System.Boolean}
      [4] (Bit) primitive; False {System.Boolean}
      [5] (Bit) primitive; False {System.Boolean}
  [LocalizedTextValue] (LocalizedText) structured
    [Locale] (CharArray) primitive; "ko" {System.String}
    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
    [Reserved1] sequence[6]
      [0] (Bit) primitive; False {System.Boolean}
      [1] (Bit) primitive; False {System.Boolean}
      [2] (Bit) primitive; False {System.Boolean}
      [3] (Bit) primitive; False {System.Boolean}
      [4] (Bit) primitive; False {System.Boolean}
      [5] (Bit) primitive; False {System.Boolean}
    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
    [TextSpecified] (Bit) primitive; True {System.Boolean}
  [NodeIdValue] (NodeId) structured
```

If you want to see the internals of the generic data in debugger, you can, of course, inspect its properties, sub-properties and so on. That can be quite a tedious task. For a quick overview, you can view the generic data in the same format as with the "V" (verbose) format specifier. To do so, find the DebugView property of the generic data, and display its contents (possibly using the Text Visualizer in Visual Studio debugger). Note that the DebugView property is private, and therefore is not listed in the reference documentation for the component.

7.2.1.2.1.1 Generic data kinds

Generic data can be of several different kinds: enumeration, opaque, primitive, sequence, structured, or union. You can determine the kind of generic data you are dealing with either by testing it actual type, or using its DataTypeKind

Property. The following table describes the generic data kinds available, and their associated concrete generic data classes.

DataTypeKind member	Associated generic data class
Enumeration	EnumerationData Class
Opaque	OpaqueData Class
Primitive	PrimitiveData Class
Sequence	SequenceData Class
Structured	StructuredData Class
Union	UnionData Class

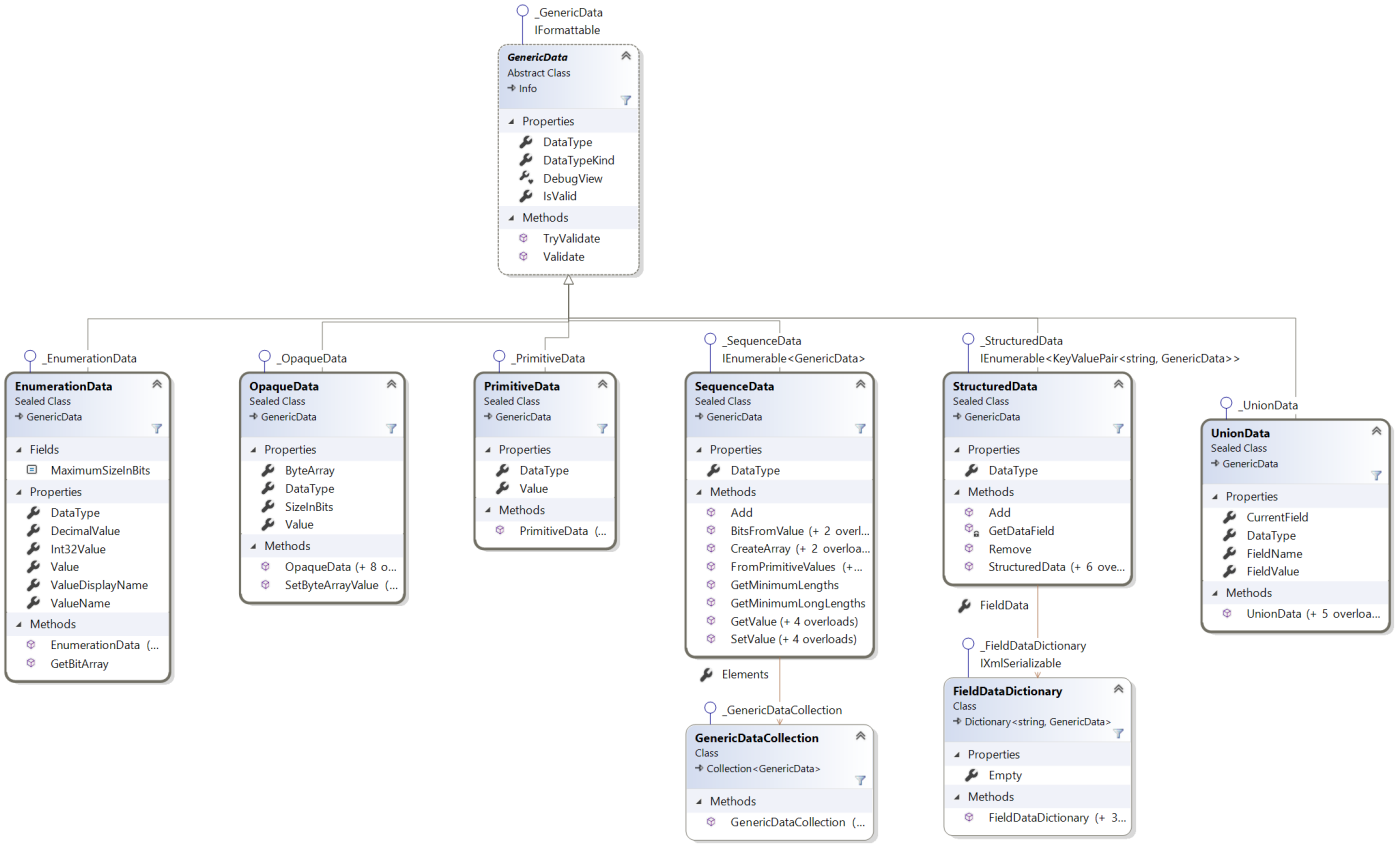
> In data type dictionary model (the only model before OPC UA 1.04), union data is not directly available. A union can, however, be quite efficiently modeled using an appropriately formed structure with optional fields.

For each kind of data type, there is also an associated data type class. See **Data type kinds (Section 7.2.1.2.2.1)** for more details. Note that if there is a data type associated with the generic data (by the means of non-null reference in its DataType Property), the kind of generic data always equals to the kind of the associated data type - this is guaranteed.

Here you will find more information to the specific data kinds:

- **Enumeration data (Section 7.2.1.2.1.1.1)**
- **Opaque data (Section 7.2.1.2.1.1.2)**
- **Primitive data (Section 7.2.1.2.1.1.3)**
- **Sequence data (Section 7.2.1.2.1.1.4)**
- **Structured data (Section 7.2.1.2.1.1.5)**
- **Union data (Section 7.2.1.2.1.1.6)**

The following class diagram shows the generic data clases and their relevant members:



7.2.1.2.1.1.1 Enumeration data

The enumeration data consists of a numeric, optionally also named value, chosen from a specified set of values. It is represented by an instance of the EnumerationData Class.

The Value Property contains the current enumeration value - an integer (64-bits).

If a data type is associated and it has a name for the value, the name is available in the ValueName Property.

7.2.1.2.1.1.2 Opaque data

The opaque data is data whose concrete representation is hidden from its users. It is represented by an instance of the OpaqueData Class. Internally, it keeps the actual data in a System.Collections.BitArray object. The size is given in bits (you can obtained in by getting the value of the SizeInBits Property).

The data is accessible through the Value Property. If you would rather access the data as a regular array of bytes rather than a bit array, you can get the ByteArray Property to do so. If you want to set the whole byte array to a value of your own, use the SetByteArrayValue Method; this is not a property setter because it also requires the size in bits (which does not have to represent a whole number of bytes) be specified.

Note, however, that the two representations (the bit array and the byte array) are not internally synchronized. Setting a bit or byte in one of them does not set the other. Assigning a wholly new value to one of them causes the other be later re-created with a copy of the source data.

7.2.1.2.1.1.3 Primitive data

The primitive data is a value given in the underlying (.NET) type system. It is represented by an instance of the PrimitiveData Class.

The actual primitive value is stored in the Value Property and it can be any System.Object.

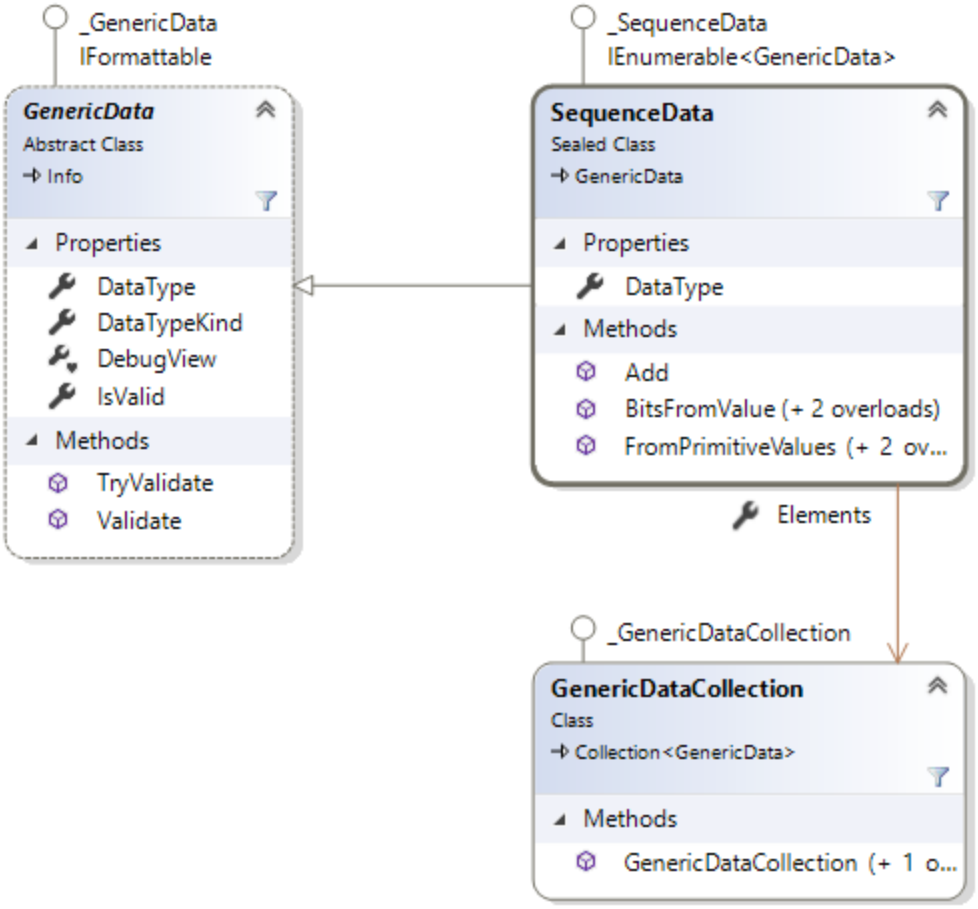
The OPC UA Complex Data extension contains intrinsic data type dictionaries that map specific standard and commonly used data types to primitive data types. For details, refer to **OPC UA Data Type Systems (Section 7.2.1.3)**.

7.2.1.2.1.1.4 Sequence data

The sequence data is an ordered collection of values. It is represented by an instance of the SequenceData Class.

The elements of the sequence data are contained in the Elements Property.

The SequenceData Class is an IEnumerable of GenericData (its elements), and it has an Add Method for the same. It means that you can easily enumerate through its elements, and it can also participate in collection initializers in C# or in VB.NET.



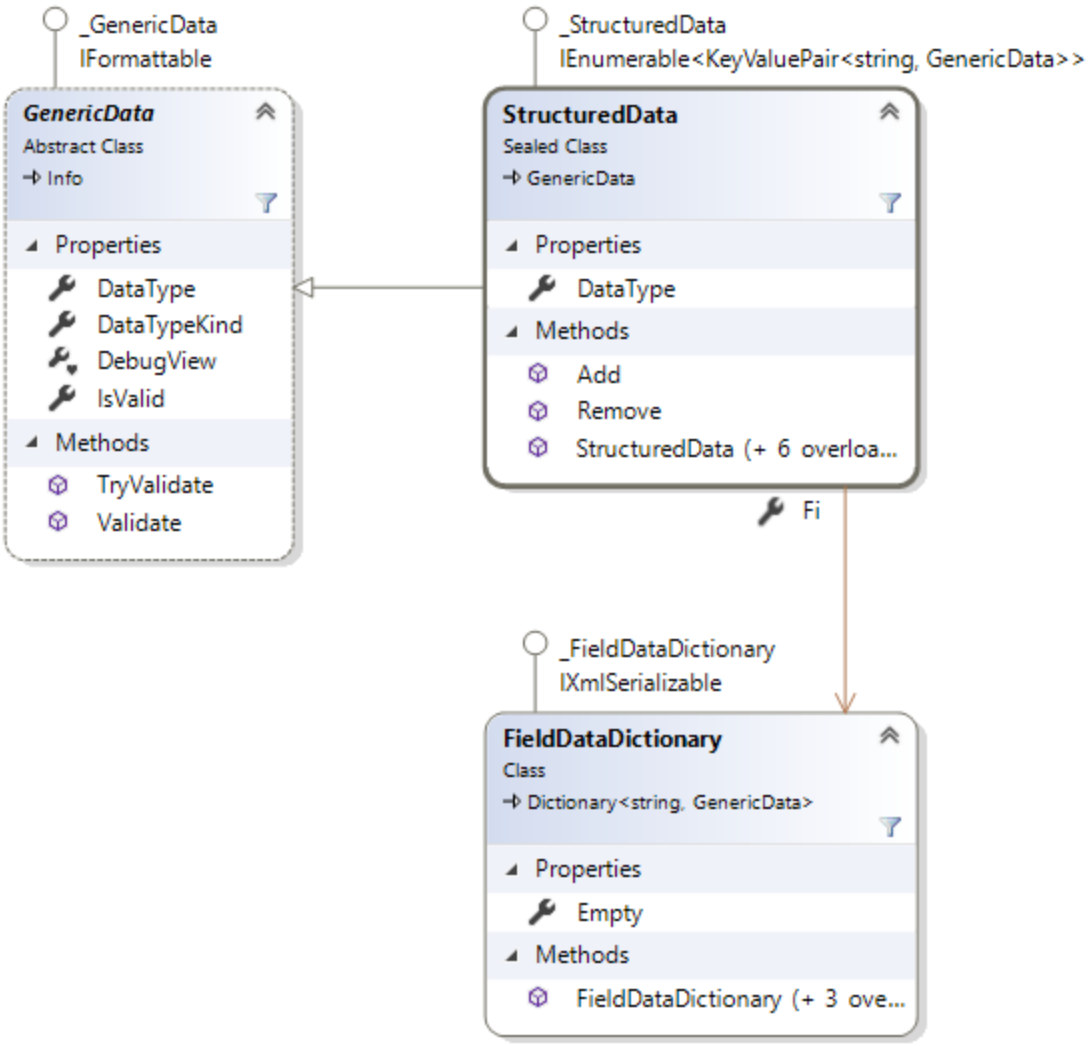
7.2.1.2.1.1.5 Structured data

The structured data is a record, or a representation of union (in data dictionary model, i.e. the only model available before

OPC UA 1.04). It is represented by an instance of the StructuredData Class.

The FieldData Property contains *field data* - a dictionary where the key is a field name, and the value is a generic data of that field.

The StructuredData Class is an IEnumerable of KeyValuePair<String, GenericData> (its field data), and it has an Add Method for the same. It means that you can easily enumerate through its field data, and it can also participate in collection initializers in C# or in VB.NET.



OptionSets

OPC UA has a concept of *OptionSets*. *OptionSet* value is basically a group of named bits. Some *OptionSets* may have an associated validity mask, which determines which of the bits are valid (when reading) or which bits should be written.

OPC Data Client represents *OptionSets* as structures of Boolean values, where each data field of the structure corresponds to one bit of the *OptionSet*. All data fields are optional, and data fields for bits that not contained in the validity mask are omitted from the data.

7.2.1.2.1.1.6 Union data

A (tagged) union contains a single value out of several possible fields (or no value at all). It is represented by an instance UnionData Class.

> In data type dictionary model (the only model before OPC UA 1.04), union data is not directly available. A union can, however, be quite efficiently modeled using an appropriately formed structure with optional fields.

The string FieldName Property contain the name of the field currently contained in the union (this is the "tag" of the union). An empty string in this property denotes a union that currently contains no data.

The FieldValue Property contains the value of the field currently selected in the union.

7.2.1.2.1.2 Creating generic data

You create generic data objects simply by calling their constructors with appropriate arguments, and possibly filling in additional properties that were not passed into the constructors.

The example below shows how different kinds of generic data can be created.

C#

```
// This example shows different ways of constructing generic data.

using System;
using System.Collections;
using OpcLabs.BaseLib.DataTypeModel;

namespace UADocExamples.ComplexData._GenericData
{
    class _Construction
    {
        public static void Main1()
        {
            // Create enumeration data with value of 1.
            var enumerationData = new EnumerationData(1);
            Console.WriteLine(enumerationData);

            // Create opaque data from an array of 2 bytes, specifying its size as 15
bits.
            var opaqueData1 = new OpaqueData(new byte[] {0xAA, 0x55}, sizeInBits:15);
            Console.WriteLine(opaqueData1);

            // Create opaque data from a bit array.
            var opaqueData2 = new OpaqueData(new BitArray(new[] { false, true, false,
true, false }));
            Console.WriteLine(opaqueData2);

            // Create primitive data with System.Double value of 180.0.
            var primitiveData1 = new PrimitiveData(180.0d);
            Console.WriteLine(primitiveData1);

            // Create primitive data with System.String value.
```

```
            var primitiveData2 = new PrimitiveData("Temperature is too high!");
            Console.WriteLine(primitiveData2);

            // Create sequence data with two elements, using collection initializer
syntax.
            var sequenceData1 = new SequenceData
            {
                opaqueData1,
                opaqueData2
            };
            Console.WriteLine(sequenceData1);

            // Create the same sequence data, using the Add method.
            var sequenceData2 = new SequenceData();
            sequenceData2.Elements.Add(opaqueData1);
            sequenceData2.Elements.Add(opaqueData2);
            Console.WriteLine(sequenceData2);

            // Create the same sequence data, using an array (an enumerable) of its
elements.
            var sequenceData3 = new SequenceData(
                new GenericDataCollection(new[] {opaqueData1, opaqueData2}));
            Console.WriteLine(sequenceData3);

            // Create structured data with two members, using collection initializer
syntax.
            var structuredData1 = new StructuredData
            {
                {"Message", primitiveData2},
                {"Status", enumerationData}
            };
            Console.WriteLine(structuredData1);

            // Create the same structured data using the Add method.
            var structuredData2 = new StructuredData();
            structuredData2.Add("Message", primitiveData2);
            structuredData2.Add("Status", enumerationData);
            Console.WriteLine(structuredData2);

            // Create union data.
            var unionData1 = new UnionData("DoubleField", primitiveData1);
            Console.WriteLine(unionData1);
        }
    }
}
```

Object Pascal

```
// This example shows different ways of constructing generic data.

class procedure _Construction.Main;
var
  ByteArray1, ByteArray2: OleVariant;
  EnumerationData: _EnumerationData;
  OpaqueData1, OpaqueData2: _OpaqueData;
  PrimitiveData1, PrimitiveData2: _PrimitiveData;
  SequenceData1: _SequenceData;
```

```
  StructuredData1: _StructuredData;
begin
  // Create enumeration data with value of 1.
  EnumerationData := CoEnumerationData.Create;
  EnumerationData.Value := 1;
  WriteLn(EnumerationData.ToString);

  // Create opaque data from an array of 2 bytes, specifying its size as 15 bits.
  OpaqueData1 := CoOpaqueData.Create;
  ByteArray1 := VarArrayCreate([0, 1], varVariant);
  ByteArray1[0] := $AA;
  ByteArray1[1] := $55;
  OpaqueData1.SetByteArrayValue(ByteArray1, 15);
  WriteLn(OpaqueData1.ToString);
  // Create opaque data from an array of 1 bytes, specifying its size as 5 bits.
  OpaqueData2 := CoOpaqueData.Create;
  ByteArray2 := VarArrayCreate([0, 0], varVariant);
  ByteArray2[0] := $A;
  OpaqueData2.SetByteArrayValue(ByteArray2, 5);
  WriteLn(OpaqueData2.ToString);

  // Create primitive data with System.Double value of 180.0.
  PrimitiveData1 := CoPrimitiveData.Create;
  PrimitiveData1.Value := 180.0;
  WriteLn(PrimitiveData1.ToString);

  // Create primitive data with System.String value.
  PrimitiveData2 := CoPrimitiveData.Create;
  PrimitiveData2.Value := 'Temperature is too high!';
  WriteLn(PrimitiveData2.ToString);

  // Create sequence data with two elements, using the Add method.
  SequenceData1 := CoSequenceData.Create;
  SequenceData1.Elements.Add(OpaqueData1);
  SequenceData1.Elements.Add(OpaqueData2);
  WriteLn(SequenceData1.ToString);

  // Create structured data with two members, using the Add method.
  StructuredData1 := CoStructuredData.Create;
  StructuredData1.Add('Message', PrimitiveData2);
  StructuredData1.Add('Status', EnumerationData);
  WriteLn(StructuredData1.ToString);
end;
```

PHP

```
// This example shows different ways of constructing generic data.

// Create enumeration data with value of 1.
$EnumerationData = new COM("OpcLabs.BaseLib.DataTypeModel.EnumerationData");
$EnumerationData->Value = 1;
printf("%s\n", $EnumerationData);

// Create opaque data from an array of 2 bytes, specifying its size as 15 bits.
$OpaqueData1 = new COM("OpcLabs.BaseLib.DataTypeModel.OpaqueData");
$ByteArray1[0] = 0xAA;
$ByteArray1[1] = 0x55;
```

```
$OpaqueData1->SetByteArrayValue($ByteArray1, 15);
printf("%s\n", $OpaqueData1);

// Create opaque data from a bit array.
$OpaqueData2 = new COM("OpcLabs.BaseLib.DataTypeModel.OpaqueData");
$BitArray = $OpaqueData2->Value;
$BitArray->Length = 5;
$BitArray->Set(0, false);
$BitArray->Set(1, true);
$BitArray->Set(2, false);
$BitArray->Set(3, true);
$BitArray->Set(4, false);
printf("%s\n", $OpaqueData2);

// Create primitive data with System.Double value of 180.0.
$PrimitiveData1 = new COM("OpcLabs.BaseLib.DataTypeModel.PrimitiveData");
$PrimitiveData1->Value = 180.0;
printf("%s\n", $PrimitiveData1);

// Create primitive data with System.String value.
$PrimitiveData2 = new COM("OpcLabs.BaseLib.DataTypeModel.PrimitiveData");
$PrimitiveData2->Value = "Temperature is too high!";
printf("%s\n", $PrimitiveData2);

// Create sequence data with two elements, using the Add method.
$SequenceData1 = new COM("OpcLabs.BaseLib.DataTypeModel.SequenceData");
$SequenceData1->Elements->Add($OpaqueData1);
$SequenceData1->Elements->Add($OpaqueData2);
printf("%s\n", $SequenceData1);

// Create structured data with two members, using the Add method.
$StructuredData1 = new COM("OpcLabs.BaseLib.DataTypeModel.StructuredData");
$StructuredData1->Add("Message", $PrimitiveData2);
$StructuredData1->Add("Status", $EnumerationData);
printf("%s\n", $StructuredData1);
```

VB.NET

```
' This example shows different ways of constructing generic data.

Imports System
Imports System.Collections
Imports OpcLabs.BaseLib.DataTypeModel

Namespace UADocExamples.ComplexData._GenericData

    Friend Class _Construction

        Public Shared Sub Main1()
            ' Create enumeration data with value of 1.
            Dim enumerationData = New EnumerationData(1)
            Console.WriteLine(enumerationData)

            ' Create opaque data from an array of 2 bytes, specifying its size as 15
bits.
            Dim opaqueData1 = New OpaqueData(New Byte() {170, 85}, sizeInBits:=15)
            Console.WriteLine(opaqueData1)
```

```
            ' Create opaque data from a bit array.
            Dim opaqueData2 = New OpaqueData(New BitArray(New Boolean() {False, True,
False, True, False}))
            Console.WriteLine(opaqueData2)

            ' Create primitive data with System.Double value of 180.0.
            Dim primitiveData1 = New PrimitiveData(180)
            Console.WriteLine(primitiveData1)

            ' Create primitive data with System.String value.
            Dim primitiveData2 = New PrimitiveData("Temperature is too high!")
            Console.WriteLine(primitiveData2)

            ' Create sequence data with two elements, using collection initializer
syntax.
            Dim sequenceData1 = New SequenceData() From {opaqueData1, opaqueData2}
            Console.WriteLine(sequenceData1)

            ' Create the same sequence data, using the Add method.
            Dim sequenceData2 = New SequenceData
            sequenceData2.Elements.Add(opaqueData1)
            sequenceData2.Elements.Add(opaqueData2)
            Console.WriteLine(sequenceData2)

            ' Create the same sequence data, using an array (an enumerable) of its
elements.
            Dim sequenceData3 = New SequenceData(New GenericDataCollection(New
OpaqueData() {opaqueData1, opaqueData2}))
            Console.WriteLine(sequenceData3)

            ' Create structured data with two members, using collection initializer
syntax.
            Dim structuredData1 = New StructuredData() From { _
                {"Message", primitiveData2}, _
                {"Status", enumerationData}}
            Console.WriteLine(structuredData1)

            ' Create the same structured data using the Add method.
            Dim structuredData2 = New StructuredData()
            structuredData2.Add("Message", primitiveData2)
            structuredData2.Add("Status", enumerationData)
            Console.WriteLine(structuredData2)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows different ways of constructing generic data.

Option Explicit

' Create enumeration data with value of 1.
Dim EnumerationData: Set EnumerationData =
CreateObject("OpcLabs.BaseLib.DataTypeModel.EnumerationData")
EnumerationData.Value = 1
```

```
WScript.Echo EnumerationData

' Create opaque data from an array of 2 bytes, specifying its size as 15 bits.
Dim OpaqueData1: Set OpaqueData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.OpaqueData")
OpaqueData1.SetByteArrayValue Array(&HAA, &H55), 15
WScript.Echo OpaqueData1

' Create opaque data from a bit array.
Dim OpaqueData2: Set OpaqueData2 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.OpaqueData")
Dim BitArray: Set BitArray = OpaqueData2.Value
BitArray.Length = 5
BitArray(0) = False
BitArray(1) = True
BitArray(2) = False
BitArray(3) = True
BitArray(4) = False
WScript.Echo OpaqueData2

' Create primitive data with System.Double value of 180.0.
Dim PrimitiveData1: Set PrimitiveData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.PrimitiveData")
PrimitiveData1.Value = 180.0
WScript.Echo PrimitiveData1

' Create primitive data with System.String value.
Dim PrimitiveData2: Set PrimitiveData2 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.PrimitiveData")
PrimitiveData2.Value = "Temperature is too high!"
WScript.Echo PrimitiveData2

' Create sequence data with two elements, using the Add method.
Dim SequenceData1: Set SequenceData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.SequenceData")
SequenceData1.Elements.Add OpaqueData1
SequenceData1.Elements.Add OpaqueData2
WScript.Echo SequenceData1

' Create structured data with two members, using the Add method.
Dim StructuredData1: Set StructuredData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.StructuredData")
StructuredData1.Add "Message", PrimitiveData2
StructuredData1.Add "Status", EnumerationData
WScript.Echo StructuredData1
```

7.2.1.2.1.3 Processing generic data

In order to process generic data (a sub-type of GenericData Class), you need to first determine its kind (see **Generic data kinds (Section 7.2.1.2.1.1)**). You can do it by

- getting its DataTypeKind Property, and then having a switch or other kind of conditional statement, taking care of different data type kinds, or

- attempting to cast it (e.g. using the as operator in C#, or TryCast in VB.NET) to a concrete generic data class (e.g. StructuredData), and testing whether it has succeeded.

All generic data classes share a common DataType Property, which contains an optional data type object.

After you have determined the kind of generic data you are dealing with, you can use the generic data object cast to appropriate type, and work with properties and methods that are specific to that data type kind. For example, SequenceData contains the elements of the sequence in its Elements Property.

The following example shows how generic data can be processed.

C#

```
// Shows how to process generic data type, displaying some of its properties,
recursively.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._GenericData
{
    class DataTypeKind1
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            UAGenericObject genericObject;
            try
            {
                genericObject = (UAGenericObject)client.ReadValue(endpointDescriptor,
nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
```

```
            // Process the generic data type. We will inspect some of its properties,
and dump them.
            ProcessGenericData(genericObject.GenericData, maximumDepth: 2);
        }


        // Process the generic data type. Its structure can sometimes be quite deep,
therefore we are limiting the depth
        // of the recursion using maximumDepth.
        public static void ProcessGenericData(GenericData genericData, int
maximumDepth)
        {
            if (maximumDepth == 0)
                return;

            Console.WriteLine();
            Console.WriteLine("genericData.DataType: {0}", genericData.DataType);

            switch (genericData.DataTypeKind)
            {
                case DataTypeKind.Enumeration:
                    Console.WriteLine("The generic data is an enumeration.");
                    var enumerationData = (EnumerationData) genericData;
                    Console.WriteLine("Its value is {0}.", enumerationData.Value);
                    // There is also a ValueName that you can inspect (if known).
                    break;

                case DataTypeKind.Opaque:
                    Console.WriteLine("The generic data is opaque.");
                    var opaqueData = (OpaqueData) genericData;
                    Console.WriteLine("Its size is {0} bits.", opaqueData.SizeInBits);
                    Console.WriteLine("The data bytes are {0}.",
BitConverter.ToString(opaqueData.ByteArray));
                    // Use the Value property (a BitArray) if you need to access the
value bit by bit.
                    break;

                case DataTypeKind.Primitive:
                    Console.WriteLine("The generic data is primitive.");
                    var primitiveData = (PrimitiveData) genericData;
                    Console.WriteLine("Its value is \"{0}\".", primitiveData.Value);
                    break;

                case DataTypeKind.Sequence:
                    Console.WriteLine("The generic data is a sequence.");
                    var sequenceData = (SequenceData) genericData;
                    Console.WriteLine("It has {0} elements.",
sequenceData.Elements.Count);
                    Console.WriteLine("A dump of the elements follows.");
                    foreach (GenericData element in sequenceData.Elements)
                        ProcessGenericData(element, maximumDepth - 1);
                    break;

                case DataTypeKind.Structured:
                    Console.WriteLine("The generic data is structured.");
                    var structuredData = (StructuredData) genericData;
                    Console.WriteLine("It has {0} field data members.",
```

```
structuredData.FieldData.Count);
                    Console.WriteLine("The names of the fields are: {0}.",
                        String.Join(", ", structuredData.FieldData.Keys));

                    Console.WriteLine("A dump of each of the fields follows.");
                    foreach (KeyValuePair<string, GenericData> pair in
structuredData.FieldData)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Field name: {0}", pair.Key);
                        ProcessGenericData(pair.Value, maximumDepth - 1);
                    }
                    break;

                case DataTypeKind.Union:
                    Console.WriteLine("The generic data is union.");
                    var unionData = (UnionData)genericData;
                    Console.WriteLine("The name of current field is: {0}",
unionData.FieldName);
                    Console.WriteLine("Current field value is: {0}",
unionData.FieldValue);
                    break;
            }
        }
    }
}
```

Object Pascal

```
// Shows how to process generic data type, displaying some of its properties,
recursively

class procedure DataTypeKind1.Main;
var
  Client: _EasyUAClient;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node. We know that this node returns complex data, so we can type cast to
UAGenericObject.

  try
    GenericObject := _UAGenericObject(IUnknown(Client.ReadValue(EndpointDescriptor,
NodeDescriptor)));
  except
    on E: EOleException do
```

```
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Process the generic data type. We will inspect some of its properties, and dump
them.
  ProcessGenericData(GenericObject.GenericData, 2);
end;

Function VariantToBytes(Const Value: OleVariant): TBytes;
Var
  Size: Integer;
  pData: Pointer;
Begin
  Size := Succ(VarArrayHighBound(Value, 1) - VarArrayLowBound(Value, 1));
  SetLength(Result, Size);
  pData := VarArrayLock(Value);
  Try
    Move(pData^, Pointer(Result)^, Size);
  Finally
    VarArrayUnlock(Value);
  End;
End;

class procedure DatatypeKind1.ProcessGenericData(GenericData:
OpcLabs_BaseLib_TLB._GenericData; MaximumDepth: Cardinal);
var
  ByteArray: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVARIANT;
  EnumerationData: _EnumerationData;
  First: boolean;
  Keys: string;
  OpaqueData: _OpaqueData;
  PrimitiveData: _PrimitiveData;
  SequenceData: _SequenceData;
  StructuredData: _StructuredData;
  Value: OpcLabs_BaseLib_TLB._GenericData;
begin
  if MaximumDepth = 0 then
    Exit;

  WriteLn;
  WriteLn('genericData.DataType: ', GenericData.DataType.ToString);

  case GenericData.DataTypeKind of
    DataTypeKind_Enumeration:
      begin
        WriteLn('The generic data is an enumeration.');
        EnumerationData := GenericData as _EnumerationData;
        WriteLn(Format('Its value is %s.', [EnumerationData.Value.ToString]));
        // There is also a ValueName that you can inspect (if known).
      end;
    DataTypeKind_Opaque:
```

```
      begin
        WriteLn('The generic data is opaque.');
        OpaqueData := GenericData as _OpaqueData;
        WriteLn(Format('Its size is %d bits.', [OpaqueData.SizeInBits]));
        TVarData(ByteArray).VType := varArray;
        TVarData(ByteArray).VArray := PVarArray(OpaqueData.ByteArray);
        WriteLn(Format('The data bytes are %s.',
[TEncoding.ANSI.GetString(VariantToBytes(ByteArray))]));
        // Use the Value property (a BitArray) if you need to access the value bit by
bit.
      end;
    DataTypeKind_Primitive:
      begin
        WriteLn('The generic data is primitive.');
        PrimitiveData := GenericData as _PrimitiveData;
        WriteLn(Format('Its value is "%s".', [PrimitiveData.Value]));
      end;
    DataTypeKind_Sequence:
      begin
        WriteLn('The generic data is a sequence.');
        SequenceData := GenericData as _SequenceData;
        WriteLn(Format('It has %s elements.', [SequenceData.Elements.Count.ToString]));
        WriteLn('A dump of the elements follows.');
        ElementEnumerator := SequenceData.Elements.GetEnumerator;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          ProcessGenericData(IUnknown(Element) as OpcLabs_BaseLib_TLB._GenericData,
MaximumDepth - 1);
        end;
      end;
    DataTypeKind_Structured:
      begin
        WriteLn('The generic data is structured.');
        StructuredData := GenericData as _StructuredData;
        WriteLn(Format('It has %s field data members.',
[StructuredData.FieldData.Count.ToString]));
        ElementEnumerator := StructuredData.FieldData.GetEnumerator;
        Keys := '';
        First := True;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          if First then
            First := False
          else
            Keys := Keys + ', ';
          Keys := Keys + Element.Key;
        end;
        WriteLn(Format('The names of the fields are: %s.', [Keys]));

        WriteLn('A dump of each of the fields follows.');
        ElementEnumerator := StructuredData.FieldData.GetEnumerator;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          WriteLn;
          WriteLn(Format('Field name: %s', [Element.Key]));
          Value := IUnknown(Element.Value) as OpcLabs_BaseLib_TLB._GenericData;
          ProcessGenericData(Value, MaximumDepth - 1);
```

```
        end;
      end;
  end;

end;
```

VB.NET

```
' Shows how to process generic data type, displaying some of its properties,
recursively.

Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._GenericData

    Friend Class DataTypeKind1

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            Dim genericObject As UAGenericObject
            Try
                genericObject = CType(client.ReadValue(endpointDescriptor,
nodeDescriptor), UAGenericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Process the generic data type. We will inspect some of its properties,
and dump them.
            ProcessGenericData(genericObject.GenericData, maximumDepth:=2)
        End Sub


        ' Process the generic data type. Its structure can sometimes be quite deep,
```

```
therefore we are limiting the depth
        ' of the recursion using maximumDepth.
        Public Shared Sub ProcessGenericData(ByVal genericData As GenericData, ByVal
maximumDepth As Integer)
            If (maximumDepth = 0) Then
                Return
            End If

            Console.WriteLine()
            Console.WriteLine("genericData.DataType: {0}", genericData.DataType)

            Select Case (genericData.DataTypeKind)
                Case DataTypeKind.Enumeration
                    Console.WriteLine("The generic data is an enumeration.")
                    Dim enumerationData = CType(genericData, EnumerationData)
                    Console.WriteLine("Its value is {0}.", enumerationData.Value)
                    ' There is also a ValueName that you can inspect (if known).

                Case DataTypeKind.Opaque
                    Console.WriteLine("The generic data is opaque.")
                    Dim opaqueData = CType(genericData, OpaqueData)
                    Console.WriteLine("Its size is {0} bits.", opaqueData.SizeInBits)
                    Console.WriteLine("The data bytes are {0}.",
BitConverter.ToString(opaqueData.ByteArray))
                    ' Use the Value property (a BitArray) if you need to access the
value bit by bit.

                Case DataTypeKind.Primitive
                    Console.WriteLine("The generic data is primitive.")
                    Dim primitiveData = CType(genericData, PrimitiveData)
                    Console.WriteLine("Its value is ""{0}"".", primitiveData.Value)

                Case DataTypeKind.Sequence
                    Console.WriteLine("The generic data is a sequence.")
                    Dim sequenceData = CType(genericData, SequenceData)
                    Console.WriteLine("It has {0} elements.",
sequenceData.Elements.Count)
                    Console.WriteLine("A dump of the elements follows.")
                    For Each element As GenericData In sequenceData.Elements
                        ProcessGenericData(element, (maximumDepth - 1))
                    Next

                Case DataTypeKind.Structured
                    Console.WriteLine("The generic data is structured.")
                    Dim structuredData = CType(genericData, StructuredData)
                    Console.WriteLine("It has {0} field data members.",
structuredData.FieldData.Count)
                    Console.WriteLine("The names of the fields are: {0}.",
String.Join(", ", structuredData.FieldData.Keys))

                    Console.WriteLine("A dump of each of the fields follows.")
                    For Each pair As KeyValuePair(Of String, GenericData) In
structuredData.FieldData
                        Console.WriteLine()
                        Console.WriteLine("Field name: {0}", pair.Key)
                        ProcessGenericData(pair.Value, (maximumDepth - 1))
                    Next
```

```
                Case DataTypeKind.Union
                    Console.WriteLine("The generic data is union.")
                    Dim unionData = CType(genericData, UnionData)
                    Console.WriteLine("The name of current field is: {0}",
unionData.FieldName)
                    Console.WriteLine("Current field value is: {0}",
unionData.FieldValue)
            End Select

        End Sub
    End Class
End Namespace
```

In many cases, you will not be processing the generic data alone, but in parallel with the data type. See **Processing data types (Section 7.2.1.2.2.3)** for details.

7.2.1.2.1.4 Generic data validation

When the GenericData is returned from the server by the component (e.g. from read operations or by subscriptions), its DataType Property property is always filled in (i.e. not null).

When the GenericData is passed in to the component for transfer to the server (e.g. to a write operation), filling in the DataType Property optional. When not null, the component checks the value in GenericDataagainst the data type given by this property, and issues an error if the value does not conform to the type. It is allowed for this property to be not null at some higher level, and be null at lower level, such as for an individual field of a structure.

In This Topic

When the data type is given and passed to the component, it is only used for the checks described above. It does not influence the actual encoding of the values, and the encoding process may further reject some input values as not conforming to the encoding.

Validation rules

Generic data can only be validated if it has a non-null DataType associated with it. When the DataType is null, the validation always succeeds.

For all kinds of generic data, the associated data type must be **complete** and **terminable** for the validation to succeed. For discussions of these terms, see **Working with data types (Section 7.2.1.2.2)**.

Besides these common rules, specific rules apply to different kinds of generic data:

- For enumeration data, if the size in bits (SizeInBits Property) is given in the data type (is non-negative), the actual value must not exceed the size that fits into the number of bits given by the data type.
- For opaque data, if the size in bits (SizeInBits Property in the OpaqueDataType Class) is given in the data type (is non-negative), the actual size in bits of the opaque value must be precisely equal to the size given by the data type.
- Primitive data is always valid.
- For sequence data, all elements must be non-null and also valid. In addition, if a length is given in the data type

(Length Property), the actual count of elements in the sequence must be precisely equal to the length given by the data type.
- For structured data, all field values must be non-null and also valid. In addition, all mandatory (non-optional), non-length fields that are defined by the data type must also be present in the actual field data (FieldData Property).
- For union data, if the FieldName Property (current field name) is empty, the current union value (the FieldValue Property) must be null. If the FieldName Property (current field name) is not empty, it must correspond to one of the data fields in the DataFields Property. This data field must not be a length field or switch field and must not be optional, and its data value must be null.

Invoking the validation

You can also invoke the validation of generic data from your code, e.g. to prevent errors upon writing the data. To do so, you can do one of the following:

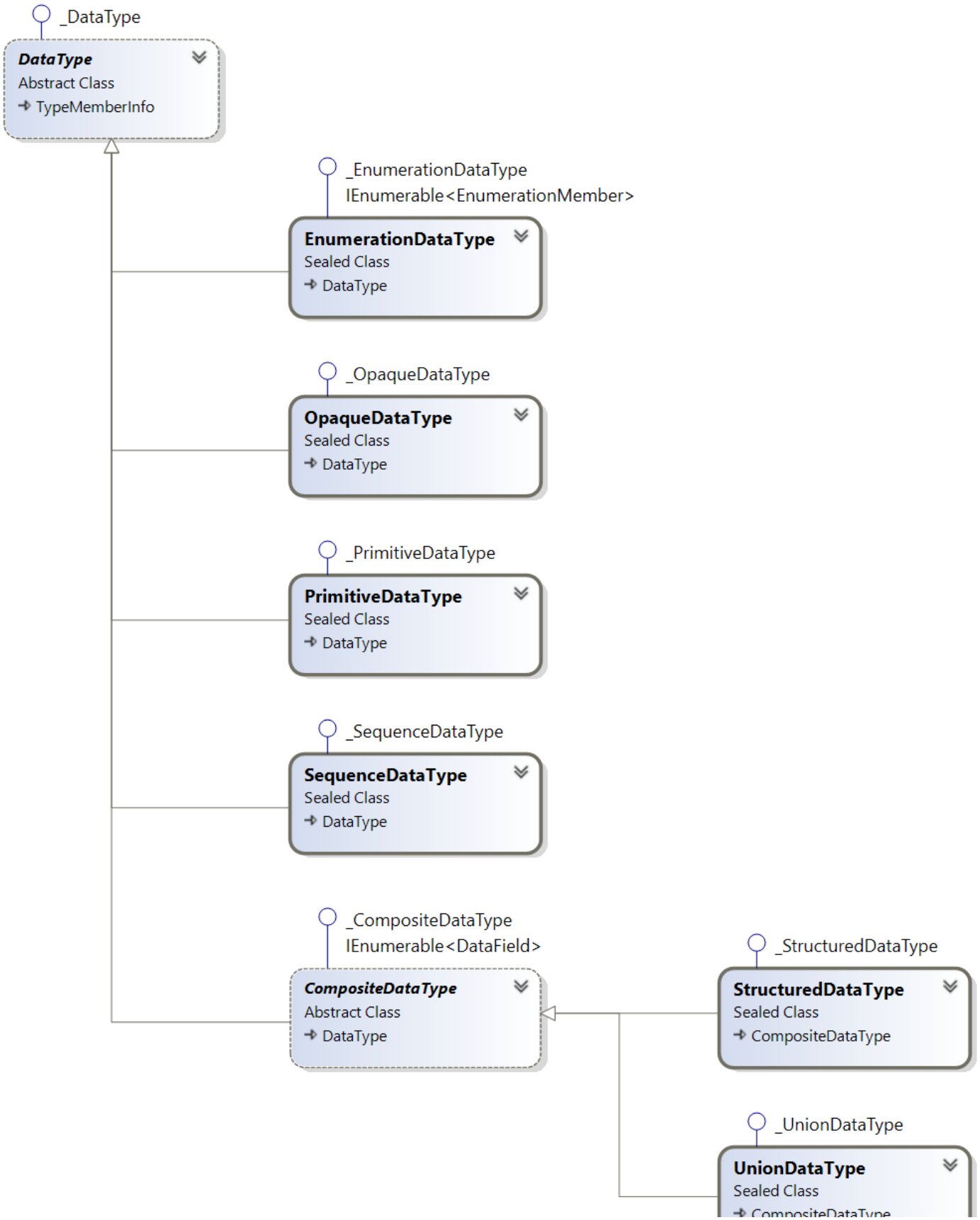
- Inspect the IsValid Property of the generic data. The return value (a boolean) indicates whether the current data is valid.
- Call the TryValidate Method on the generic data. This method attempts to validate the current data. It returns a non-null Exception if the validations fails; it returns a null reference otherwise.
- Call the Validate Method on the generic data. This method validates the current data. It throws an exception when the validation fails.

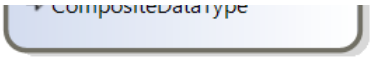
7.2.1.2.2 Working with data types

A data type is represented by an instance of some type derived from the DataType Class. The DataType Class itself is abstract, but there are several concrete classes that derive from it.

Data types can be of several different kinds: enumeration, opaque, primitive, sequence, or structured. Refer to **Data type kinds (Section 7.2.1.2.2.1)** for mode details on the individual kinds. You can determine the kind of data type you are dealing with either by testing it actual type, or using its Kind Property.

In This Topic





Refer to **Creating data types (Section 7.2.1.2.2.2)** if you need to create an instance of some data type (derived from the DataType Class; that is actually rare, maybe just for generic data validation purposes - see **Generic data validation (Section 7.2.1.2.1.4)**). If you need to process a data type object that you have obtained from the comonent, refer to **Processing data types (Section 7.2.1.2.2.3)**.

Recursive data types

The data type is **recursive** when it contains a cycle, directly or indirectly. For example, a structured data type may contain a field that is a sequence of the same data type. Recursive data types as such are not invalid, and are even not so uncommon; for example, the Variant data type is defined recursively. You can determine whether the data type is recursive using its IsRecursive Property.

For the data type to be usable, however, it must be **terminable**. The data type is terminable if it is capable of producing finite data. All non-recursive data types are terminable, but not all recursive types are. For example, in the example with a structured data type above, the data type would be terminable if the field was optional, or if the length of the sequence was not fixed (and could be zero). In such cases the expansion of the data type can be stopped at some point, by omitting the field form the data, or having a zero length sequence. If, however, the field was mandatory and the sequence had a fixed non-zero length, such data type would not be terminable - and would be basically useless, because the only data it can describe is infinite.

You can determine whether a data type is terminable using its IsTerminable Property.

Incomplete types

The data type is **complete** when all its constituent data type are filled in are are themselves complete. For example, the field in the structured data type must have their types defined by a non-null data type.

Normally, when data types are constructed, you create them already as complete. But in order construct recursive data types, an incomplete data type needs to be created first, and the loop - the missing data type - is closed later. Incomplete data types should therefore exist only temporarily, during their construction. They must not participate in further operations, and generic data is invalid for operations when it has an incomplete data type associated with it - see **Generic data validation (Section 7.2.1.2.1.4)**.

You can determine whether a data type is complete using its IsComplete Property.

Displaying data types

When you invoke the ToString Method on some data type, or view it in the debugger, you will get a short, on-line rendering of the data type. It contains its name (if it has a name), its kind (such as 'structured' or 'enumerable'), and possibly some details specific to the data type kind (such as the .NET type associated with primitive data). You may get, for example:

```
(ScalarValueDataType) structured
```

If you want to see the information about the internals of the data type, you can use the "V" (verbose) format specifier. For example, this code (C#):

```
Console.WriteLine("{0:V}", dataType);
```

may produce following output (truncated):

```
ScalarValueDataType = structured
  [BooleanValue] Boolean = primitive(System.Boolean)
  [ByteStringValue] ByteString = primitive(System.Byte[])
  [ByteValue] Byte = primitive(System.Byte)
  [DateTimeValue] DateTime = primitive(System.DateTime)
  [DoubleValue] Double = primitive(System.Double)
  [EnumerationValue] Int32 = primitive(System.Int32)
  [ExpandedNodeIdValue] ExpandedNodeId = structured
    [ByteString] optional ByteStringNodeId = structured
    [Identifier] ByteString = primitive(System.Byte[])
      [NamespaceIndex] UInt16 = primitive(System.UInt16)
    [FourByte] optional FourByteNodeId = structured
      [Identifier] UInt16 = primitive(System.UInt16)
      [NamespaceIndex] Byte = primitive(System.Byte)
    [Guid] optional GuidNodeId = structured
      [Identifier] Guid = primitive(System.Guid)
      [NamespaceIndex] UInt16 = primitive(System.UInt16)
    [NamespaceURI] optional CharArray = primitive(System.String)
    [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
    [NodeIdType] switch NodeIdType = enumeration(6)
      TwoByte = 0
      FourByte = 1
      Numeric = 2
      String = 3
      Guid = 4
      ByteString = 5
    [Numeric] optional NumericNodeId = structured
      [Identifier] UInt32 = primitive(System.UInt32)
      [NamespaceIndex] UInt16 = primitive(System.UInt16)
    [ServerIndex] optional UInt32 = primitive(System.UInt32)
    [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
    [String] optional StringNodeId = structured
      [Identifier] CharArray = primitive(System.String)
      [NamespaceIndex] UInt16 = primitive(System.UInt16)
    [TwoByte] optional TwoByteNodeId = structured
    [Identifier] Byte = primitive(System.Byte)
  [FloatValue] Float = primitive(System.Single)
  [GuidValue] Guid = primitive(System.Guid)
  [Int16Value] Int16 = primitive(System.Int16)
  [Int32Value] Int32 = primitive(System.Int32)
  [Int64Value] Int64 = primitive(System.Int64)
  [Integer] Variant = structured
    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
    [ArrayLength] length optional Int32 = primitive(System.Int32)
    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
    [Byte] optional sequence[*] of Byte = primitive(System.Byte)
```

The component will correctly (without looping infinitely) display recursive data types as well.

If you want to see the internals of the data type in debugger, you can, of course, inspect its properties, sub-properties and so on. That can be quite a tedious task. For a quick overview, you can view the data type in the same format as with the "V" (verbose) format specifier. To do so, find the DebugView property of the data type, and display its contents (possibly using the Text Visualizer in Visual Studio debugger). Note that the DebugView property is private, and therefore is not

listed in the reference documentation for the component.

7.2.1.2.2.1 Data type kinds

Data types can be of several different kinds: enumeration, opaque, primitive, sequence, or structured. You can determine the kind of data type you are dealing with either by testing it actual type, or using its Kind Property.

The following table describes the data type kinds available, and their associated concrete data type classes.

DataTypeKind member	Associated data type class
Enumeration	EnumerationDataType Class
Opaque	OpaqueDataType Class
Primitive	PrimitiveDataType Class
Sequence	SequenceDataType Class
Structured	StructuredDataType Class
Union	UnionDataType Class

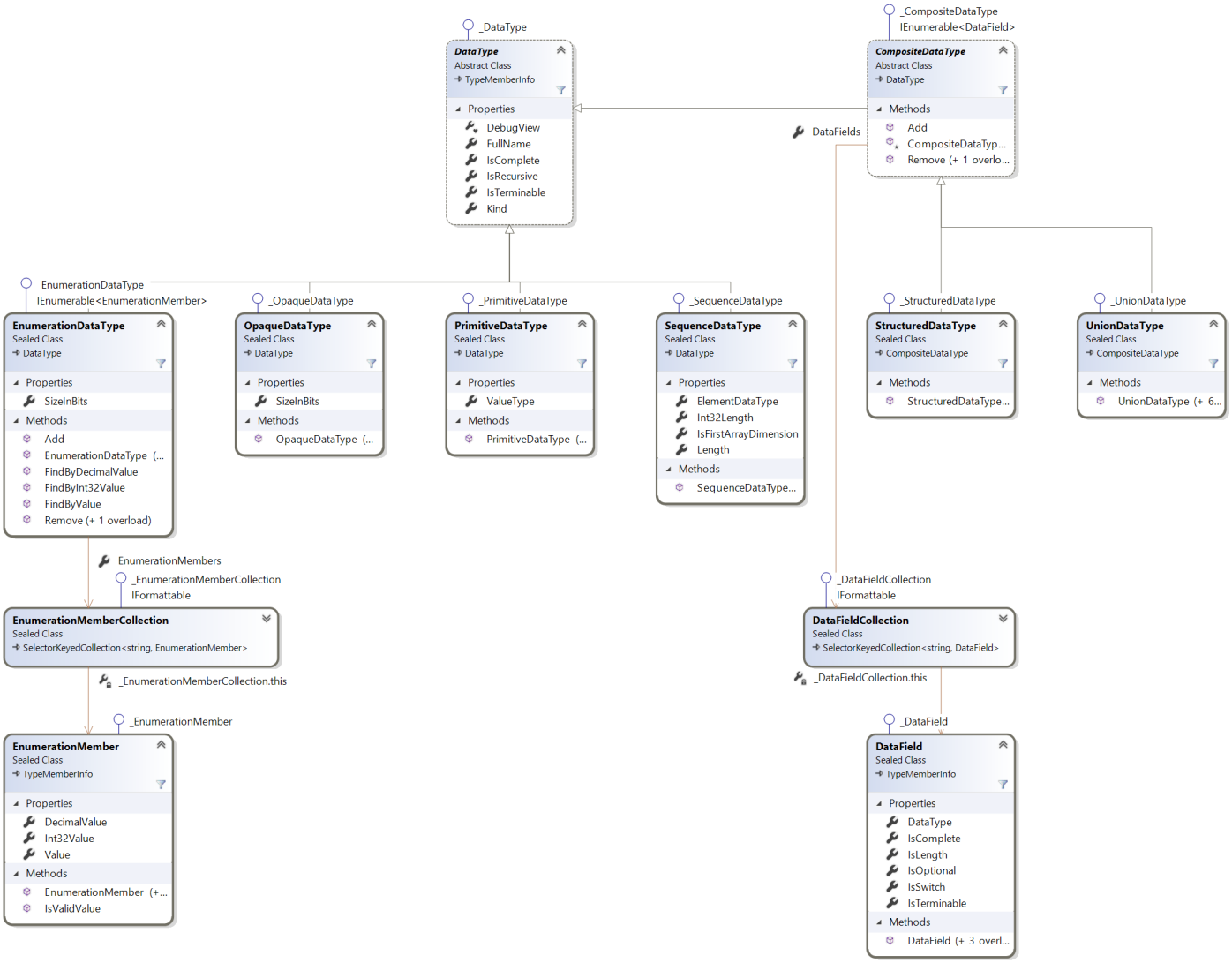
In data type dictionary model (the only model before OPC UA 1.04), union data is not directly available. A union can, however, be quite efficiently modeled using an appropriately formed structure with optional fields.

For each kind of data type, there is also an associated generic data class. See **Generic data kinds (Section 7.2.1.2.1.1)** for more details.

Here you will find more information to the specific data type kinds:

- **Enumeration data type (Section 7.2.1.2.2.1.1)**
- **Opaque data type (Section 7.2.1.2.2.1.2)**
- **Primitive data type (Section 7.2.1.2.2.1.3)**
- **Sequence data type (Section 7.2.1.2.2.1.4)**
- **Structured data type (Section 7.2.1.2.2.1.5)**
- **Union data type (Section 7.2.1.2.2.1.6)**

The following class diagram shows the data type classes and their relevant members:



7.2.1.2.2.1.1 Enumeration data type

The enumeration data type is a data type consisting of a set of named values. It is represented by an instance of the EnumerationDataType Class.

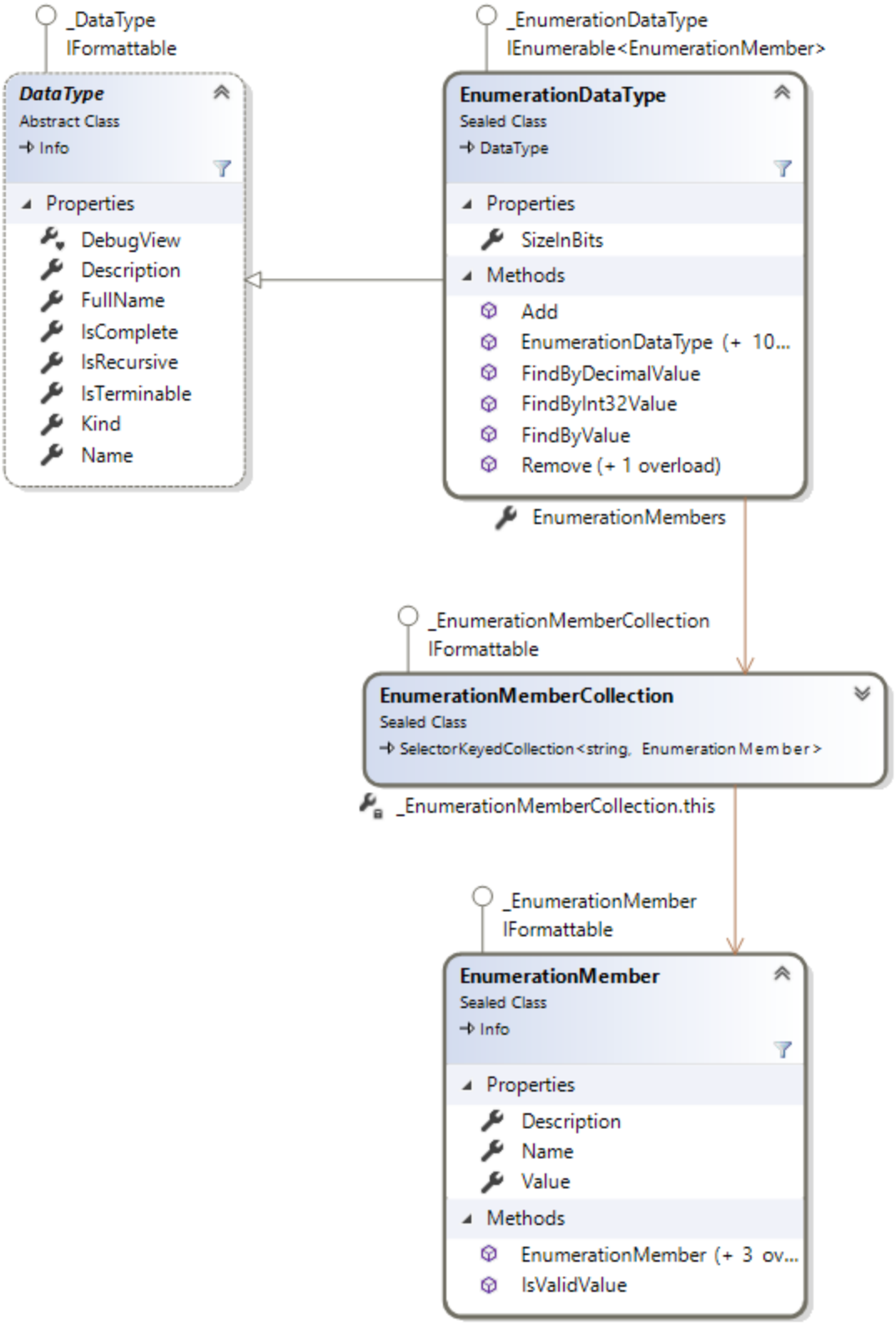
The SizeInBits Property determines how many bits the enumeration value can occupy. It is set to -1 when the size is unknown.

The EnumerationMembers Property contains a collection of enumeration members, i.e. descriptions of possible enumeration values. Each such description is an instance of EnumerationMember Class, and specifies:

- The Value: The actual integer value of the enumeration member.
- The Name: A short descriptive name of the enumeration member.
- The Description: Human readable description of the enumeration members.

Note that having the enumeration members in the EnumerationMembers Property is purely optional from the perspective of validation the generic data against the data type (see **Generic data validation (Section 7.2.1.2.1.4)**). Any enumeration value that first into the given size in bits (if specified) is valid.

The EnumerationDataType Class is an IEnumerable of EnumerationMember, and it has an Add Method for the same. It means that you can easily enumerate through its members, and it can also participate in collection initializers in C# or in VB.NET.



7.2.1.2.2.1.2 Opaque data type

The opaque data type is a data type whose concrete representation is hidden from its users. It is represented by an instance of the OpaqueDataType Class.

The SizeInBits Property determines how many bits the opaque value occupies. It equals to -1 when the size in unknown.

7.2.1.2.2.1.3 Primitive data type

The primitive data type is a basic or built-in type of the data type system. It is represented by an instance of the PrimitiveDataType Class.

The ValueType Property contains the .NET type of values represented by that instance. It can be, for example, System.Byte type.

The OPC UA Complex Data extension contains intrinsic data type dictionaries that map specific standard and commonly used data types to primitive data types. For details, refer to **OPC UA Data Type Systems (Section 7.2.1.3)**. You can therefore expect these primitive data types to appear in the complex data that you get from the server. Conversely, when you are sending some complex data to the server, your choice of primitive data types is limited by what the server supports.

7.2.1.2.2.1.4 Sequence data type

The sequence data type is a data type that represents a countable number of ordered values. It is represented by an instance of the SequenceDataType Class.

The ElementDataType Property contains the data type of the elements of the sequence.

The Length Property determines how many elements the sequence has. It is equals to -1 when the length is unknown (e.g. dynamic).

7.2.1.2.2.1.5 Structured data type

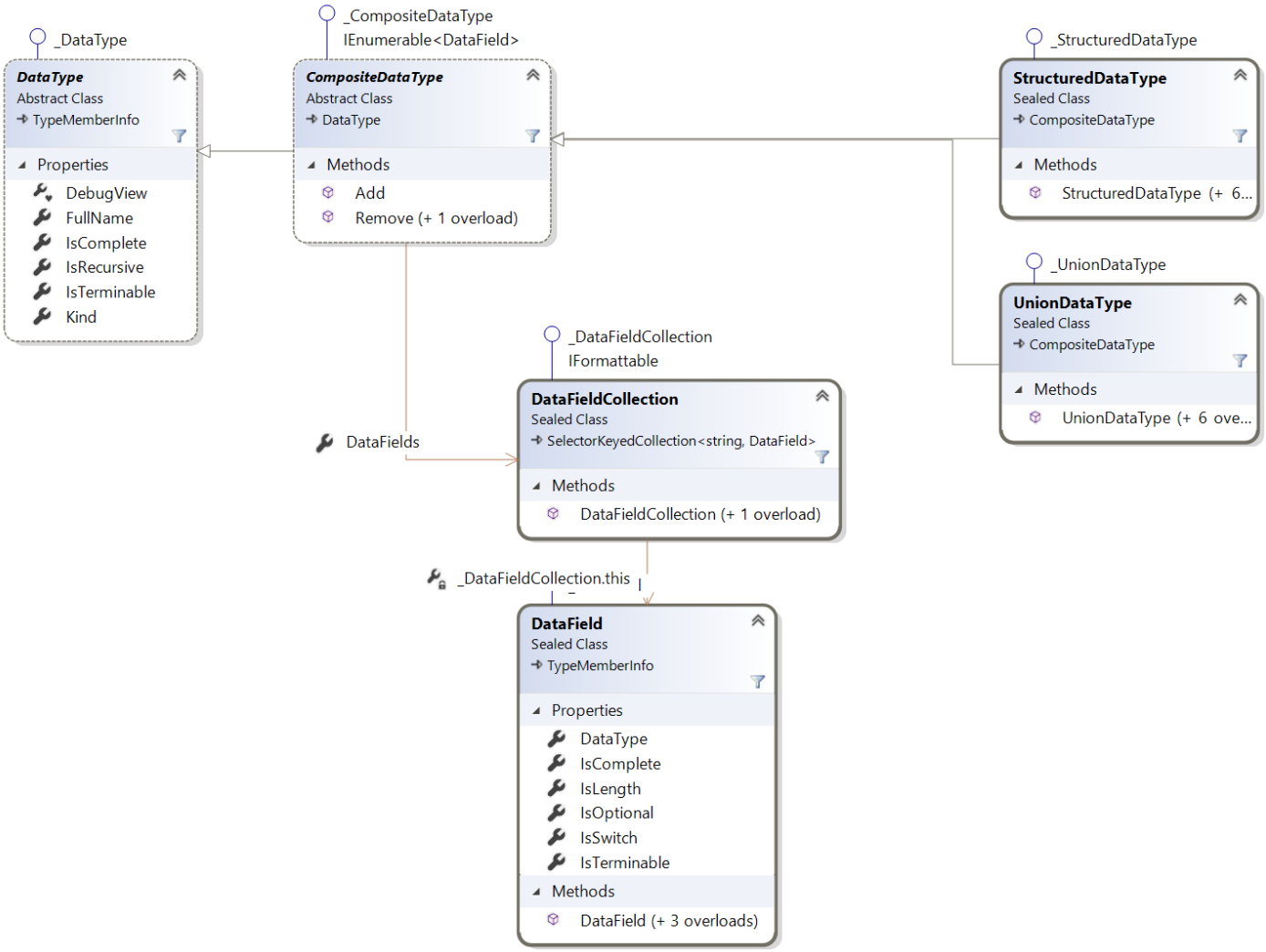
The structured data type is a record or a representation of union (in data dictionary model, i.e. the only model available before OPC UA 1.04). It is represented by an instance of the StructuredDataType Class.

The DataFields Property contains a collection of data fields of the structured data type. Each such data field is an instance of DataField Class, and specifies:

- A DataType of the contents of the field.
- A Name: Data field name, unique within the structured data type.
- A Description: Human readable description of the data field.
- An IsLength flag: Determines whether the field denotes a length of some sequence. Length fields may have special behavior e.g. with respect to **Generic data auto-complete (Section 7.2.1.3.1)**, and in validation of generic data (see **Generic data validation (Section 7.2.1.2.1.4)**).
- An IsOptional flag: Determines whether the field can be omitted from the encoded data. Optional fields play role role in determining whether a recursive data type is terminable (see **Working with data types (Section 7.2.1.2.2)**) and in validation of generic data (see **Generic data validation (Section 7.2.1.2.1.4)**).
- An IsSwitch flag: Determines whether the field acts as a switch for some other field. Switch fields may have special

behavior e.g. with respect to **Generic data auto-complete (Section 7.2.1.3.1)**.

The StructuredDataType Class is an IEnumerable of DataField, and it has an Add Method for the same. It means that you can easily enumerate through its data fields, and it can also participate in collection initializers in C# or in VB.NET.



7.2.1.2.2.1.6 Union data type

The union data type represents a (tagged) union.

> In data type dictionary model (the only model before OPC UA 1.04), union data is not directly available. A union can, however, be quite efficiently modeled using an appropriately formed structure with optional fields.

The DataFields Property contains a collection of data fields of the union data type. Each such data field is an instance of DataField Class, and specifies:

- A DataType of the contents of the field.
- A Name: Data field name, unique within the union data type.

- A Description: Human readable description of the data field.

7.2.1.2.2.2 Creating data types

You create data type objects simply by calling their constructors with appropriate arguments, and possibly filling in additional properties that were not passed into the constructors. For example:

C#

```
StructuredDataType structuredDataType = new StructuredDataType(
        "ControllerConfiguration",

"urn:UnifiedAutomation:CppDemoServer:BuildingAutomation:ControllerConfiguration")
    {
        new DataField("Name", UAOpcBinaryStandardDataTypes.CharArray),
        new DataField("DeviceAddress", UAOpcBinaryStandardDataTypes.UInt32),
        new DataField("TemperatureSetpoint", UAOpcBinaryStandardDataTypes.Double)
    };
```

Notice the use of pre-defined primitive types from the UAOpcBinaryStandardDataTypes Class. It is common to encounter references to standard data types from OPC Binary data type system, and the OPC UA Complex Data extension gives you this class so that you can refer to them easily.

7.2.1.2.2.3 Processing data types

In order to process a data type (a sub-type of DataType Class), you need to first determine its kind (see **Data type kinds (Section 7.2.1.2.2.1)**). You can do it by

- getting its Kind Property, and then having a switch or other kind of conditional statement, taking care of different data type kinds, or
- attempting to cast it (e.g. using the as operator in C#, or TryCast in VB.NET) to a concrete data type class (e.g. StructuredDataType), and testing whether it has succeeded.

There are also some common properties that all data types have, such as Name Property, FullName Property and Description Property.

After you have determined the kind of data type you are dealing with, you can use the data type object cast to appropriate type, and work with properties and methods that are specific to that data type kind. For example, a SequenceDataType contains the length of the sequence (the Length Property), and the data type of its elements (the ElementDataType Property).

The following example shows how a data type can be processed.

C#

```
// Shows how to process a data type, displaying some of its properties, recursively.

using System;
using System.Linq;
using OpcLabs.BaseLib.DataTypeModel;
```

```
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._DataType
{
    class Kind
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            UAGenericObject genericObject;
            try
            {
                genericObject = (UAGenericObject)client.ReadValue(endpointDescriptor,
nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
            // The data type is in the GenericData.DataType property of the
UAGenericObject.
            DataType dataType = genericObject.GenericData.DataType;

            // Process the data type. We will inspect some of its properties, and dump
them.
            ProcessDataType(dataType, maximumDepth: 2);
        }


        // Process the data type. It can be recursive in itself, so if you do not know
the data type you are dealing with,
        // it is recommended to make safeguards against infinite looping or recursion -
here, the maximumDepth.
        public static void ProcessDataType(DataType dataType, int maximumDepth)
        {
            if (maximumDepth == 0)
                return;

            Console.WriteLine();
```

```
            Console.WriteLine("dataType.Name: {0}", dataType.Name);

            switch (dataType.Kind)
            {
                case DataTypeKind.Enumeration:
                    Console.WriteLine("The data type is an enumeration.");
                    var enumerationDataType = (EnumerationDataType) dataType;
                    Console.WriteLine("It has {0} enumeration members.",
enumerationDataType.EnumerationMembers.Count);
                    Console.WriteLine("The names of the enumeration members are: {0}.",
                        String.Join(", ",
enumerationDataType.EnumerationMembers.Select(member => member.Name)));
                    // Here you can process the members, or inspect SizeInBits etc.
                    break;

                case DataTypeKind.Opaque:
                    Console.WriteLine("The data type is opaque.");
                    var opaqueDataType = (OpaqueDataType) dataType;
                    Console.WriteLine("Its size is {0} bits.",
opaqueDataType.SizeInBits);
                    // There isn't much more you can learn about an opaque data type
(well, it may have Description and
                    // other common members). It is, after all, opaque...
                    break;

                case DataTypeKind.Primitive:
                    Console.WriteLine("The data type is primitive.");
                    var primitiveDataType = (PrimitiveDataType) dataType;
                    Console.WriteLine("Its .NET value type is \"{0}\".",
primitiveDataType.ValueType);
                    // There isn't much more you can learn about the primitive data
type.
                    break;

                case DataTypeKind.Sequence:
                    Console.WriteLine("The data type is a sequence.");
                    var sequenceDataType = (SequenceDataType) dataType;
                    Console.WriteLine("Its length is {0} (-1 means that the length can
vary).", sequenceDataType.Length);

                    Console.WriteLine("A dump of the element data type follows.");
                    ProcessDataType(sequenceDataType.ElementDataType, maximumDepth -
1);
                    break;

                case DataTypeKind.Structured:
                    Console.WriteLine("The data type is structured.");
                    var structuredDataType = (StructuredDataType) dataType;
                    Console.WriteLine("It has {0} data fields.",
structuredDataType.DataFields.Count);
                    Console.WriteLine("The names of the data fields are: {0}.",
                        String.Join(", ", structuredDataType.DataFields.Select(field =>
field.Name)));

                    Console.WriteLine("A dump of each of the data fields follows.");
                    foreach (DataField dataField in structuredDataType.DataFields)
                    {
```

```
                        Console.WriteLine();
                        Console.WriteLine("dataField.Name: {0}", dataField.Name);
                        // Note that every data field also has properties like
IsLength, IsOptional, IsSwitch which might
                        // be of interest but we are not dumping them here.
                        ProcessDataType(dataField.DataType, maximumDepth - 1);
                    }
                    break;

                case DataTypeKind.Union:
                    Console.WriteLine("The data type is union.");
                    var unionDataType = (UnionDataType)dataType;
                    Console.WriteLine("It has {0} data fields.",
unionDataType.DataFields.Count);
                    Console.WriteLine("The names of the data fields are: {0}.",
                        String.Join(", ", unionDataType.DataFields.Select(field =>
field.Name)));
                    break;
            }
        }
    }
}
```

Object Pascal

```
// Shows how to process a data type, displaying some of its properties, recursively.

class procedure Kind.Main;
var
  Client: _EasyUAClient;
  DataType: OpcLabs_BaseLib_TLB._DataType;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node. We know that this node returns complex data, so we can type cast to
UAGenericObject.

  try
    GenericObject := IUnknown(Client.ReadValue(EndpointDescriptor, NodeDescriptor)) as
_UAGenericObject;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
```

```
  end;

  // The data type is in the GenericData.DataType property of the UAGenericObject.
  DataType := genericObject.GenericData.DataType;

  // Process the data type. We will inspect some of its properties, and dump them.
  ProcessDataType(DataType, 2);
end;

// Process the data type. It can be recursive in itself, so if you do not know the data
type you are dealing with,
// it is recommended to make safeguards against infinite looping or recursion - here,
the maximumDepth.
class procedure Kind.ProcessDataType(DataType: OpcLabs_BaseLib_TLB._DataType;
MaximumDepth: Cardinal);
var
  Count: Cardinal;
  DataField: _DataField;
  Element: OleVariant;
  ElementEnumerator: IEnumVARIANT;
  EnumerationMember: _EnumerationMember;
  EnumerationDataType: _EnumerationDataType;
  FieldNames: string;
  First: boolean;
  MemberNames: string;
  OpaqueDataType: _OpaqueDataType;
  PrimitiveDataType: _PrimitiveDataType;
  SequenceDataType: _SequenceDataType;
  StructuredDataType: _StructuredDataType;
  TypeName: WideString;
begin
  if MaximumDepth = 0 then
    Exit;

  WriteLn;
  WriteLn('dataType.Name: ', DataType.Name);

  case DataType.Kind of
    DataTypeKind_Enumeration:
      begin
        WriteLn('The data type is an enumeration.');
        EnumerationDataType := DataType as _EnumerationDataType;
        WriteLn(Format('It has %s enumeration members.',
[EnumerationDataType.EnumerationMembers.Count]));
        ElementEnumerator := EnumerationDataType.EnumerationMembers.GetEnumerator;
        MemberNames := '';
        First := True;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          EnumerationMember := IUnknown(Element) as _EnumerationMember;
          if First then
            First := False
          else
            MemberNames := MemberNames + ', ';
          MemberNames := MemberNames + EnumerationMember.Name;
        end;
        WriteLn(Format('The names of the enumeration members are: %s.',
```

```
[MemberNames]));
        // Here you can process the members, or inspect SizeInBits etc.
      end;
    DataTypeKind_Opaque:
      begin
        WriteLn('The data type is opaque.');
        OpaqueDataType := DataType as _OpaqueDataType;
        WriteLn(Format('Its size is %s bits.', [OpaqueDataType.SizeInBits]));
        // There isn't much more you can learn about an opaque data type (well, it may
have Description and
        // other common members). It is, after all, opaque...
      end;
    DataTypeKind_Primitive:
      begin
        WriteLn('The data type is primitive.');
        PrimitiveDataType := DataType as _PrimitiveDataType;
        PrimitiveDataType.ValueType.Get_ToString(TypeName);
        WriteLn(Format('Its .NET value type is "%s".', [TypeName]));
        // There isn't much more you can learn about the primitive data type.
      end;
    DataTypeKind_Sequence:
      begin
        WriteLn('The data type is a sequence.');
        SequenceDataType := DataType as _SequenceDataType;
        WriteLn(Format('Its length is %s (-1 means that the length can vary).',
[SequenceDataType.Length.ToString]));
        WriteLn('A dump of the element data type follows.');
        ProcessDataType(SequenceDataType.ElementDataType, MaximumDepth - 1);
      end;
    DataTypeKind_Structured:
      begin
        WriteLn('The data type is structured.');
        StructuredDataType := DataType as _StructuredDataType;
        WriteLn(Format('It has %s data fields.',
[StructuredDataType.DataFields.Count.ToString]));
        ElementEnumerator := StructuredDataType.DataFields.GetEnumerator;
        FieldNames := '';
        First := True;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          if First then
            First := False
          else
            FieldNames := FieldNames + ', ';
          FieldNames := FieldNames + Element.Name;
        end;
        WriteLn(Format('The names of the data fields are: %s.', [FieldNames]));

        WriteLn('A dump of each of the data fields follows.');
        ElementEnumerator := StructuredDataType.DataFields.GetEnumerator;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          DataField := IUnknown(Element) as _DataField;
          WriteLn;
          WriteLn(Format('dataField.Name: %s', [DataField.Name]));
          // Note that every data field also has properties like IsLength, IsOptional,
IsSwitch which might
```

```
          // be of interest but we are not dumping them here.
          ProcessDataType(DataField.DataType, MaximumDepth - 1);
        end;
    end;
  end;

end;
```

VB.NET

```
' Shows how to process a data type, displaying some of its properties, recursively.

Imports System
Imports System.Linq
Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._DataType

    Friend Class Kind

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            Dim genericObject As UAGenericObject
            Try
                genericObject = CType(client.ReadValue(endpointDescriptor,
nodeDescriptor), UAGenericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
            ' The data type is in the GenericData.DataType property of the
UAGenericObject.

            Dim dataType As DataType = genericObject.GenericData.DataType
```

```
            ' Process the data type. We will inspect some of its properties, and dump
them.
            ProcessDataType(dataType, maximumDepth:=2)
        End Sub


        ' Process the data type. It can be recursive in itself, so if you do not know
the data type you are dealing with,
        ' it is recommended to make safeguards against infinite looping or recursion -
here, the maximumDepth.
        Public Shared Sub ProcessDataType(dataType As DataType, ByVal maximumDepth As
Integer)
            If (maximumDepth = 0) Then
                Return
            End If

            Console.WriteLine()
            Console.WriteLine("dataType.Name: {0}", dataType.Name)

            Select Case (dataType.Kind)
                Case DataTypeKind.Enumeration
                    Console.WriteLine("The data type is an enumeration.")
                    Dim enumerationDataType = CType(dataType, EnumerationDataType)
                    Console.WriteLine("It has {0} enumeration members.",
enumerationDataType.EnumerationMembers.Count)
                    Console.WriteLine("The names of the enumeration members are: {0}.",
_
                                      String.Join(", ",
enumerationDataType.EnumerationMembers.Select(Function(member) member.Name)))
                    ' Here you can process the members, or inspect SizeInBits etc.

                Case DataTypeKind.Opaque
                    Console.WriteLine("The data type is opaque.")
                    Dim opaqueDataType = CType(dataType, OpaqueDataType)
                    Console.WriteLine("Its size is {0} bits.",
opaqueDataType.SizeInBits)
                    ' There isn't much more you can learn about an opaque data type
(well, it may have Description and
                    ' other common members). It is, after all, opaque...

                Case DataTypeKind.Primitive
                    Console.WriteLine("The data type is primitive.")
                    Dim primitiveDataType = CType(dataType, PrimitiveDataType)
                    Console.WriteLine("Its .NET value type is ""{0}"".",
primitiveDataType.ValueType)
                    ' There isn't much more you can learn about the primitive data
type.

                Case DataTypeKind.Sequence
                    Console.WriteLine("The data type is a sequence.")
                    Dim sequenceDataType = CType(dataType, SequenceDataType)
                    Console.WriteLine("Its length is {0} (-1 means that the length can
vary).", sequenceDataType.Length)
                    Console.WriteLine("A dump of the element data type follows.")
                    ProcessDataType(sequenceDataType.ElementDataType, (maximumDepth -
1))
```

```
                Case DataTypeKind.Structured
                    Console.WriteLine("The data type is structured.")
                    Dim structuredDataType = CType(dataType, StructuredDataType)
                    Console.WriteLine("It has {0} data fields.",
structuredDataType.DataFields.Count)
                    Console.WriteLine("The names of the data fields are: {0}.", _
                                      String.Join(", ",
structuredDataType.DataFields.Select(Function(field) field.Name)))
                    Console.WriteLine("A dump of each of the data fields follows.")

                    For Each dataField As DataField In structuredDataType.DataFields
                        Console.WriteLine()
                        Console.WriteLine("dataField.Name: {0}", dataField.Name)
                        ' Note that every data field also has properties like IsLength,
IsOptional, IsSwitch which might
                        ' be of interest but we are not dumping them here.
                        ProcessDataType(dataField.DataType, (maximumDepth - 1))
                    Next

                Case DataTypeKind.Union
                    Console.WriteLine("The data type is union.")
                    Dim unionDataType = CType(dataType, UnionDataType)
                    Console.WriteLine("It has {0} data fields.",
unionDataType.DataFields.Count)
                    Console.WriteLine("The names of the data fields are: {0}.",
                                      String.Join(", ",
unionDataType.DataFields.Select(Function(field) field.Name)))

            End Select

        End Sub
    End Class
End Namespace
```

In many cases, you will not be processing the data type alone, but in parallel with the generic data. See **Processing generic data (Section 7.2.1.2.1.3)** for details.

7.2.1.3 OPC UA Data Type Systems

The data type system determines how the data types are described, and how the data is encoded.

In This Topic

OPC Binary data type system

This data type system uses encoding name "Default Binary" (DefaultBinary in UABrowseNames Class).

The OPC Binary data type system is defined in the OPC UA specifications. It uses XML format to describe the data types, and the data is encoded in binary form. It is the most commonly used OPC UA data type system.

The OPC UA Complex Data extension can interpret data type dictionaries in the OPC Binary data type system, and encode

and decode custom types accordingly. In addition, it contains an intrinsic knowledge of the standard data type dictionary for this data type system, as defined by the OPC Foundation. The standard data types are mapped as follows:

OPC UA standard type	Data type (DataType Class) - raw	Data type (DataType Class) - CLS compliant
http://opcfoundation.org/BinarySchema/:Bit	primitive(System.Boolean)	primitive(System.Boolean)
http://opcfoundation.org/BinarySchema/:Boolean	primitive(System.Boolean)	primitive(System.Boolean)
http://opcfoundation.org/BinarySchema/:SByte	primitive(System.SByte)	primitive(System.Int16)
http://opcfoundation.org/BinarySchema/:Byte	primitive(System.Byte)	primitive(System.Byte)
http://opcfoundation.org/BinarySchema/:Int16	primitive(System.Int16)	primitive(System.Int16)
http://opcfoundation.org/BinarySchema/:UInt16	primitive(System.UInt16)	primitive(System.Int32)
http://opcfoundation.org/BinarySchema/:Int32	primitive(System.Int32)	primitive(System.Int32)
http://opcfoundation.org/BinarySchema/:UInt32	primitive(System.UInt32)	primitive(System.Int64)
http://opcfoundation.org/BinarySchema/:Int64	primitive(System.Int64)	primitive(System.Int64)
http://opcfoundation.org/BinarySchema/:UInt64	primitive(System.UInt64)	primitive(System.Decimal)
http://opcfoundation.org/BinarySchema/:Float	primitive(System.Single)	primitive(System.Single)
http://opcfoundation.org/BinarySchema/:Double	primitive(System.Double)	primitive(System.Double)
http://opcfoundation.org/BinarySchema/:Char	primitive(System.Char)	primitive(System.Char)
http://opcfoundation.org/BinarySchema/:String	primitive(System.String)	primitive(System.String)
http://opcfoundation.org/BinarySchema/:CharArray	primitive(System.String)	primitive(System.String)
http://opcfoundation.org/BinarySchema/:WideChar	primitive(System.Char)	primitive(System.Char)
http://opcfoundation.org/BinarySchema/:WideString	primitive(System.String)	primitive(System.String)
http://opcfoundation.org/BinarySchema/:WideCharArray	primitive(System.String)	primitive(System.String)
http://opcfoundation.org/BinarySchema/:ByteString	primitive(System.Byte[])	primitive(System.Byte[])
http://opcfoundation.org/BinarySchema/:DateTime	primitive(System.DateTime)	primitive(System.DateTime)
http://opcfoundation.org/BinarySchema/:Guid	primitive(System.Guid)	primitive(System.Guid)

When the OPC Binary data type system returns primitive data provided by the server to you, it makes sure that the type of the returned is CLS-compliant, and uses the data types listed under the "CLS compliant" column. For more information, see https://docs.microsoft.com/en-us/dotnet/standard/language-independence-and-language-independent-components#Types. Most types remain unchanged during this conversion, but values of the highlighted types undergo a conversion.

When sending primitive data types from the OPC Binary data type system to the server, the OPC UA Complex Data extension will attempt to convert the value provided by the caller to the .NET type listed in the above table (under the "raw" column). This means that you can freely write e.g. System.Int32 into http://opcfoundation.org/BinarySchema/:Byte, as long as the value in the required range (0-255 for System.Byte, in this example).

Standard types defined by the OPC Foundation in the specification have their corresponding DataType Class instances in

the UAOpcBinaryStandardDataTypes Class. You can use various properties of this class to refer to standard OPC UA data types.

XML Schema data type system

This data type system uses encoding name "Default XML" (DefaultXml in UABrowseNames Class).

The OPC UA Complex Data extension currently does not support the XML Schema data type system.

7.2.1.3.1 Generic data auto-complete

> ⚠ This article only applies to the data type dictionary model, i.e. the only complex type model available before OPC UA 1.04. It does not apply to the data type definition model.

In OPC Binary data type system (see **OPC UA Data Type Systems (Section 7.2.1.3)**), there can be length fields in a structure that determine length of an array that resides in some other field of the same structure.

The OPC UA Complex Data has an auto-complete functionality which allows you, in most cases, to omit the length fields from the StructuredData you pass to the component. With this feature, the component will use the actual length of the associated SequenceData field for the value of the length field, and the length value can be omitted, or (if present) will be ignored. This allows for easier coding and also provides robustness, because the encoded data must match the value of the length field, and the encoding would be incorrect if the value of the length fields differed from the actual length of the sequence.

The value of length field will **not** be ignored if you set it to -1. This is a special value that indicates that the associated sequence should not be present in the encoded data at all. Specifying -1 for the length always causes the associated sequence be omitted (its actual length is then irrelevant).

7.2.1.4 Advanced OPC UA Complex Data Tasks

Tasks described in this chapter are usually not necessary, but they might be needed in some more advanced scenarios.

Disabling the OPC UA Complex Data extension (Section 7.2.1.4.1): For licensing or performance reasons.

Configuring the OPC UA Complex Data extension and its parts (Section 7.2.1.4.2): E.g. in order to use shared data type model provider, or to work around errors in data type dictionaries.

Navigation in the OPC UA data model (Section 7.2.1.4.3): Access metadata about data types in OPC UA.

Adding or removing data type dictionaries to or from the OPC UA data type system (Section 7.2.1.4.4): Use a data type dictionary that is different from what resides in the OPC server, or is not present in the OPC server at all

The OPC UA Complex Data extension is, well, complex. This conceptual document cannot describe all its capabilities and internal workings in full. Remember that individual types and their members are documented in the reference documentation. If you have a task that is not covered in the documentation and you are unclear about how to resolve it, contact your vendor for technical support and advice.

7.2.1.4.1 Disabling the OPC UA Complex Data extension

The OPC UA Complex Data extension is enabled by default.

Disabling the Complex Data plug-in may be useful e.g. for licensing reasons (when the product edition you have does not support the Complex Data plug-in, and you want to avoid the associated error), or for performance reasons (if you do not need the internal content of the value, for example if your code just needs to take the value read, and write it elsewhere).

Enabling or disabling the OPC UA Complex Data extension is done by settings the Enabled Property in the PluginSetup for the extension. You can locate the PluginSetup in the PluginSetups Property of the InstanceParameters of EasyUAClient. In order to find the proper PluginSetup, call the FindName Method on the PluginSetupCollection Class, giving it "UAComplexData" string as an argument, or search the PluginSetups for a PluginSetup that has its UriString Property equal to EasyUAClientUriStrings.ComplexDataPlugin.

C#

```
// Shows how to disable and enable the OPC UA Complex Data plug-in.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._PluginSetup
{
    class Enabled
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue


            // We will explicitly disable the Complex Data plug-in, and read a node
which returns complex data. We will
            // receive an object of type UAExtensionObject, which contains the encoded
data in its binary form. In this
            // form, the data cannot be easily further processed by your application.
            //
            // Disabling the Complex Data plug-in may be useful e.g. for licensing
reasons (when the product edition you
            // have does not support the Complex Data plug-in, and you want to avoid
the associated error), or for
            // performance reasons (if you do not need the internal content of the
value, for example if your code just
            // needs to take the value read, and write it elsewhere).

            var client1 = new EasyUAClient();
            client1.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
```

```
false;

            object value1;
            try
            {
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine(value1);


            // Now we will read the same value, but with the Complex Data plug-in
enabled. This time we will receive an
            // object of type UAGenericObject, which contains the data in the decoded
form, accessible for further
            // processing by your application.
            //
            // Note that it is not necessary to explicitly enable the Complex Data
plug-in like this, because it is enabled
            // by default.

            var client2 = new EasyUAClient();
            client2.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
true;

            object value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor);
            Console.WriteLine(value2);


            // Example output:
            //
            // Binary Byte[1373]; {nsu=http://test.org/UA/Data/;i=11437}
            // (ScalarValueDataType) structured

            // On the first line, the type and length of the encoded data is shown, and
the node ID is the encoding ID.
            // On the 2nd line, the kind (structured) and the name of the complex data
type (ScalarValueDataType) is shown.
        }
    }
}
```

Object Pascal

```
// Shows how to disable and enable the OPC UA Complex Data plug-in.

class procedure Enabled.Main;
var
  Client1, Client2: _EasyUAClient;
  EndpointDescriptor: string;
  NodeDescriptor: string;
  Value1, Value2: OleVariant;
```

```
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // We will explicitly disable the Complex Data plug-in, and read a node which returns
complex data. We will
  // receive an object of type UAExtensionObject, which contains the encoded data in
its binary form. In this
  // form, the data cannot be easily further processed by your application.
  //
  // Disabling the Complex Data plug-in may be useful e.g. for licensing reasons (when
the product edition you
  // have does not support the Complex Data plug-in, and you want to avoid the
associated error), or for
  // performance reasons (if you do not need the internal content of the value, for
example if your code just
  // needs to take the value read, and write it elsewhere).
  Client1 := CoEasyUAClient.Create;
  Client1.InstanceParameters.PluginSetups.FindName('UAComplexData').Enabled := false;

  try
    Value1 := Client1.ReadValue(EndpointDescriptor, NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
  WriteLn(Value1);

  // Now we will read the same value, but with the Complex Data plug-in enabled. This
time we will receive an
  // object of type UAGenericObject, which contains the data in the decoded form,
accessible for further
  // processing by your application.
  //
  // Note that it is not necessary to explicitly enable the Complex Data plug-in like
this, because it is enabled
  // by default.

  Client2 := CoEasyUAClient.Create;
  Client2.InstanceParameters.PluginSetups.FindName('UAComplexData').Enabled := true;

  Value2 := Client2.ReadValue(EndpointDescriptor, NodeDescriptor);
  WriteLn(Value2);

  // Example output:
  //
  // Binary Byte[1373]; {nsu=http://test.org/UA/Data/;i=11437}
  // (ScalarValueDataType) structured
```

```
  // On the first line, the type and length of the encoded data is shown, and the node
ID is the encoding ID.
  // On the 2nd line, the kind (structured) and the name of the complex data type
(ScalarValueDataType) is shown.

end;
```

VB.NET

```
' Shows how to disable and enable the OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._PluginSetup

    Friend Class Enabled

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue


            ' We will explicitly disable the Complex Data plug-in, and read a node
which returns complex data. We will
            ' receive an object of type UAExtensionObject, which contains the encoded
data in its binary form. In this
            ' form, the data cannot be easily further processed by your application.
            '
            ' Disabling the Complex Data plug-in may be useful e.g. for licensing
reasons (when the product edition you
            ' have does not support the Complex Data plug-in, and you want to avoid the
associated error), or for
            ' performance reasons (if you do not need the internal content of the
value, for example if your code just
            ' needs to take the value read, and write it elsewhere).

            Dim client1 = New EasyUAClient
            client1.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
False

            Dim value1 As Object
            Try
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
```

```
uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value1)


            ' Now we will read the same value, but with the Complex Data plug-in
enabled. This time we will receive an
            ' object of type UAGenericObject, which contains the data in the decoded
form, accessible for further
            ' processing by your application.
            '
            ' Note that it is not necessary to explicitly enable the Complex Data plug-
in like this, because it is enabled
            ' by default.

            Dim client2 = New EasyUAClient
            client2.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
True

            Dim value2 As Object
            Try
                value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value2)


            ' Example output:
            '
            ' Binary Byte[1373]; {nsu=http://test.org/UA/Data/;i=11437}
            ' (ScalarValueDataType) structured

            ' On the first line, the type and length of the encoded data is shown, and
the node ID is the encoding ID.
            ' On the 2nd line, the kind (structured) and the name of the complex data
type (ScalarValueDataType) is shown.
        End Sub
    End Class
End Namespace
```

When the OPC UA Complex Data extension is disabled, the component

- returns instances of UAExtensionObject Class when nodes with complex data are read or subscribed to,
- expects instances of UAExtensionObject Class when nodes with complex data are written to,
- does not allow you to obtain services related to OPC UA Complex Data (such as IEasyUAClientComplexData Interface from the instance of the EasyUAClient Class using the GetService Method.

When the OPC UA Complex Data extension is enabled (the default) but the current license does not allow the OPC UA Complex Data feature, reads, subscriptions and writes, or attempts to obtain services related to OPC UA Complex Data,

will return an error.

> ⚠ Enabling or disabling the extension only has effect when done before you attempt first OPC operation or GetService Method call on the object instance.

7.2.1.4.2 Configuring the OPC UA Complex Data extension and its parts

There are various parameters that can be configured on the OPC UA Complex Data extension. Take following steps to read or modify them:

1. Access the InstanceParameters Property of your EasyUAClient Class instance.
2. In the InstanceParameters Property, access the PluginConfigurations Property.
3. Call the Find Method of the ConfigurationPartCollection Class, passing it the name of the type of the configuration part you want to modify. The available type names and their purposes are listed further below.
4. Read or modify the properties of the obtained object as needed.

Some typical scenarios for which the configuration might be needed are described below.

> ⚠ Configuring the extension only has effect when done before you attempt first OPC operation or GetService Method call on the object instance.

In This Topic

Using shared data type model provider

The data type model provider (DTMP) is an object that provides access to the metadata about data type in the target OPC server(s). The data type model provider also caches the metadata obtained, so that repeated decodings of the same data types do not require retrieving the metadata from the OPC server(s) again and again.

By default, each instance of the EasyUAClient Class has its own data type model provider. If you use multiple instances of the EasyUAClient Class and do not require them be fully isolated in this respect, you should consider switching to a shared data type model provider.

Conceptually, there is no noticeable difference in the results from the default state in which the client objects are set to use per-instance data type model provider. But, with the shared data type model provider, the metadata obtained during the operations on one client object and cached inside the data type model provider are reused during operations on other client objects, making this and the subsequent operations more efficient.

The configuration part that you need to access is the UAComplexDataPluginParameters Class.

C#

```
// Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.Plugins.ComplexData;

namespace UADocExamples.ComplexData._UAComplexDataPluginParameters
{
    class IsolatedDataTypeModelProvider
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; // [ObjectsFolder]/Data.Static.Scalar.StructureValue


            // We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
            // model provider.

            // Configure the first client object.
            var client1 = new EasyUAClient();
            UAComplexDataPluginParameters complexDataPluginParameters1 =
                client1.InstanceParameters.PluginConfigurations.Find<UAComplexDataPluginParameters>();
            Debug.Assert(complexDataPluginParameters1 != null);
            complexDataPluginParameters1.IsolatedDataTypeModelProvider = false;

            // Configure the second client object.
            var client2 = new EasyUAClient();
            UAComplexDataPluginParameters complexDataPluginParameters2 =
                client2.InstanceParameters.PluginConfigurations.Find<UAComplexDataPluginParameters>();
```

```
            Debug.Assert(complexDataPluginParameters2 != null);
            complexDataPluginParameters2.IsolatedDataTypeModelProvider = false;


            // We will now read the same complex data node using the two client objects.
            //
            // There is no noticeable difference in the results from the default state in which the client objects are
            // set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
            // obtained during the read on the first client object and cached inside the data type model provider are reused
            // during the read on the second client object, making this and the subsequent operations more efficient.

            // Read the complex data node using the first client.
            object value1;
            try
            {
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine(value1);

            // Read the complex data node using the second client.
            object value2;
            try
            {
                value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine(value2);
        }
    }
}
```

Object Pascal

```
// Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

class procedure IsolatedDataTypeModelProvider.Main;
var
  Client1, Client2: _EasyUAClient;
  ComplexDataPluginParameters1, ComplexDataPluginParameters2: OpcLabs_EasyOpcUA_TLB._UAComplexDataPluginParameters;
  EndpointDescriptor: string;
  NodeDescriptor: string;
  Value1, Value2: OleVariant;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  // [ObjectsFolder]/Data.Static.Scalar.StructureValue

  // We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
  // model provider.

  // Configure the first client object.
  Client1 := CoEasyUAClient.Create;
  ComplexDataPluginParameters1 :=
IUnknown(Client1.InstanceParameters.PluginConfigurations.Find('OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters'))
as OpcLabs_EasyOpcUA_TLB._UAComplexDataPluginParameters;
  ComplexDataPluginParameters1.IsolatedDataTypeModelProvider := false;

  // Configure the second client object.
  Client2 := CoEasyUAClient.Create;
  ComplexDataPluginParameters2 :=
IUnknown(Client2.InstanceParameters.PluginConfigurations.Find('OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters'))
as OpcLabs_EasyOpcUA_TLB._UAComplexDataPluginParameters;
  ComplexDataPluginParameters2.IsolatedDataTypeModelProvider := false;

  // We will now read the same complex data node using the two client objects.
  //
  // There is no noticeable difference in the results from the default state in which the client objects are
  // set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
  // obtained during the read on the first client object and cached inside the data type model provider are reused
  // during the read on the second client object, making this and the subsequent operations more efficient.

  // Read the complex data node using the first client.
  try
    Value1 := Client1.ReadValue(EndpointDescriptor, NodeDescriptor);
```

```
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
  WriteLn(Value1);

  // Read the complex data node using the second client.
  try
    Value2 := Client2.ReadValue(EndpointDescriptor, NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
  WriteLn(Value2);

end;
```

VB.NET

```
' Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.Plugins.ComplexData

Namespace UADocExamples.ComplexData._UAComplexDataPluginParameters

    Friend Class IsolatedDataTypeModelProvider

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue


            ' We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
            ' model provider.

            ' Configure the first client object.
            Dim client1 = New EasyUAClient
            Dim complexDataPluginParameters1 As UAComplexDataPluginParameters = _
                client1.InstanceParameters.PluginConfigurations.Find(Of UAComplexDataPluginParameters)()
            Debug.Assert(complexDataPluginParameters1 IsNot Nothing)
            complexDataPluginParameters1.IsolatedDataTypeModelProvider = False

            ' Configure the second client object.
            Dim client2 = New EasyUAClient
            Dim complexDataPluginParameters2 As UAComplexDataPluginParameters = _
                client2.InstanceParameters.PluginConfigurations.Find(Of UAComplexDataPluginParameters)()
            Debug.Assert(complexDataPluginParameters2 IsNot Nothing)
            complexDataPluginParameters2.IsolatedDataTypeModelProvider = False


            ' We will now read the same complex data node using the two client objects.
            '
            ' There is no noticeable difference in the results from the default state in which the client objects are
            ' set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
            ' obtained during the read on the first client object and cached inside the data type model provider are reused
            ' during the read on the second client object, making this and the subsequent operations more efficient.

            ' Read the complex data node using the first client.
            Dim value1 As Object
            Try
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value1)

            ' Read the complex data node using the second client.
            Dim value2 As Object
```

```
            Try
                value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value2)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

Option Explicit

' Define which server and node we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
Dim nodeDescriptor: nodeDescriptor = _
    "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue


' We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
' model provider.

' Configure the first client object.
Dim Client1: Set Client1 = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ComplexDataPluginParameters1: Set ComplexDataPluginParameters1 = Client1.InstanceParameters.PluginConfigurations.Find( _
    "OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters")
ComplexDataPluginParameters1.IsolatedDataTypeModelProvider = False

' Configure the second client object.
Dim Client2: Set Client2 = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ComplexDataPluginParameters2: Set ComplexDataPluginParameters2 = Client2.InstanceParameters.PluginConfigurations.Find( _
    "OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters")
ComplexDataPluginParameters2.IsolatedDataTypeModelProvider = False

' We will now read the same complex data node using the two client objects.
'
' There is no noticeable difference in the results from the default state in which the client objects are
' set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
' obtained during the read on the first client object and cached inside the data type model provider are reused
' during the read on the second client object, making this and the subsequent operations more efficient.

' Read the complex data node using the first client.
On Error Resume Next
Dim Value1: Set Value1 = Client1.ReadValue(endpointDescriptor, nodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo Value1

' Read the complex data node using the second client.
On Error Resume Next
Dim Value2: Set Value2 = Client2.ReadValue(endpointDescriptor, nodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo Value2
```

Configuring data type dictionary locations

In OPC Binary data type system, data type dictionaries that reside in the OPC server may use types from other data type dictionaries, and optionally specify location of these other dictionaries, using a string. The OPC UA specification does not define a format for such string.

The OPC UA Complex Data extension already contains data the standard data type dictionary defined in the OPC UA specification, and some others. It also knows how to resolve their namespaces to locations, therefore (by default), the data type dictionaries in the server do not have to specify the location for them.

For other namespaces, the OPC Binary data type system recognizes following syntax for the location string:

- If the string can be parsed as an OPC UA Node ID with explicitly specified namespace (by URI or namespace index), it is understood to be a reference to a node in the same OPC server, and resolved as such.
- Otherwise, the location string is understood to be a URI.

If the location string is understood to be a URI, it can be only:

- A URI with special "net.res" URI scheme, referring to a managed (.NET) resource embedded in some .NET assembly. The syntax of this URI is "net.res://*assemblyName/resourceName*".
- A file URI, or a universal naming convention (UNC) path.

For import directives that do not have their location string specified, the OPC Binary data type system tries to find the location in the WellKnownLocationTemplateDictionary Property. You can therefore access the UAOpcBinaryDtsParameters Class configuration part, and add your entry (entries) to this dictionary, in order to tell the OPC Binary data type system where to find the data type dictionaries. You can specify a node ID in the server if the data type dictionary resides there, or place the data type dictionary into a file, or embed it in a managed resource in your program.

Working around errors in data type dictionaries

The OPC UA Complex Data is commonly implemented with some bugs on the server side. In addition, at time of writing this article, it is not rigorously (or at all) checked during OPC compliance certification process, and only to very limited extent during OPC Interoperability Workshops. The client must therefore be prepared to handle situations in which the server behavior is erroneous.

We have tested the OPC UA Complex Data extension against several servers with complex data support available, and continue to expand the range of interoperable servers. We have found several issues in these servers (and in the OPC UA specifications) during these tests, and have already provided special coding and configuration inside the OPC UA Complex Data extension to work around these issues.

If you encounter a buggy server and the workaround is not already built into the OPC UA Complex Data extension, you might be able to work around the problem by configuring the extension accordingly. You are also welcome to report the problem to us, so that we can adjust the software and prevent the issue in future versions.

For errors related to the OPC Binary data type system, the configuration part you need to access is the UAOpcBinaryDtsParameters Class.

Here are some common issues with OPC Binary data type system, and their possible workarounds:

- The data type dictionary uses some external types, but fails to list the namespace (and possibly location) in the import directives. To rectify this, add the import directive to ImplicitImportDictionary Property.
- The data type dictionary uses incorrect (e.g. misspelled) namespace in some of its type references. To rectify this, redirect the namespace by adding an entry to the NamespaceRedirectionDictionary Property.
- The data type dictionary uses some incorrect (e.g. misspelled) type name, or the type name is correct but the type definition in the dictionary does not match the actual encoding used "on the wire". To rectify this, add an appropriate entry to TypeNameRemappingDictionary Property.

7.2.1.4.3 Navigation in the OPC UA data model

For accessing the metadata about data types in OPC UA and navigating in it, the OPC UA Complex Data extension gives you

- An OPC UA data type model provider (DTMP) , represented by the IUADataTypeModelProvider Interface.
- An OPC UA data type dictionary provider, represented by the IUADataTypeDictionaryProvider Interface, and used with the data type dictionary model (the only complex data model available before OPC UA 1.04).
- An OPC UA data type definition provider, represented by the IUADataTypeDefinitionProvider Interface, and used with the data type definition model (the complex data model available since OPC UA 1.04).

The providers retrieve the information from the OPC server in various ways (by reading and browsing the metadata nodes). In addition, they cache the information obtained. This is very important for performance reasons, because the same metadata can be reused without having to retrieve them from the server again and again.

In This Topic

OPC UA data type model provider

Take following steps to get access to the OPC UA data type model provider:

1. Get the IEasyUAClientComplexData Interface service from your EasyUAClient Class instance using the GetService Method.
2. Using the DataTypeModelProvider Property, obtain the IUADataTypeModelProvider Interface.

When you have the IUADataTypeModelProvider Interface, you can call its methods to:

- Get data type IDs from given data type encoding IDs (GetDataTypeIdsFromDataTypeEncodingIds Method).
- Get data type IDs from given model node descriptors (GetDataTypeIdsFromModelNodeDescriptors Method).
- Get encoding node element collections from given data type IDs (GetEncodingNodeElementCollectionsFromDataTypeIds Method).

The basic methods on the IUADataTypeModelProvider Interface provide just one navigation "hop" at a time. Some of these "hops" are, however, commonly needed together, in a chain, possibly with some kind of additional simple processing. For this purpose, there are extension methods available (in

the IUADataTypeModelProviderExtension Class) that make such tasks easier. With help of these methods, you can then, for example:

- Resolve encoding IDs from given data type IDs and an encoding name.
- Resolve encoding IDs from given model node descriptors and an encoding name.

If you have more pieces of information to work on in the same way, it is much more efficient to process them together, in one operation. All methods on the IUADataTypeModelProvider Interface and the IUADataTypeModelProviderExtension Class therefore accept a read-only list on input, and return an array of processed element on the output. If performance is not a concern, or if you have just one piece of information to work on at a time, there are extension methods to make it easier as well. For example, instead of calling GetDataTypeIdsFromModelNodeDescriptors Method for just one node, you can use the GetDataTypeIdFromModelNodeDescriptor Method instead.

OPC UA data type dictionary provider

> ⚠ This chapter only applies to the data type dictionary model, i.e. the only complex type model available before OPC UA 1.04. It does not apply to the data type definition model.

Take following steps to get access to the OPC UA data type dictionary provider:

1. Get the IEasyUAClientComplexData Interface service from your EasyUAClient Class instance using the GetService Method.
2. Using the DataTypeDictionaryProvider Property, obtain the IUADataTypeDictionaryProvider Interface.

When you have the IUADataTypeDictionaryProvider Interface, you can call its methods to:

- Get data type description IDs from given data type encoding IDs (GetDataTypeDescriptionIdsFromDataTypeEncodingIds Method).
- Get data type descriptions from given data type description IDs (GetDataTypeDescriptionsFromDataTypeDescriptionIds Method).
- Get data type dictionaries from given data type dictionary IDs (GetDataTypeDictionariesFromDataTypeDictionaryIds Method).
- Get data type dictionary IDs from given data type description IDs (GetDataTypeDictionaryIdsFromDataTypeDescriptionIds Method).

The basic methods on the IUADataTypeDictionaryProvider Interface provide just one navigation "hop" at a time. Some of these "hops" are, however, commonly needed together, in a chain, possibly with some kind of additional simple processing. For this purpose, there are extension methods available (in the IUADataTypeDictionaryProviderExtension Class) that make such tasks easier. With help of these methods, you can then, for example:

- Resolve data type descriptors from given data type encoding IDs.

If you have more pieces of information to work on in the same way, it is much more efficient to process them together, in one operation. All methods on the IUADataTypeDictionaryProvider Interface and the IUADataTypeDictionaryProviderExtension Class therefore accept a read-only list on input, and return an array of processed element on the output. If performance is not a concern, or if you have just one piece of information to work on at a time, there are extension methods to make it easier as well.

OPC UA data type definition provider

> ⚠ This chapter only applies to the data type definition model, i.e. the complex type model available since OPC UA 1.04. It does not apply to the data type description model.

Take following steps to get access to the OPC UA data type dictionary provider:

1. Get the IEasyUAClientComplexData Interface service from your EasyUAClient Class instance using the GetService Method.
2. Using the DataTypeDefinitionProvider Property, obtain the IUADataTypeDefinitionProvider Interface.

When you have the IUADataTypeDefinitionProvider Interface, you can call its methods to:

- Get data type infos from given data type IDs (GetDataTypeInfosFromDataTypeIds Method). The data type info is represented by an instance of the UADataTypeInfo Class, and provides metadata for the OPC UA custom data type in the data type definition model.

If you have more pieces of information to work on in the same way, it is much more efficient to process them together, in one operation. All methods on the IUADataTypeDictionaryProvider Interface therefore accept a read-only list on input, and return an array of processed element on the output. If performance is not a concern, or if you have just one piece of information to work on at a time, there are extension methods in the IUADataTypeDefinitionProviderExtension Class to make it easier as well.

Resolving the data type

When you receive complex data in the GenericData Property of a UAGenericObject, it has its DataType Property filled in with information about the associated data type, so that you can process it as you need. But what you have not yet received the complex data from the server - maybe you are just preparing something to be written, and you want to inspect what is the data type structure for certain node or data type ID? In this case, you can use the

methods on the IUADataTypeModelProvider Interface to obtain an instance of DataType Class.

The example below shows how to resolve a data type. The main work is done by the ResolveDataType Method.

C#

```
// Shows how to obtain object describing the data type of complex data node with OPC UA Complex Data plug-in.

using System;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.OperationModel.Generic;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.InformationModel;
using OpcLabs.EasyOpc.UA.Plugins.ComplexData;

namespace UADocExamples.ComplexData._IEasyUAClientComplexData
{
    class ResolveDataType
    {
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Obtain the data type ID.
            //
            // In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
            // attribute, or easier, by calling the extension method ReadDataType on the IEasyUAClient interface.
            // The sample server, however, shows a more advanced approach in which the data type ID refers to an
abstract
            // data type, and the actual values are then sub-types of this base data type. This abstract data
type does not
            // have any encodings associated with it and it is therefore not possible to extract its description
from the
            // server. We therefore use a hard-coded data type ID for one of the sub-types in this example.
            //
            // The code to obtain the data type ID for given node would normally look like this:
            //    UANodeId dataTypeId = client.ReadDataType(
            //        endpointDescriptor,
            //        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            //
            UANodeId dataTypeId = "nsu=http://test.org/UA/Data/;i=9440";    // ScalarValueDataType

            // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
            // operations.
            IEasyUAClientComplexData complexData = client.GetService<IEasyUAClientComplexData>();

            // Resolve the data type ID to the data type object, containing description of the data type.
            ValueResult<DataType> dataTypeResult = complexData.ResolveDataType(
                new UAModelNodeDescriptor(endpointDescriptor, dataTypeId),
                UABrowseNames.DefaultBinary);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!dataTypeResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", dataTypeResult.ErrorMessageBrief);
                return;
            }

            // The actual data type is in the Value property.
            // Display basic information about what we have obtained.
            Console.WriteLine(dataTypeResult.Value);
```

```
            // If we want to see the whole hierarchy of the received data type, we can format it with the "V"
(verbose)
            // specifier. In the debugger, you can view the same by displaying the private DebugView property.
            Console.WriteLine();
            Console.WriteLine("{0:V}", dataTypeResult.Value);

            // For processing the internals of the data type, refer to examples for GenericData class.


            // Example output (truncated):
            //
            //ScalarValueDataType = structured
            //
            //ScalarValueDataType = structured
            //  [BooleanValue] Boolean = primitive(System.Boolean)
            //  [ByteStringValue] ByteString = primitive(System.Byte[])
            //  [ByteValue] Byte = primitive(System.Byte)
            //  [DateTimeValue] DateTime = primitive(System.DateTime)
            //  [DoubleValue] Double = primitive(System.Double)
            //  [EnumerationValue] Int32 = primitive(System.Int32)
            //  [ExpandedNodeIdValue] ExpandedNodeId = structured
            //    [ByteString] optional ByteStringNodeId = structured
            //      [Identifier] ByteString = primitive(System.Byte[])
            //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //    [FourByte] optional FourByteNodeId = structured
            //      [Identifier] UInt16 = primitive(System.UInt16)
            //      [NamespaceIndex] Byte = primitive(System.Byte)
            //    [Guid] optional GuidNodeId = structured
            //      [Identifier] Guid = primitive(System.Guid)
            //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //    [NamespaceURI] optional CharArray = primitive(System.String)
            //    [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
            //    [NodeIdType] switch NodeIdType = enumeration(6)
            //      TwoByte = 0
            //      FourByte = 1
            //      Numeric = 2
            //      String = 3
            //      Guid = 4
            //      ByteString = 5
            //    [Numeric] optional NumericNodeId = structured
            //      [Identifier] UInt32 = primitive(System.UInt32)
            //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //    [ServerIndex] optional UInt32 = primitive(System.UInt32)
            //    [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
            //    [String] optional StringNodeId = structured
            //      [Identifier] CharArray = primitive(System.String)
            //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //    [TwoByte] optional TwoByteNodeId = structured
            //      [Identifier] Byte = primitive(System.Byte)
            //  [FloatValue] Float = primitive(System.Single)
            //  [GuidValue] Guid = primitive(System.Guid)
            //  [Int16Value] Int16 = primitive(System.Int16)
            //  [Int32Value] Int32 = primitive(System.Int32)
            //  [Int64Value] Int64 = primitive(System.Int64)
            //  [Integer] Variant = structured
            //    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
            //    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            //    [ArrayLength] length optional Int32 = primitive(System.Int32)
            //    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            //    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
            //    [Byte] optional sequence[*] of Byte = primitive(System.Byte)
        }
    }
}
```

Object Pascal

```
// Shows how to obtain object describing the data type of complex data node with OPC UA Complex Data plug-in.

class procedure ResolveDataType.Main;
var
  Client: _EasyUAClient;
```

```
  ComplexData: _EasyUAClientComplexData;
  DataType: _DataType;
  DataTypeId: string;
  DataTypeResult: _ValueResult;
  EncodingName: _UAQualifiedName;
  EndpointDescriptor: string;
  ModelNodeDescriptor: _UAModelNodeDescriptor;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
    //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
    //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain the data type ID.
  //
  // In many cases, you would be able to obtain the data type ID of a particular node by reading its DataType
  // attribute.
  // The sample server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type, and
  // the actual values are then sub-types of this base data type. This abstract data type does not have any
encodings
  // associated with it and it is therefore not possible to extract its description from the server. We therefore
use
  // a hard-coded data type ID for one of the sub-types in this example.
  DataTypeId := 'nsu=http://test.org/UA/Data/;i=9440';    // ScalarValueDataType

  // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
  // operations.
  ComplexData :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData,
OpcLabs.EasyOpcUA')) as _EasyUAClientComplexData;

  // Resolve the data type ID to the data type object, containing description of the data type.
  ModelNodeDescriptor := CoUAModelNodeDescriptor.Create;
  ModelNodeDescriptor.EndpointDescriptor.UrlString := EndpointDescriptor;
  ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText := DataTypeId;
  EncodingName := CoUAQualifiedName.Create;
  EncodingName.StandardName := 'DefaultBinary';
  DataTypeResult := ComplexData.ResolveDataType(ModelNodeDescriptor, EncodingName);
  if not DataTypeResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [DataTypeResult.ErrorMessageBrief]));
    Exit;
  end;

  // The actual data type is in the Value property.
  DataType := IUnknown(DataTypeResult.Value) as _DataType;

  // Display basic information about what we have obtained.
  WriteLn(DataType.ToString);

  // If we want to see the whole hierarchy of the received data type, we can format it with the "V" (verbose)
  // specifier.
  Writeln;
  Writeln(DataType.ToString_2['V', nil]);

  // For processing the internals of the data type, refer to examples for GenericData class.


  // Example output (truncated):
  //
  //ScalarValueDataType = structured
  //
  //ScalarValueDataType = structured
  //  [BooleanValue] Boolean = primitive(System.Boolean)
  //  [ByteStringValue] ByteString = primitive(System.Byte[])
  //  [ByteValue] Byte = primitive(System.Byte)
  //  [DateTimeValue] DateTime = primitive(System.DateTime)
  //  [DoubleValue] Double = primitive(System.Double)
```

```
  //  [EnumerationValue] Int32 = primitive(System.Int32)
  //  [ExpandedNodeIdValue] ExpandedNodeId = structured
  //    [ByteString] optional ByteStringNodeId = structured
  //      [Identifier] ByteString = primitive(System.Byte[])
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [FourByte] optional FourByteNodeId = structured
  //      [Identifier] UInt16 = primitive(System.UInt16)
  //      [NamespaceIndex] Byte = primitive(System.Byte)
  //    [Guid] optional GuidNodeId = structured
  //      [Identifier] Guid = primitive(System.Guid)
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [NamespaceURI] optional CharArray = primitive(System.String)
  //    [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
  //    [NodeIdType] switch NodeIdType = enumeration(6)
  //      TwoByte = 0
  //      FourByte = 1
  //      Numeric = 2
  //      String = 3
  //      Guid = 4
  //      ByteString = 5
  //    [Numeric] optional NumericNodeId = structured
  //      [Identifier] UInt32 = primitive(System.UInt32)
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [ServerIndex] optional UInt32 = primitive(System.UInt32)
  //    [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
  //    [String] optional StringNodeId = structured
  //      [Identifier] CharArray = primitive(System.String)
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [TwoByte] optional TwoByteNodeId = structured
  //      [Identifier] Byte = primitive(System.Byte)
  //  [FloatValue] Float = primitive(System.Single)
  //  [GuidValue] Guid = primitive(System.Guid)
  //  [Int16Value] Int16 = primitive(System.Int16)
  //  [Int32Value] Int32 = primitive(System.Int32)
  //  [Int64Value] Int64 = primitive(System.Int64)
  //  [Integer] Variant = structured
  //    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
  //    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
  //    [ArrayLength] length optional Int32 = primitive(System.Int32)
  //    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
  //    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
  //    [Byte] optional sequence[*] of Byte = primitive(System.Byte)

end;
```

VB.NET

```
' Shows how to obtain object describing the data type of complex data node with OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.OperationModel.Generic
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.InformationModel
Imports OpcLabs.EasyOpc.UA.Plugins.ComplexData

Namespace UADocExamples.ComplexData._IEasyUAClientComplexData

    Friend Class ResolveDataType

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object.
            Dim client = New EasyUAClient
```

```
            ' Obtain the data type ID.
            '
            ' In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
            ' attribute, or easier, by calling the extension method ReadDataType on the IEasyUAClient interface.
The sample
            ' server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type,
            ' and the actual values are then sub-types of this base data type. This abstract data type does not
have any
            ' encodings associated with it and it is therefore not possible to extract its description from the
server.
            ' We therefore use a hard-coded data type ID for one of the sub-types in this example.
            '
            ' The code to obtain the data type ID for given node would normally look like this:
            '    UANodeId dataTypeId = client.ReadDataType(
            '        endpointDescriptor,
            '        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            '
            Dim dataTypeId As UANodeId = "nsu=http://test.org/UA/Data/;i=9440"  ' ScalarValueDataType

            ' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
            ' operations.
            Dim complexData As IEasyUAClientComplexData = client.GetService(Of IEasyUAClientComplexData)()

            ' Resolve the data type ID to the data type object, containing description of the data type.
            Dim dataTypeResult As ValueResult(Of DataType) = complexData.ResolveDataType( _
                New UAModelNodeDescriptor(endpointDescriptor, dataTypeId), _
                UABrowseNames.DefaultBinary)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not dataTypeResult.Succeeded Then
                Console.WriteLine("*** Failure: {0}", dataTypeResult.ErrorMessageBrief)
                Exit Sub
            End If

            ' The actual data type is in the Value property.
            ' Display basic information about what we have obtained.
            Console.WriteLine(dataTypeResult.Value)

            ' If we want to see the whole hierarchy of the received data type, we can format it with the "V"
(verbose)
            ' specifier. In the debugger, you can view the same by displaying the private DebugView property.
            Console.WriteLine()
            Console.WriteLine("{0:V}", dataTypeResult.Value)

            ' For processing the internals of the data type, refer to examples for GenericData class.


            ' Example output (truncated):
            '
            'ScalarValueDataType = structured
            '
            'ScalarValueDataType = structured
            '   [BooleanValue] Boolean = primitive(System.Boolean)
            '   [ByteStringValue] ByteString = primitive(System.Byte[])
            '   [ByteValue] Byte = primitive(System.Byte)
            '   [DateTimeValue] DateTime = primitive(System.DateTime)
            '   [DoubleValue] Double = primitive(System.Double)
            '   [EnumerationValue] Int32 = primitive(System.Int32)
            '   [ExpandedNodeIdValue] ExpandedNodeId = structured
            '     [ByteString] optional ByteStringNodeId = structured
            '       [Identifier] ByteString = primitive(System.Byte[])
            '       [NamespaceIndex] UInt16 = primitive(System.UInt16)
            '     [FourByte] optional FourByteNodeId = structured
            '       [Identifier] UInt16 = primitive(System.UInt16)
            '       [NamespaceIndex] Byte = primitive(System.Byte)
            '     [Guid] optional GuidNodeId = structured
            '       [Identifier] Guid = primitive(System.Guid)
            '       [NamespaceIndex] UInt16 = primitive(System.UInt16)
            '     [NamespaceURI] optional CharArray = primitive(System.String)
```

```
            '    [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
            '    [NodeIdType] switch NodeIdType = enumeration(6)
            '      TwoByte = 0
            '      FourByte = 1
            '      Numeric = 2
            '      String = 3
            '      Guid = 4
            '      ByteString = 5
            '    [Numeric] optional NumericNodeId = structured
            '      [Identifier] UInt32 = primitive(System.UInt32)
            '      [NamespaceIndex] UInt16 = primitive(System.UInt16)
            '    [ServerIndex] optional UInt32 = primitive(System.UInt32)
            '    [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
            '    [String] optional StringNodeId = structured
            '      [Identifier] CharArray = primitive(System.String)
            '      [NamespaceIndex] UInt16 = primitive(System.UInt16)
            '    [TwoByte] optional TwoByteNodeId = structured
            '      [Identifier] Byte = primitive(System.Byte)
            '  [FloatValue] Float = primitive(System.Single)
            '  [GuidValue] Guid = primitive(System.Guid)
            '  [Int16Value] Int16 = primitive(System.Int16)
            '  [Int32Value] Int32 = primitive(System.Int32)
            '  [Int64Value] Int64 = primitive(System.Int64)
            '  [Integer] Variant = structured
            '    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
            '    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            '    [ArrayLength] length optional Int32 = primitive(System.Int32)
            '    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            '    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
            '    [Byte] optional sequence[*] of Byte = primitive(System.Byte)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to obtain object describing the data type of complex data node with OPC UA Complex Data plug-in.

Option Explicit

' Define which server we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain the data type ID.
'
' In many cases, you would be able to obtain the data type ID of a particular node by reading its DataType
' attribute.
' The sample server, however, shows a more advanced approach in which the data type ID refers to an abstract data
type, and
' the actual values are then sub-types of this base data type. This abstract data type does not have any
encodings
' associated with it and it is therefore not possible to extract its description from the server. We therefore
use
' a hard-coded data type ID for one of the sub-types in this example.
Dim dataTypeId: dataTypeId = "nsu=http://test.org/UA/Data/;i=9440"    ' ScalarValueDataType

' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
' operations.
Dim ComplexData: Set ComplexData = _
    Client.GetServiceByName("OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData, OpcLabs.EasyOpcUA")

' Resolve the data type ID to the data type object, containing description of the data type.
Dim ModelNodeDescriptor: Set ModelNodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.InformationModel.UAModelNodeDescriptor")
ModelNodeDescriptor.EndpointDescriptor.UrlString = endpointDescriptor
ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText = dataTypeId
Dim EncodingName: Set EncodingName = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UAQualifiedName")
```

```
EncodingName.StandardName = "DefaultBinary"
Dim DataTypeResult: Set DataTypeResult = ComplexData.ResolveDataType(ModelNodeDescriptor, EncodingName)
If Not DataTypeResult.Succeeded Then
    WScript.Echo "*** Failure: "  & DataTypeResult.Exception.GetBaseException().Message
    WScript.Quit
End If

' The actual data type is in the Value property.
Dim DataType: Set DataType = DataTypeResult.Value

' Display basic information about what we have obtained.
WScript.Echo DataType

' If we want to see the whole hierarchy of the received data type, we can format it with the "V" (verbose)
' specifier.
WScript.Echo
WScript.Echo DataType.ToString_2("V", Nothing)

' For processing the internals of the data type, refer to examples for GenericData class.


' Example output (truncated):
'
'ScalarValueDataType = structured
'
'ScalarValueDataType = structured
'  [BooleanValue] Boolean = primitive(System.Boolean)
'  [ByteStringValue] ByteString = primitive(System.Byte[])
'  [ByteValue] Byte = primitive(System.Byte)
'  [DateTimeValue] DateTime = primitive(System.DateTime)
'  [DoubleValue] Double = primitive(System.Double)
'  [EnumerationValue] Int32 = primitive(System.Int32)
'  [ExpandedNodeIdValue] ExpandedNodeId = structured
'    [ByteString] optional ByteStringNodeId = structured
'      [Identifier] ByteString = primitive(System.Byte[])
'      [NamespaceIndex] UInt16 = primitive(System.UInt16)
'    [FourByte] optional FourByteNodeId = structured
'      [Identifier] UInt16 = primitive(System.UInt16)
'      [NamespaceIndex] Byte = primitive(System.Byte)
'    [Guid] optional GuidNodeId = structured
'      [Identifier] Guid = primitive(System.Guid)
'      [NamespaceIndex] UInt16 = primitive(System.UInt16)
'    [NamespaceURI] optional CharArray = primitive(System.String)
'    [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
'    [NodeIdType] switch NodeIdType = enumeration(6)
'      TwoByte = 0
'      FourByte = 1
'      Numeric = 2
'      String = 3
'      Guid = 4
'      ByteString = 5
'    [Numeric] optional NumericNodeId = structured
'      [Identifier] UInt32 = primitive(System.UInt32)
'      [NamespaceIndex] UInt16 = primitive(System.UInt16)
'    [ServerIndex] optional UInt32 = primitive(System.UInt32)
'    [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
'    [String] optional StringNodeId = structured
'      [Identifier] CharArray = primitive(System.String)
'      [NamespaceIndex] UInt16 = primitive(System.UInt16)
'    [TwoByte] optional TwoByteNodeId = structured
'      [Identifier] Byte = primitive(System.Byte)
'  [FloatValue] Float = primitive(System.Single)
'  [GuidValue] Guid = primitive(System.Guid)
'  [Int16Value] Int16 = primitive(System.Int16)
'  [Int32Value] Int32 = primitive(System.Int32)
'  [Int64Value] Int64 = primitive(System.Int64)
'  [Integer] Variant = structured
'    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
'    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
'    [ArrayLength] length optional Int32 = primitive(System.Int32)
'    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
'    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
'    [Byte] optional sequence[*] of Byte = primitive(System.Byte)
```

Obtaining content of the data type dictionary

You may sometimes need to view the actual content of the data type dictionary that resides in the OPC server. In itself, this is easy: The data type dictionary is in a node, so you just read the node. But, how do you find out which node to read? Here is where the IUADataTypeModelProvider Interface or the IUADataTypeDictionaryProvider Interface can help. They have methods to navigate from the node you want to access (read, write, or subscribe to) to the data type dictionary. Or, they have similar methods if you know the data type ID of the data type you are interested in. The example below shows the steps necessary.

C#

```
// Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
// content of the data type dictionary.

using System;
using System.Text;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.OperationModel.Generic;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.DataTypeModel;
using OpcLabs.EasyOpc.UA.DataTypeModel.Extensions;
using OpcLabs.EasyOpc.UA.InformationModel;
using OpcLabs.EasyOpc.UA.Plugins.ComplexData;

namespace UADocExamples.ComplexData._IUADataTypeDictionaryProvider
{
    class ResolveDataTypeDescriptorFromDataTypeEncodingId
    {
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Obtain the data type ID.
            //
            // In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
            // attribute, or easier, by calling the extension method ReadDataType on the IEasyUAClient interface.
The sample
            // server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type,
            // and the actual values are then sub-types of this base data type. This abstract data type does not
have any
            // encodings associated with it and it is therefore not possible to extract its description from the
server.
            // We therefore use a hard-coded data type ID for one of the sub-types in this example.
            //
            // The code to obtain the data type ID for given node would normally look like this:
            //    UANodeId dataTypeId = client.ReadDataType(
            //        endpointUriString,
            //        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            //
            UANodeId dataTypeId = "nsu=http://test.org/UA/Data/;i=9440";    // ScalarValueDataType

            // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
            // operations.
            IEasyUAClientComplexData complexData = client.GetService<IEasyUAClientComplexData>();
```

```
            // Get the data type model provider. Provides methods to access data types in OPC UA model.
            IUADataTypeModelProvider dataTypeModelProvider = complexData.DataTypeModelProvider;

            // Resolve the data type ID from our data type ID, for encoding name "Default Binary".
            ValueResult<UAModelNodeDescriptor> encodingIdResult =
dataTypeModelProvider.ResolveEncodingIdFromDataTypeId(
                new UAModelNodeDescriptor(endpointDescriptor, dataTypeId),
                UABrowseNames.DefaultBinary);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!encodingIdResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", encodingIdResult.ErrorMessageBrief);
                return;
            }
            UAModelNodeDescriptor encodingId = encodingIdResult.Value;

            // Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA
model.
            IUADataTypeDictionaryProvider dataTypeDictionaryProvider = complexData.DataTypeDictionaryProvider;

            // Resolve the data type descriptor from the encoding ID.
            ValueResult<UADataTypeDescriptor> dataTypeDescriptorResult =
                dataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(encodingId);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!dataTypeDescriptorResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", dataTypeDescriptorResult.ErrorMessageBrief);
                return;
            }
            UADataTypeDescriptor dataTypeDescriptor = dataTypeDescriptorResult.Value;

            // The data type descriptor contains two pieces of information:
            // The data type dictionary ID: This determines the dictionary where the data type is defined.
            Console.WriteLine(dataTypeDescriptor.DataTypeDictionaryId);
            // And the data type description: It is a "pointer" into the data type dictionary itself, selecting a
specific
            // type definition inside the data type dictionary. The format of it depends on the data type system;
            // in our case, it is a string that is the name of one of the type elements in the XML document of
the data type
            // dictionary.
            Console.WriteLine(dataTypeDescriptor.DataTypeDescription);

            // Obtain the actual content of the data type dictionary.
            ValueResult<byte[]> dataTypeDictionaryResult =

dataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(dataTypeDescriptor.DataTypeDictionaryId);

            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!dataTypeDictionaryResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", dataTypeDictionaryResult.ErrorMessageBrief);
                return;
            }
            byte[] dataTypeDictionary = dataTypeDictionaryResult.Value;

            // The data type dictionary returned is an array of bytes; its syntax and semantics depends on the
data type
            // system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
            string text = Encoding.UTF8.GetString(dataTypeDictionary);
            Console.WriteLine();
            Console.WriteLine(text);


            // Example output (truncated):
            //
            //http://opcua.demo-this.com:51211/UA/SampleServer;
NodeId="nsu=http://test.org/UA/Data/;ns=2;i=11422"
            //ScalarValueDataType
            //
```

```
            //<opc:TypeDictionary
            //  xmlns:opc="http://opcfoundation.org/BinarySchema/"
            //  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            //  xmlns:ua="http://opcfoundation.org/UA/"
            //  xmlns:tns="http://test.org/UA/Data/"
            //  DefaultByteOrder="LittleEndian"
            //  TargetNamespace="http://test.org/UA/Data/"
            //>
            //  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version
1.1.334.0 of the OPC UA deliverables. -->
            //  <opc:Import Namespace="http://opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
            //
            //  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
            //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
            //    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
            //    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
            //    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
            //    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
            //    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
            //    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
            //    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
            //    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
            //    <opc:Field Name="FloatValue" TypeName="opc:Float" />
            //    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
            //    <opc:Field Name="StringValue" TypeName="opc:String" />
            //    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
            //    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
            //    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
            //    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
            //    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
            //    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
            //    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
            //    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
            //    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
            //    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
            //    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
            //    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
            //    <opc:Field Name="Number" TypeName="ua:Variant" />
            //    <opc:Field Name="Integer" TypeName="ua:Variant" />
            //    <opc:Field Name="UInteger" TypeName="ua:Variant" />
            //  </opc:StructuredType>
            //
            //  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
            //    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
            //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
            //    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
        }
    }
}
```

Object Pascal

```
// Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
// content of the data type dictionary.

class procedure ResolveDataTypeDescriptorFromDataTypeEncodingId.Main;
var
  Client: _EasyUAClient;
  ComplexData: _EasyUAClientComplexData;
  DataTypeDictionary: OleVariant;
  DataTypeDictionaryResult: _ValueResult;
  DataTypeDescriptor: _UADataTypeDescriptor;
  DataTypeDescriptorResult: _ValueResult;
  DataTypeId: string;
  DataTypeModelProvider: _UADataTypeModelProvider;
  DataTypeDictionaryProvider: _UADataTypeDictionaryProvider;
  EncodingId: _UAModelNodeDescriptor;
  EncodingIdResult: _ValueResult;
  EncodingName: _UAQualifiedName;
  EndpointDescriptor: string;
  I: Cardinal;
  ModelNodeDescriptor: _UAModelNodeDescriptor;
```

```
  Text: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
    //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
    //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain the data type ID.
  //
  // In many cases, you would be able to obtain the data type ID of a particular node by reading its DataType
  // attribute.
  // The sample server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type, and
  // the actual values are then sub-types of this base data type. This abstract data type does not have any
encodings
  // associated with it and it is therefore not possible to extract its description from the server. We therefore
use
  // a hard-coded data type ID for one of the sub-types in this example.
  DataTypeId := 'nsu=http://test.org/UA/Data/;i=9440';    // ScalarValueDataType

  // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
  // operations.
  ComplexData :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData,
OpcLabs.EasyOpcUA')) as _EasyUAClientComplexData;

  // Get the data type model provider. Provides methods to access data types in OPC UA model.
  DataTypeModelProvider := ComplexData.DataTypeModelProvider;

  // Resolve the data type ID from our data type ID, for encoding name "Default Binary".
  ModelNodeDescriptor := CoUAModelNodeDescriptor.Create;
  ModelNodeDescriptor.EndpointDescriptor.UrlString := EndpointDescriptor;
  ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText := DataTypeId;
  EncodingName := CoUAQualifiedName.Create;
  EncodingName.StandardName := 'DefaultBinary';
  EncodingIdResult := DataTypeModelProvider.ResolveEncodingIdFromDataTypeId(ModelNodeDescriptor, EncodingName);
  // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
  if not EncodingIdResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [EncodingIdResult.ErrorMessageBrief]));
    Exit;
  end;
  EncodingId := IUnknown(EncodingIdResult.Value) as _UAModelNodeDescriptor;

  // Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA model.
  DataTypeDictionaryProvider := ComplexData.DataTypeDictionaryProvider;

  // Resolve the data type descriptor from the encoding ID.
  DataTypeDescriptorResult :=
DataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(EncodingId);
  // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
  if not DataTypeDescriptorResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [DataTypeDescriptorResult.ErrorMessageBrief]));
    Exit;
  end;
  DataTypeDescriptor := IUnknown(DataTypeDescriptorResult.Value) as _UADataTypeDescriptor;

  // The data type descriptor contains two pieces of information:
  // The data type dictionary ID: This determines the dictionary where the data type is defined.
  WriteLn(DataTypeDescriptor.DataTypeDictionaryId.ToString);
  // And the data type description: It is a "pointer" into the data type dictionary itself, selecting a specific
  // type definition inside the data type dictionary. The format of it depends on the data type system;
  // in our case, it is a string that is the name of one of the type elements in the XML document of the data
type
  // dictionary.
  WriteLn(DataTypeDescriptor.DataTypeDescription);

  // Obtain the actual content of the data type dictionary.
```

```
  DataTypeDictionaryResult :=

DataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(DataTypeDescriptor.DataTypeDictionaryId);

  // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
  if not DataTypeDictionaryResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [DataTypeDictionaryResult.ErrorMessageBrief]));
    Exit;
  end;
  DataTypeDictionary := DataTypeDictionaryResult.Value;

  // The data type dictionary returned is an array of bytes; its syntax and semantics depends on the data type
  // system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
  for I := VarArrayLowBound(DataTypeDictionary, 1) to VarArrayHighBound(DataTypeDictionary, 1) do
  begin
      Text := Text + Chr(Byte(DataTypeDictionary[I]));
  end;

  WriteLn;
  WriteLn(Text);

  // Example output (truncated):
  //
  //http://opcua.demo-this.com:51211/UA/SampleServer; NodeId="nsu=http://test.org/UA/Data/;ns=2;i=11422"
  //ScalarValueDataType
  //
  //<opc:TypeDictionary
  //  xmlns:opc="http://opcfoundation.org/BinarySchema/"
  //  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  //  xmlns:ua="http://opcfoundation.org/UA/"
  //  xmlns:tns="http://test.org/UA/Data/"
  //  DefaultByteOrder="LittleEndian"
  //  TargetNamespace="http://test.org/UA/Data/"
  //>
  //  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version 1.1.334.0
of the OPC UA deliverables. -->
  //  <opc:Import Namespace="http://opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
  //
  //  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
  //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
  //    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
  //    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
  //    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
  //    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
  //    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
  //    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
  //    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
  //    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
  //    <opc:Field Name="FloatValue" TypeName="opc:Float" />
  //    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
  //    <opc:Field Name="StringValue" TypeName="opc:String" />
  //    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
  //    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
  //    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
  //    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
  //    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
  //    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
  //    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
  //    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
  //    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
  //    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
  //    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
  //    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
  //    <opc:Field Name="Number" TypeName="ua:Variant" />
  //    <opc:Field Name="Integer" TypeName="ua:Variant" />
  //    <opc:Field Name="UInteger" TypeName="ua:Variant" />
  //  </opc:StructuredType>
  //
  //  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
  //    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
  //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
  //    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
```

```
end;
```

VB.NET

```
' Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
' content of the data type dictionary.

Imports System
Imports System.Text
Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.OperationModel.Generic
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.DataTypeModel
Imports OpcLabs.EasyOpc.UA.DataTypeModel.Extensions
Imports OpcLabs.EasyOpc.UA.InformationModel
Imports OpcLabs.EasyOpc.UA.Plugins.ComplexData

Namespace UADocExamples.ComplexData._IUADataTypeDictionaryProvider

    Friend Class ResolveDataTypeDescriptorFromDataTypeEncodingId

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Obtain the data type ID.
            '
            ' In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
            ' attribute, or easier, by calling the extension method ReadDataType on the IEasyUAClient interface.
The sample
            ' server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type,
            ' and the actual values are then sub-types of this base data type. This abstract data type does not
have any
            ' encodings associated with it and it is therefore not possible to extract its description from the
server.
            ' We therefore use a hard-coded data type ID for one of the sub-types in this example.
            '
            ' The code to obtain the data type ID for given node would normally look like this:
            '    UANodeId dataTypeId = client.ReadDataType(
            '        endpointUriString,
            '        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            '
            Dim dataTypeId As UANodeId = "nsu=http://test.org/UA/Data/;i=9440"  ' ScalarValueDataType

            ' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
            ' operations.
            Dim complexData As IEasyUAClientComplexData = client.GetService(Of IEasyUAClientComplexData)()

            ' Get the data type model provider. Provides methods to access data types in OPC UA model.
            Dim dataTypeModelProvider As IUADataTypeModelProvider = complexData.DataTypeModelProvider

            ' Resolve the data type ID from our data type ID, for encoding name "Default Binary".
            Dim encodingIdResult As ValueResult(Of UAModelNodeDescriptor) =
dataTypeModelProvider.ResolveEncodingIdFromDataTypeId(
                New UAModelNodeDescriptor(endpointDescriptor, dataTypeId),
                UABrowseNames.DefaultBinary)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not encodingIdResult.Succeeded Then
```

```
                Console.WriteLine("*** Failure: {0}", encodingIdResult.ErrorMessageBrief)
                Exit Sub
            End If
            Dim encodingId As UAModelNodeDescriptor = encodingIdResult.Value

            ' Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA
model.
            Dim dataTypeDictionaryProvider As IUADataTypeDictionaryProvider =
complexData.DataTypeDictionaryProvider

            ' Resolve the data type descriptor from the encoding ID.
            Dim dataTypeDescriptorResult As ValueResult(Of UADataTypeDescriptor) =
                    dataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(encodingId)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not dataTypeDescriptorResult.Succeeded Then
                Console.WriteLine("*** Failure: {0}", dataTypeDescriptorResult.ErrorMessageBrief)
                Exit Sub
            End If
            Dim dataTypeDescriptor As UADataTypeDescriptor = dataTypeDescriptorResult.Value

            ' The data type descriptor contains two pieces of information:
            ' The data type dictionary ID: This determines the dictionary where the data type is defined.
            Console.WriteLine(dataTypeDescriptor.DataTypeDictionaryId)
            ' And the data type description: It is a "pointer" into the data type dictionary itself, selecting a
specific
            ' type definition inside the data type dictionary. The format of it depends on the data type system;
            ' in our case, it is a string that is the name of one of the type elements in the XML document of the
data type
            ' dictionary.
            Console.WriteLine(dataTypeDescriptor.DataTypeDescription)

            ' Obtain the actual content of the data type dictionary.
            Dim dataTypeDictionaryResult As ValueResult(Of Byte()) =

dataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(dataTypeDescriptor.DataTypeDictionaryId)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not dataTypeDictionaryResult.Succeeded Then
                Console.WriteLine("*** Failure: {0}", dataTypeDictionaryResult.ErrorMessageBrief)
                Exit Sub
            End If
            Dim dataTypeDictionary() As Byte = dataTypeDictionaryResult.Value

            ' The data type dictionary returned is an array of bytes; its syntax and semantics depends on the
data type
            ' system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
            Dim text As String = Encoding.UTF8.GetString(dataTypeDictionary)
            Console.WriteLine()
            Console.WriteLine(text)


            ' Example output (truncated):
            '
            'http://opcua.demo-this.com:51211/UA/SampleServer; NodeId="nsu=http://test.org/UA/Data/;ns=2;i=11422"
            'ScalarValueDataType
            '
            '<opc:TypeDictionary
            '  xmlns:opc="http://opcfoundation.org/BinarySchema/"
            '  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            '  xmlns:ua="http://opcfoundation.org/UA/"
            '  xmlns:tns="http://test.org/UA/Data/"
            '  DefaultByteOrder="LittleEndian"
            '  TargetNamespace="http://test.org/UA/Data/"
            '>
            '  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version
1.1.334.0 of the OPC UA deliverables. -->
            '  <opc:Import Namespace="http://opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
            '
            '  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
            '    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
            '    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
            '    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
```

```
            '    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
            '    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
            '    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
            '    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
            '    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
            '    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
            '    <opc:Field Name="FloatValue" TypeName="opc:Float" />
            '    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
            '    <opc:Field Name="StringValue" TypeName="opc:String" />
            '    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
            '    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
            '    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
            '    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
            '    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
            '    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
            '    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
            '    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
            '    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
            '    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
            '    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
            '    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
            '    <opc:Field Name="Number" TypeName="ua:Variant" />
            '    <opc:Field Name="Integer" TypeName="ua:Variant" />
            '    <opc:Field Name="UInteger" TypeName="ua:Variant" />
            '  </opc:StructuredType>
            '
            '  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
            '    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
            '    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
            '    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
Rem content of the data type dictionary.

Option Explicit

' Define which server we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain the data type ID.
'
' In many cases, you would be able to obtain the data type ID of a particular node by reading its DataType
' attribute.
' The sample server, however, shows a more advanced approach in which the data type ID refers to an abstract data
type, and
' the actual values are then sub-types of this base data type. This abstract data type does not have any
encodings
' associated with it and it is therefore not possible to extract its description from the server. We therefore
use
' a hard-coded data type ID for one of the sub-types in this example.
Dim dataTypeId: dataTypeId = "nsu=http://test.org/UA/Data/;i=9440"    ' ScalarValueDataType

' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
' operations.
Dim ComplexData: Set ComplexData = _
    Client.GetServiceByName("OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData, OpcLabs.EasyOpcUA")

' Get the data type model provider. Provides methods to access data types in OPC UA model.
Dim DataTypeModelProvider: Set DataTypeModelProvider = ComplexData.DataTypeModelProvider

' Resolve the data type ID from our data type ID, for encoding name "Default Binary".
```

```
Dim ModelNodeDescriptor: Set ModelNodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.InformationModel.UAModelNodeDescriptor")
ModelNodeDescriptor.EndpointDescriptor.UrlString = endpointDescriptor
ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText = dataTypeId
Dim EncodingName: Set EncodingName = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UAQualifiedName")
EncodingName.StandardName = "DefaultBinary"
Dim EncodingIdResult: Set EncodingIdResult = _
    DataTypeModelProvider.ResolveEncodingIdFromDataTypeId(ModelNodeDescriptor, EncodingName)
' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
If Not EncodingIdResult.Succeeded Then
    WScript.Echo "*** Failure: "  & EncodingIdResult.Exception.GetBaseException().Message
    WScript.Quit
End If
Dim EncodingId: Set EncodingId = EncodingIdResult.Value

' Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA model.
Dim DataTypeDictionaryProvider: Set DataTypeDictionaryProvider = ComplexData.DataTypeDictionaryProvider

' Resolve the data type descriptor from the encoding ID.
Dim DataTypeDescriptorResult: Set DataTypeDescriptorResult = _
    DataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(EncodingId)
' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
If Not DataTypeDescriptorResult.Succeeded Then
    WScript.Echo "*** Failure: "  & DataTypeDescriptorResult.Exception.GetBaseException().Message
    WScript.Quit
End If
Dim DataTypeDescriptor: Set DataTypeDescriptor = DataTypeDescriptorResult.Value

' The data type descriptor contains two pieces of information:
' The data type dictionary ID: This determines the dictionary where the data type is defined.
WScript.Echo DataTypeDescriptor.DataTypeDictionaryId
' And the data type description: It is a "pointer" into the data type dictionary itself, selecting a specific
' type definition inside the data type dictionary. The format of it depends on the data type system;
' in our case, it is a string that is the name of one of the type elements in the XML document of the data type
' dictionary.
WScript.Echo DataTypeDescriptor.DataTypeDescription

' Obtain the actual content of the data type dictionary.
Dim DataTypeDictionaryResult: Set DataTypeDictionaryResult = _

DataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(DataTypeDescriptor.DataTypeDictionaryId)
' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
If Not DataTypeDictionaryResult.Succeeded Then
    WScript.Echo "*** Failure: "  & DataTypeDictionaryResult.Exception.GetBaseException().Message
    WScript.Quit
End If
Dim DataTypeDictionary: DataTypeDictionary = DataTypeDictionaryResult.Value

' The data type dictionary returned is an array of bytes; its syntax and semantics depends on the data type
' system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
Dim text
Dim i: For i = LBound(DataTypeDictionary) To UBound(DataTypeDictionary)
    text = text & Chr(DataTypeDictionary(i))
Next
WScript.Echo
WScript.Echo text


' Example output (truncated):
'
'http://opcua.demo-this.com:51211/UA/SampleServer; NodeId="nsu=http:'test.org/UA/Data/;ns=2;i=11422"
'ScalarValueDataType
'
'<opc:TypeDictionary
'  xmlns:opc="http:'opcfoundation.org/BinarySchema/"
'  xmlns:xsi="http:'www.w3.org/2001/XMLSchema-instance"
'  xmlns:ua="http:'opcfoundation.org/UA/"
'  xmlns:tns="http:'test.org/UA/Data/"
'  DefaultByteOrder="LittleEndian"
'  TargetNamespace="http:'test.org/UA/Data/"
'>
'  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version 1.1.334.0 of
the OPC UA deliverables. -->
```

```
'  <opc:Import Namespace="http:'opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
'
'  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
'    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
'    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
'    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
'    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
'    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
'    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
'    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
'    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
'    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
'    <opc:Field Name="FloatValue" TypeName="opc:Float" />
'    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
'    <opc:Field Name="StringValue" TypeName="opc:String" />
'    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
'    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
'    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
'    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
'    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
'    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
'    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
'    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
'    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
'    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
'    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
'    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
'    <opc:Field Name="Number" TypeName="ua:Variant" />
'    <opc:Field Name="Integer" TypeName="ua:Variant" />
'    <opc:Field Name="UInteger" TypeName="ua:Variant" />
'  </opc:StructuredType>
'
'  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
'    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
'    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
'    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
```

7.2.1.4.4 Adding or removing data type dictionaries to or from the OPC UA data type system

In some cases, you may want the component to use a data type dictionary that is different from what resides in the OPC server, or is not present in the OPC server at all. These situations may be, for example:

- The server vendor has intentionally omitted the data type dictionary from the OPC server due to performance and size constraints (small embedded devices).
- The server vendor has omitted the data type dictionary from the OPC server due to mistake or overlook.
- The OPC server contains the data type dictionary, but the dictionary does not match the actual data types and encoding.

Each data type dictionary is identified by the endpoint descriptor and node descriptor (data type dictionary ID) in the data type system. You can externally add or remove data type dictionaries from your code, identifying them by the data type dictionary ID.

Take following steps to add or remove the data type dictionary:

1. Get the IEasyUAClientComplexData Interface service from your EasyUAClient Class instance using the GetService Method.
2. Using the FindDataTypeSystem Method on the IEasyUAClientComplexData Interface, find the data type system you

want to manipulate. Pass in the encoding name of the data type system, e.g. DefaultBinary Property.
3. Call the AddDataTypeDictionary Method or the RemoveDataTypeDictionary Method on the IUADataTypeSystem Interface.

Note that the data type system may only contain data type dictionaries that were previously referenced. Once the data type dictionary is added, be it from your code or otherwise, it stays there until removed. If your intent is to re-define an existing data type dictionary or define a data type dictionary that does not exist in the server, you need to add it to the data type system at the beginning, before an attempt tor etrieve it from the server is made.

8 Application Deployment

This chapter describes how to deploy applications developed with use of OPC Data Client.

For some uses, it is acceptable to perform the deployment manually. In other cases, you will find yourself writing an installer for your application, and you will probably want to include the installation of OPC Data Client components in it as well. On Windows, Production installer (part of OPC Data Client) can help with that.

8.1 Deployment Methods

You can deploy your application with OPC Data Client using following methods:

1. **Manual Deployment (Section 8.1.1)**: Doing the necessary deployment steps manually. On Windows and when not targeting the .NET Standard, the OPC Data Client installer can be used to help with initial parts of the manual process.
2. **Automated deployment, roll your own (Section 8.1.2)**: By writing your an installer for your application, and including in it the steps needed to deploy the individual OPC Data Client elements.
3. **Automated deployment with Production installer (Section 8.1.3)** (Windows only; not when targeting .NET Standard): By writing an installer for your application, and having your installer embed and invoke the OPC Data Client Production installer.

8.1.1 Manual Deployment

.NET Framework, COM and Excel Development

In order to deploy your .NET Framework, COM or Excel application with OPC Data Client on Windows manually, follow the steps below:

1. Check that proper version of Microsoft .NET Framework is installed, and if it is not, install it. See **.NET Runtimes (Section 2.4.1)** for details.
2. Run the OPC Data Client Full installation program, selecting “Custom install” choice and later “Production installation” type. Alternatively, run the OPC Data Client Production installer instead. The main purpose of this step is to obtain the set of OPC Data Client assemblies; if you have them available by other means, you can skip this step.
3. a. Copy the OPC Data Client assemblies to their target locations for your application.
   b. If you are developing on a COM platform, perform additional actions (such as the assembly registrations) as described with the respective files in **Installing COM Components and Type Libraries (Section 8.3.2)** . This might have been performed by the installer in Step 2 above, but needs to be re-done if you have moved the assemblies from their original locations.
4. Perform any steps needed to install your own application.
5. If you are using a license located in the **Registry License Store (Section 8.5.2)**, run the License Manager from the Start menu or the Launcher application, and install the runtime license. Alternative, use command line and the LMConsole utility for this.
6. With OPC Data Client-UA only: Run your application once with the administrative privileges, giving it a chance to

In This Topic

create its client instance certificate (this step is only needed if you do not provide the client instance certificate in some other way).

.NET Standard Development

In .NET Standard, manual deployment steps are more or less indistinguishable from what is considered as creating an "automated deployment". For this reason, refer to **Automated deployment, roll your own (Section 8.1.2)** for the deployment steps.

8.1.2 Automated deployment, roll your own

.NET Framework, COM and Excel Development

Using this method, you write an installer for your application, and include an installation of OPC Data Client elements in it as well, step by step.

The installer for your application should contain following steps:

1. Check that prerequisites (in their proper versions), are installed, and if they are not, instruct the user to install them (or install them automatically).
2. Install OPC Data Client.NET and/or OPC Data Client-UA assemblies to their target locations for your application.
3. For OPC Data Client-COM and OPC Data Client-UA for COM, perform additional actions (such as the assembly registrations) as described with the respective files in **Installing COM Components and Type Libraries (Section 8.3.2)**.
4. Perform any steps needed to install your own application.
5. With OPC Data Client-UA only: While running with administrative privileges, execute your application with a special command-line switch that will cause it to call static EasyUAClient**.**Install method. This will create the client instance certificate, in accordance with settings of other static properties on the object (this step is only needed if you do not provide the client instance certificate in some other way). For more information, see **Providing OPC UA Client Instance Certificate (Section 4.13.1.1)** and **OPC UA Certificate Stores (Section 4.13.1.4)**.
6. If you are using a license located in the **Registry License Store (Section 8.5.2)**, take care of the license installation (this is described further down).

If you want the end user to install the license:

1. Install LicenseManager.exe or LMConsole.exe (from Bin or Bin\x64).
2. Create an empty registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Licensing\QuickOPC**
3. Offer the user an option to run the License Manager.

The end user who deploys the application will then:

1. Run your installer and follow the steps.
2. Use the License Manager (GUI or console-based) and install the runtime license.

If the above described procedure for installing the license (presenting the user with the License Manager utility user interface) does not fit the needs of your application, you can automate the deployment of the license as well. This can be done in two ways:

a. Directly - by writing information into the license store. For more information, see Direct License Deployment.
b. By using the **LMConsole Utility (License Manager Console) (Section 11.1.1)**. Your installer will (at least temporarily) place the LMConsole.exe and the actual license file on the disk, and then call the LMConsole utility with corresponding parameters on the command line.

The **LicenseManager** (GUI-based) and **LMConsole** (console-based) License Manager utilities are installed on the disk using the OPC Data Client-s **Setup** program.

.NET Standard Development

In .NET Standard, the major installation element is not an assembly, but rather a (NuGet) package, which may consist of multiple assemblies, differentiated by the target platform, and may depend on other packages.

The OPC Data Client license needs to be stored inside your aplication in the **Managed Resource License Store (Section 8.5.1)** (the **Registry License Store (Section 8.5.2)** is not supported in .NET Standard). Consequently, there are no additional deployment elements or steps needed with regard to licensing, because the license key is already embedded in your project's assembly by the build process.

With OPC Data Client-UA only: After deploying the assemblies, your installation may want to execute your application with a special switch that will cause it to call static EasyUAClient.Install method. This will create the client instance certificate, in accordance with settings of other static properties on the object. With the default settings, this step is optional, and the application will create the client instance certificate on the first run anyway. If you, however, change the location of the application certificate store to a store that requires administrative privileges for write access, you should use this approach (while running with administrative privileges) in order to assure that the client instance certificate is properly created. For more information, see **Providing OPC UA Client Instance Certificate (Section 4.13.1.1)** and **OPC UA Certificate Stores (Section 4.13.1.4)**.

.NET Core

In order to deploy your application developed with OPC Data Client for the .NET Core runtime (on Windows, Linux, or other operating system if supported), you can use either:

- **Framework-dependent Deployment (FDD)**, or
- **Self-contained Deployment (SCD)**.

Both these deployment types are described in Microsoft's article .NET Core application deployment. OPC Data Client does not have any special requirements or procedures needed above that.

8.1.3 Automated deployment with Production installer

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

In This Topic

The OPC Data Client contains a separate installation package (for Windows) that is for

production purposes only. This installer executable has a "-Production" postfix. It is much smaller than the Full installer: At the time of writing this article, the full installer has 90 MB, while the production installer has only 10 MB. The production installer contains only the assemblies and the LMConsole utility (for licensing), and a code to optionally register the assemblies as COM components, and/or produce the type libraries (TLB) for them. The production installer does not install any Start menu icons. It is intended for embedding in other installation programs. It can be automated from the command line, and can also be run silently.

The developer will find the link to download the production installer in the Redist directory.

The production installer is created using Inno Setup. This means that if you want to automate it from your own installer using command line, you can use all Setup Command Line Parameters it provides, and check Setup Exit Codes as needed. You will typically use the **/SILENT** or **/VERYSILENT** parameter, possibly combined with **/TYPE**, **/COMPONENTS**, **/TASKS** or **/MERGETASKS** parameters.

Setup Types

The available setup types (for the **/TYPE=*type name*** parameter) are:

Setup Type Name	Description
production	Production installation
productionnet	Production installation (.NET only)
productioncom	Production installation (COM only)
productionexcel	Production installation (Excel only)
custom	Custom installation

Setup Components

If you need more control over the installed parts, you can specify the components to be installed individually. The available components (for the **/COMPONENTS="*comma separated list of component names*"** parameter) are:

Component Name	Description
assemblies	Assemblies
assemblies\embedded	Embedded assemblies
comcomponents	COM Components
typelibs	Type Libraries (COM)
tools\productiontools	Production Tools (License Manager, ...)
options\excel	Excel Option
options\streaminsight	StreamInsight Option

The License Manager component in the Production installer contains only the console-based license manager

(LMConsole.exe), and not the GUI.

Setup Tasks

You can also instruct the Setup to perform additional tasks during installation, using **/TASKS** in the command line. The available tasks are:

Task Name	Description
installtogac	Install component assemblies into the GAC. This is normally neither necessary nor recommended.

License Install

The production installer can be passed an additional command line parameter ("**/LicenseFile=...**") with a path to the license key file (typically with .BIN or .TXT extension), causing it to automatically install the license as part of the setup process.

If you have two license files to install (with some of the **Product Options (Section 12)**), use "**/LicenseFile2=...**" to install the second license file.

⚠ Unless you specify a path in the file name, the file is assumed to be in the same directory as the installer's executable. It is recommended to use a full path in the file name to avoid ambiguity and assure correct installation of the license.

Uninstallation

It is also possible to automate the uninstallation. For that, the production installer places a unins000.exe file in the Setup directory of the product folder, where your own uninstaller can find it and call it as well. More information to this you will find in Uninstaller Command Line Parameters and Uninstaller Exit Codes.

The Production installer does not configure the software. For example, if you need the Connectivity RTD Server to use a specific connectivity configuration, you need to install the **ConnectivityConfiguration2.xml** file yourself.

8.2 Deployment Prerequisites

.NET Framework, COM and Excel Development

Besides the actual library assemblies, QuickOPC requires that the software described below is present on the target system.

.NET Framework - you need (at least) one of the following:

- .NET Framework 4.7
- .NET Framework 4.7.1
- .NET Framework 4.7.2
- .NET Framework 4.8

In This Topic

Note that

- .NET Framework 4.7 is included with Windows 10 version 1703 (Creators Update).
- .NET Framework 4.7.1 is included with Windows 10 version 1709 (Fall Creators update).
- .NET Framework 4.7.2 is included with Windows 10 version 1803 (April 2018 Update), version 1809 (October 2018 Update), Windows Server versions 1803 and 1809, and Windows Server 2019.
- .NET Framework 4.8 is included with Windows 10 version 1903 (May 2019 Update).

In some special cases, you may want to install other prerequisites – please refer to **Prerequisites Boxing (Section 9.12)** under **Advanced Topics (Section 9)**.

.NET Standard Development

.NET Core

If you are using a **framework-dependent deployment (FDD)**, it is necessary that proper version of .NET Core is installed on the target system. If you are using a **self-contained deployment (SCD)**, there are usually no deployment prerequisites other than using a supported operating system and hardware.

See **.NET Runtimes (Section 2.4.1)** for more information about the support .NET Core versions and **Operating Systems (Section 2.4.2)**. For more information about deployment types in .NET Core, see the Microsoft's article .NET Core application deployment.

8.3 Installation Elements

The elements described in this chapter need to be installed with your application based on OPC Data Client.

8.3.1 Installing Assemblies (.NET)

.NET Framework, COM and Excel Development

Depending on which assemblies you have referenced in your application, you may need one or more of the following files be installed with your application:

- App_Web_OpcLabs.EasyOpcClassicRaw.amd64.dll
- App_Web_OpcLabs.EasyOpcClassicRaw.x86.dll
- OpcLabs.BaseLib.dll
- OpcLabs.BaseLibForms.dll
- OpcLabs.BaseLibPresentation.dll
- OpcLabs.EasyOpcClassic.dll
- OpcLabs.EasyOpcForms.dll
- OpcLabs.EasyOpcUA.dll

Please refer to “Product Parts” chapter for description of purpose of individual assemblies. You will find them under the Assemblies\net47 subdirectory in the installation directory of the product.

In This Topic

The assemblies need to be placed so that the referencing software (your application) can find them, according to standard Microsoft .NET loader (Fusion) rules. The easiest is to simply place the assembly files alongside (in the same directory as) your application file.

There are other methods of how assemblies can be located, including their installation to GAC (Global Assembly Cache). Which method you use depends on the needs of your application. Unless you have a compelling need, however, we do not recommended installing the assemblies into the GAC.

.NET Standard Development

Remember that in .NET Standard, the deployment will contain not only the files of your application and OPC Data Client files, but also the files from NuGet packages that OPC Data Client depends on, and that you have to comply with their licensing requirements as well.

.NET Core

In .NET Core, the assemblies that become the part of the deployment are determined and included automatically by the tools that build the deployment. For more information, see Microsoft's article .NET Core application deployment.

OpcLabs.Pcap and its dependencies

If your application needs to use OPC UA PubSub Ethernet mapping, or it needs to work with Wireshark capture files, you also need to install:

- OpcLabs.Pcap.dll
- PacketDotNet.dll
- SharpPcap.dll

In addition, you need to comply with LGPL requirements of PacketDotNet and SharpPcap.

8.3.2 Installing COM Components and Type Libraries

Because OPC Data Client-COM and OPC Data Client-UA for COM are implemented by exposing the .NET objects of OPC Data Client to COM, you need to basically do two things:

1. Deploy the assemblies needed for .NET (see preceding chapter).
2. Register them using the Assembly Registration tool (**Regasm.exe**)

The Assembly Registration tool is a part of .NET Framework, and is therefore available on every computer that has the .NET Framework installed. You only need to apply the Assembly Registration tool to the following assemblies:

- OpcLabs.BaseLib.dll
- OpcLabs.BaseLibForms.dll
- OpcLabs.EasyOpcClassic.dll
- OpcLabs.EasyOpcForms.dll
- OpcLabs.EasyOpcUA.dll

For more information, please refer to http://msdn.microsoft.com/en-us/library/tzat5yw6(v=vs.110).aspx. It is recommended that you use the **/codebaseoption** when registering the assemblies.

⚠ On 64-bit systems, a 32-bit and a 64-bit version of **RegAsm.exe** are present. Using the 32-bit version of **RegAsm.exe** registers the assembly for use by 32-bit COM applications, and using the 64-bit of **RegAsm.exe** registers the assembly for use by 64-bit COM applications. Depending on your needs, you need to select the proper version of **RegAsm.exe**, or even use them both.

8.4 Configuration Elements

There are currently no configuration elements for OPC Data Client deployments.

8.5 Licensing Elements

License Location

In This Topic

The OPC Data Client license is located in a *license store*. The following table lists the license stores available under each development platform, and their search order.

Search Order	License Store Type	COM	.NET Framework	.NET Standard (new in 2018.3)	Excel (with Excel Option (Section 12.1))
1.	**Managed Resource License Store (Section 8.5.1)** (new in 2018.3)	×	✓	✓	×
2.	**Registry License Store (Section 8.5.2)**	✓	✓	×	✓
3.	Built-in License Store (trial license only)	✓	✓	✓	✓

OPC Data Client attempts to find a valid license in the available license stores, using a pre-set search order as given above (from top to bottom). If a valid license is found, it is used. If it is not found, the search proceeds to the next available license store. There is always a valid trial license available in the Built-in License Store. This is the one that will be used if everything else fails.

The **Managed Resource License Store (Section 8.5.1)** allows you to embed the license directly within your application (in a .NET assembly). This method cannot be used on the COM platform. Its advantage is that it does not require any further actions during deployment.

With the **Registry License Store (Section 8.5.2)**, proper license (for runtime usage) must be installed on the target computer (the computer where the product will be running). The **License Manager** utility (GUI or console-based) is normally needed for this. The GUI-based **License Manager** is contained in the **LicenseManager.exe** file. The console-based License Manager is contained in the **LMConsole.exe** file. Both files are located under the **Bin** subdirectory of the product. Alternatively, you can write an installation program that installs the license into the **Registry License Store (Section 8.5.2)** automatically. The **Registry License Store (Section 8.5.2)** is not available on the .NET Standard

development development platform.

> ⚠ The fact that you have used a valid OPC Data Client license in the **Registry License Store (Section 8.5.2)** during development and building of your application does not mean that the resulting application will automatically be licensed on a different computer. You must do the extra steps described in this documentation (depending on the license store chosen) in order to assure that your deployed application is licensed in runtime.

License Verification

If the proper license is not installed, you will end up with the trial license being used instead, as described above. That is usually manifested by the fact that after 30 minutes of the process run time, the component stops delivering valid data and gives an error instead.

Of course, you might want to know and verify sooner whether the license you intended to use has been found, validated and used. You can do so from your program by inspecting the elements of the dictionary returned from the LicenseInfo Property on the EasyXXClient object. The dictionary is keyed by strings, and you can obtain various information related to the license from it. If you simply want to verify that the right license is in use, we suggest that you test its serial number, either by excluding a range for demo and trial licenses (as in the following example), or by requiring that the serial number of the license in use is equal to the precise serial number of your license. You obtain the serial number by getting the **"Multipurpose.SerialNumber"** element of the dictionary. The license key file you obtained when you purchased OPC Data Client has your serial number as its first series of digits.

The example below shows how to obtain the serial number of the active license, and determine whether it is a demo or trial license.

C#

```
// Shows how to obtain the serial number of the active license, and determine whether
it is a stock demo or trial license.

using System;
using OpcLabs.EasyOpc.UA;

namespace UADocExamples.Licensing
{
    class LicenseInfo
    {
        public static void SerialNumber()
        {
            // Instantiate the client object.
            var client = new EasyUAClient();

            // Obtain the serial number from the license info.
            long serialNumber = (uint)client.LicenseInfo["Multipurpose.SerialNumber"];

            // Display the serial number.
            Console.WriteLine("SerialNumber: {0}", serialNumber);

            // Determine whether we are running as demo or trial.
            if ((1111110000 <= serialNumber) && (serialNumber <= 1111119999))
                Console.WriteLine("This is a stock demo or trial license.");
            else
                Console.WriteLine("This is not a stock demo or trial license.");
```

```
        }
    }
}
```

Object Pascal

```
// Shows how to obtain the serial number of the active license, and determine whether
it is a stock demo or trial license.

class procedure LicenseInfo.SerialNumber;
var
  Client: _EasyUAClient;
  SerialNumber: Int64;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain the serial number from the license info.
  SerialNumber := Int64(client.LicenseInfo['Multipurpose.SerialNumber']);

  // Display the serial number.
  WriteLn('SerialNumber: ', SerialNumber);

  // Determine whether we are running as demo or trial.
  if ((1111110000 <= SerialNumber) and (SerialNumber <= 1111119999)) then
    WriteLn('This is a stock demo or trial license.')
  else
    WriteLn('This is not a stock demo or trial license.');
end;
```

PHP

```
// This example shows how to obtain nodes under a given node of the OPC-UA address
space.
// Shows how to obtain the serial number of the active license, and determine whether
it is a stock demo or trial license.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain the serial number from the license info.
$SerialNumber = $Client->LicenseInfo["Multipurpose.SerialNumber"];

// Display the serial number.
printf("SerialNumber: %s\n", $SerialNumber);

// Determine whether we are running as demo or trial.
if ((1111110000 <= $SerialNumber) and ($SerialNumber <= 1111119999))
    printf("This is a stock demo or trial license.\n");
  else
    printf("This is not a stock demo or trial license.\n");
```

VB.NET

```
' Shows how to obtain the serial number of the active license, and determine whether it
is a stock demo or trial license.
```

```
Imports OpcLabs.EasyOpc.UA

Namespace UADocExamples.Licensing
    Friend Class LicenseInfo
        Public Shared Sub SerialNumber()

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            ' Obtain the serial number from the license info.
            Dim serialNumber As Long =
CUInt(client.LicenseInfo("Multipurpose.SerialNumber"))

            ' Display the serial number.
            Console.WriteLine("SerialNumber: {0}", serialNumber)

            ' Determine whether we are running as demo or trial.
            If (1111110000 <= serialNumber) And (serialNumber <= 1111119999) Then
                Console.WriteLine("This is a stock demo or trial license.")
            Else
                Console.WriteLine("This is not a stock demo or trial license.")
            End If
        End Sub
    End Class
End Namespace
```

Legal Concerns

Your usage of the software created with elements from OPC Data Client must comply with the terms of OPC Data Client EULA. In addition, because OPC Data Client contains and/or may depend on licensed material from 3rd parties, you must comply with license terms of such material as well. See **Licenses for Redistributed Material (Section 8.5.3)** for more details.

8.5.1 Managed Resource License Store

With the Managed Resource License Store, you embed your license into one of the assemblies that your application comprises of.

> This functionality is not available under (or the text does not apply to) COM development platform.

The main advantage of this method is that it does not require any additional actions during deployment. Also, because the Managed Resource License is the first one in the search order, you have a guarantee that if you properly place a specific valid license into the managed resource license store, it will be the one actually used - it cannot be changed by License Manager, for example. The disadvantage of this method is that you need to put extra code and data into your application, before building it.

Concepts

Follow the steps below in order to place a license into the Managed Resource License Store and use it. The procedure is written for C# in Visual Studio, but should be similar with other managed languages and tools.

1. Place the license file into your project. To do so, pick a folder in your project, and in **Solution Explorer**, right-click on it and select Add -> Existing Item. Then, select your license file; this is the file that has been given to you with OPC Data Client purchase. It has .BIN extension, and its name usually starts with a **"Key-Permanent-..."**. Press Add in order to finalize the selection.
2. Turn the license file in your project to an embedded resource: Select the file in **Solution Explorer**. Then, in the **Properties** window, select the Build Action row. Click on the drop-down button on its right side and select "Embedded Resource" from the list; alternatively, use up and down keys to make the same selection.
3. At the beginning of your code, before you call any OPC Data Client operations, obtain the static Instance Field from the LicensingManagement Class, and call the RegisterManagedResource Method with appropriate arguments. The first two arguments should always be **"QuickOPC"** and **"Multipurpose"**. The third argument determines the assembly where the license resides; you would typically use the Assembly.GetExecutingAssembly Method here, if the resource is in the same assembly as the code that is calling the RegisterManagedResource Method. The fourth argument is the namespace-qualified name of the managed resource. Visual Studio determines the namespace by concatenating the default namespace (as given in project properties) with names of the project folders (starting from the project root) the embedded resource resides in. So for example, if your default namespace is **MyApp**, and you have placed the embedded resource with license key (named **"Key-Permanent-ShareIt-1912345678-20180612.bin"**) to the **Resources** folder, the namespace-qualified name of the managed resource will be **"MyApp.Resource.Key-Permanent-ShareIt-1912345678-20180612.bin"**. If you are unsure, use the ILDASM tool.

Note that the actual presence and validity of the license is not checked at the time when the location is registered.

Example

The example below shows how to register a license located in an embedded managed resource.

C#

```
// Shows how to register a license located in an embedded managed resource.

using System;
using System.Reflection;
using OpcLabs.BaseLib.ComponentModel;
using OpcLabs.EasyOpc.UA;

namespace UADocExamples.Licensing
{
    class _LicensingManagement
    {
        public static void RegisterManagedResource()
        {
            // Register a license that is we embed as a managed resource in this
program.

            // The first two arguments should always be "QuickOPC" and "Multipurpose".
            // The third argument determines the assembly where the license resides.
            // The fourth argument is the namespace-qualified name of the managed
resource.
            LicensingManagement.Instance.RegisterManagedResource("QuickOPC",
```

```
"Multipurpose",
                Assembly.GetExecutingAssembly(),
                "UADocExamples.Licensing.Key-DemoOrTrial-WebForm-1999003494-
20180611.bin");

            // Instantiate the client object, obtain the serial number from the license
info, and display the serial number.
            var client = new EasyUAClient();
            long serialNumber = (uint)client.LicenseInfo["Multipurpose.SerialNumber"];
            Console.WriteLine("SerialNumber: {0}", serialNumber);

            // The license we ship for this purpose is a trial license with low runtime
limit, so it won't be of much use.
            // But you get the point...
        }
    }
}
```

VB.NET

```
' Shows how to register a license located in an embedded managed resource.

Imports System.Reflection
Imports OpcLabs.BaseLib.ComponentModel
Imports OpcLabs.EasyOpc.UA

Namespace UADocExamples.Licensing
    Friend Class _LicensingManagement
        Public Shared Sub RegisterManagedResource()

            ' Register a license that is we embed as a managed resource in this
program.

            ' The first two arguments should always be "QuickOPC" and "Multipurpose".
            ' The third argument determines the assembly where the license resides.
            ' The fourth argument is the namespace-qualified name of the managed
resource.
            LicensingManagement.Instance.RegisterManagedResource("QuickOPC",
"Multipurpose",
                    Assembly.GetExecutingAssembly(),
                    "UADocExamples.Licensing.Key-DemoOrTrial-WebForm-1999003494-
20180611.bin")

            ' Instantiate the client object, obtain the serial number from the license
info, and display the serial number.
            Dim client = New EasyUAClient()
            Dim serialNumber As Long =
CUInt(client.LicenseInfo("Multipurpose.SerialNumber"))
            Console.WriteLine("SerialNumber: {0}", serialNumber)

            ' The license we ship for this purpose is a trial license with low runtime
limit, so it won't be of much use.
            ' But you get the point...

        End Sub
    End Class
End Namespace
```

Troubleshooting

If the registration is not done properly, the component will continue the search for the license key in the order described in **Licensing Elements (Section 8.5)**. Usually, you will end up with a trial license instead, which is manifested by receiving a serial number other than the serial number of the license key you attempted to register. The serial number of stock demo or trial license is between 1111110000 and 1111119999 - see **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**.

Here are the common reasons that can cause this issue:

- Your license key file is included in the project, but its Build Action (in **Properties**) is not set to "Embedded Resource". This happens easily, because (in Visual Studio) the file looks the same in **Solution Explorer**; the difference is just a subtle setting visible after you click on the file node in the **Solution Explorer** and inspect its **Properties**.
- The actual namespace of the managed resource does not match the one used with the RegisterManagedResource Method. This happens easily, because the actual namespace is determined using the default project namespace and a path of project folders in the solution, which is not always apparent, and there is no direct way (in Visual Studio) to check the resulting namespace.

If you cannot rule out the the cause is in incorrect resource namespace (or resource name, in general), we recommend to use the ILDASM tool (IL Disassembler, https://docs.microsoft.com/en-us/dotnet/framework/tools/ildasm-exe-il-disassembler) to figure out the precise actual name of your license key managed resource. You can use following steps:

1. **Start Developer Command Prompt** for Visual Studio.
2. Change current directory to the directory where the resulting assembly of your program resides (using the ***CD*** command).
3. Type ***ILDASM PEfilename*** , where *PEFilename* is the name of your assembly file (including file extension, typically .DLL or .EXE).
4. In the **IL DASM** window, double-click on the MANIFEST node.
5. In the **MANIFEST** window, find the license key managed resource and verify its namespace and name with the string used in the call to the RegisterManagedResource Method. Sometimes the differences are slight and not apparent at the first site; if you do not find a difference, we recommend that you copy the relevant part of the text from the MANIFEST window into your .NET source code.

For illustration, the part of the MANIFEST with the license key for our example looks like this:

```
.mresource public 'UADocExamples.Licensing.Key-DemoOrTrial-WebForm-1999003494-
20180611.bin'
{
  // Offset: 0x00000000 Length: 0x00000A68
}
```

8.5.2 Registry License Store

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

In This Topic

Manual License Deployment

When you use the Registry License Store, proper license (for runtime usage) must be installed on the target computer (the computer where the product will be running).

The standard method for installing the OPC Data Client license on runtime machines is manual: The user runs the License Manager utility (GUI or console-based), and installs the license by pointing the program to the license file stored on the disk.

If you have multiple OPC Data Client versions installed on the computer, you need to deploy the license that covers them all.

The subsequent part of the documentation describes how to automate the above procedure so that the user does not have to interact with the License Manager utility.

Registry Key

The license is stored in the registry, under the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\Licensing\QuickOPC\Multipurpose

In order to install the license, this key and anything that is beneath it, must be properly deployed to the target machine.

Note: On 64-bit systems, the license is stored in 32-bit registry. When the registry is accessed from 64-bit application, you must use the following key instead:

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Licensing\QuickOPC\Multipurpose

⚠ On 64-bit systems, if OPC Data Client is used from both 32-bit and 64-bit processes, you have to install the license into both 32-bit and 64-bit parts of the registry.

Access to the License

The license is stored in the registry. For the license to be recognized, the process (that runs OPC Data Client) must have read access to the corresponding registry keys. That is normally not a problem under "interactive" user accounts and traditional desktop applications, but it can be an issue when OPC Data Client runs as Windows service, or in hosted environments (such as IIS).

Specifically, IIS has one process for each Application Pool, and the application pool runs under a configurable user account. You need to assure that this user account has read access to the registry keys where the license is stored.

8.5.2.1 Deployment Automation

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

In order to automate the installation of the license, you first need to obtain the proper registry content that corresponds

to it.

Use the License Manager utility on the development computer, and install the license you intend to distribute. Then, start the REGEDIT utility, navigate to the registry key for the license, select it, and perform File -> Export command. You will obtain a registration file (with .REG extension) that contains the license information.

Your installation program can then use this information in various ways. For example, it can invoke the REGEDIT utility, and pass it the name of the generated .REG file on the command line. Alternatively, you can use other methods that your installation program provides for manipulating the registry, and simply have it replicate the information contained in the .REG file to the target system.

The license must not be modified in any way. Changing any of its fields (such as the Subject name) will render the license invalid.

Note: Take care on 64-bit systems, as both 32-bit and 64-bit REGEDIT applications exist, and each has a different view of the registry.

Some customers have notified us that when using Visual Studio to create an .MSI Installer, exporting the license from the registry and importing it into the Visual Studio project has modified the license fields and caused the license to be not recognized by the component. When in doubt, compare the registry content created by the License Manager against the one created by your installer. They must be precisely equal.

When a license in the registry is damaged, the License Manager may still display it as valid; but the component will not recognize it in runtime.

8.5.3 Licenses for Redistributed Material

Included Material

In This Topic

Your usage of the software created with elements from OPC Data Client must (in addition to the terms of OPC Data Client EULA) also comply with license terms of the 3rd-party material that comes with OPC Data Client. For the precise license terms, see **Licenses for 3rd-party Material (Section 2.6.1)**.

Depending on the particular subset of OPC Data Client elements that you redistribute or deploy with your application, the specific 3rd-party licensed material may not be included in your distribution/deployment. You only need to be concerned about the material that actually gets redistributed with your software. The table below explains which 3rd-party licensed material is needed when.

Material	Needed When
16x16 Free Application Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Diagone Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Discovery Icon Theme	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Fugue 16px Additional Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Fugue 16px Additional Icons 2	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.

Fugue 16px Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Iconza Black Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Krypton software	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
SharpPcap	If your distribution contains the files of the **OpcLabs.Pcap** assembly.
PacketDotNet	If your distribution contains the files of the **OpcLabs.Pcap** assembly.
Ravenna 3D Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.
Web Mini Icons	If your distribution contains the files of the **OpcLabs.BaseLibForms** assembly.

Typically, you just need to make an attribution to these somewhere in the product and/or its documentation. It always remains your responsibility, however, to comply with the precise license conditions of OPC Data Client EULA and the 3rd-party material that you redistribute.

Libraries under LGPL (such as SharpPcap, PacketDotNet) have additional requirements, such as the inclusion of the license text files.

Dependent Material

When you target .NET Standard development platform (see **Development Platforms (Section 2.3)**), OPC Data Client depends on (but does not ship with or include) various 3rd-party NuGet packages (mainly from Microsoft, and OPC Foundation), as described in the **Licensing (Section 2.6)** article. The application thay you build under .NET Standard with OPC Data Client therefore needs to contain the dependency packages (and their dependencies, recursively) as well, and you must make sure such usage complies with the licensing terms of the dependency packages.

9 Advanced Topics

9.1 Supported OPC Specifications

OPC Data Client directly supports following OPC specifications

(not in .NET Standard):

- all OPC DA (Data Access) 1.0x Specifications (Released)
- all OPC DA (Data Access) 2.0x Specifications (Released)
- all OPC DA (Data Access) 3.0x Specifications (Released)
- OPC Alarms and Events Custom Interface Standard 1.00, 1.01 and 1.10 (Released)
- all OPC Common 1.0x Specifications (Released)
- OPC Common 1.10 Specification (Draft)

OPC Data Client directly supports following OPC specifications (also in .NET Standard):

- OPC XML-DA 1.01 Specification (Released)
- OPC Unified Architecture 1.00 Specifications (Released)
- OPC Unified Architecture 1.01 Specifications (Released)
- OPC Unified Architecture 1.02 Specifications (Released)
- OPC Unified Architecture 1.03 Specifications (Released)
- OPC Unified Architecture 1.04 Specifications (Released) **(not in .NET Framework, COM or Excel, i.e. only in .NET Standard)**

Parts needed to support Data Access, Alarms & Conditions, Discovery and PubSub functionality are covered.

OPC Data Client is officially certified (by the means of its Connectivity Explorer application - because toolkits cannot be certified as such) by OPC Foundation (Certification Date: 09/24/2017, Certificate Number 1710CE0091), https://opcfoundation.org/products/view/52 . This is a renewed certification; OPC Data Client has been the first product of its kind (.NET-based client toolkit) that has ever been awarded this certification.

In This Topic

Specifically, the certified profiles are:

- Core Client Facet
- Base Client Behavior Facet
- Discovery Client Facet
- Attribute Read Client Facet
- Attribute Write Client Facet
- DataChange Subscriber Client Facet
- DataAccess Client Facet

OPC Data Client supports more facets than just those that have been certified. See **Supported OPC UA Profiles (Section 9.1.1)** for the full list.

You may also want to consult the list of **Supported OPC UA Services (Client-Server) (Section 9.1.2)** and **Supported OPC UA Security Policies (Section 9.1.3)** supported by OPC Data Client.

OPC Foundation Certified for Compliance logo is a trademark of the OPC Foundation and may be used only by written permission of the OPC Foundation. Any unauthorized use of the Certified for Compliance logo is prohibited.

Award of OPC Certification

OPC FOUNDATION

Certification Testing Laboratory Presents in the Year 2017, the

Award of OPC Certification

Certified OPC Profiles:

OPC UACTT 1.2.336.273

Core Client Facet

Base Client Behaviour Facet

Discovery Client Facet

Attribute Read Client Facet

Attribute Write Client Facet

Datachange Subscriber Client Facet

DataAccess Client Facet

to

OPC Labs

for

Successfully demonstrating OPC Excellence in Compliance, Interoperability and Robustness in the OPC Foundation European Certification Lab for the following Product:

Connectivity Explorer

***Version**: 5.50.325.1*

***Presented on**: 9/24/2017*

***Expires**: 9/30/2020*

***Serial Number**: 1710CE0091*

CERTIFIED FOR COMPLIANCE OPC FOUNDATION

Paul Hunkar
Director of Compliance & Certification

Thomas J. Burke
President

The reproduction of this official certificate is strictly prohibited without prior written consent from the OPC Foundation Certification Test Lab.

The certification has been performed using the Connectivity Explorer application, built with OPC Data Client and targeting the .NET Framework development platform.

9.1.1 Supported OPC UA Profiles

OPC Data Client supports following facets of OPC UA Profiles. Categories and profiles are sorted alphabetically.

In This Topic

Here, we can only list profiles that are actually defined by the OPC Foundation. OPC Data Client also supports features for which there are no standard profiles defined. Specifically, in the Security and Transport categories, OPC Data Client has support for additional security policies and transports other than just those listed here.

For this reason, the OPC UA Profiles list should not be understood as a list of OPC Data Client features with regard to OPC UA - it is only their subset. The purpose of this list is to allow matching of the client capabilities against a list of requirements, if the requirements are expressed in terms of OPC UA Profiles.

Client Category

Alarm & Condition

- A & C Alarm Client Facet
- A & C Address Space Instance Client Facet
- A & C Base Condition Client Facet
- A & C Base Refresh2 Client Facet
- A & C Dialog Client Facet
- A & C Enable Client Facet

Core Characteristics

- Address Space Lookup Client Facet
- Base Client Behaviour Facet
- Core 2017 Client Facet
- Core Client Facet [deprecated]
- Discovery Client Facet
- Documentation - Client
- Global Discovery Client Facet
- Multi-Server Client Connection Facet
- Subnet Discovery Client Facet

Data Access

- Attribute Read Client Facet
- Attribute Write Client Facet
- DataAccess Client Facet
- DataChange Subscriber Client Facet

Event Access

- Base Event Processing Client Facet
- Event Subscriber Client Facet
- Notifier and Source Hierarchy Client Facet

Generic Features

- Method Client Facet

Subscriber Category

Core Characteristics

- Subscriber Cyclic Limits Facet
- Subscriber Acyclic Limits Facet

UADP

- Subscriber UADP Periodic Fixed Layout Facet
- Subscriber UADP Dynamic Data or Events Layout Facet
- Subscriber UADP Flexible Layout Facet

Security Category

SecurityPolicy

(Note: More security policies are available than just those covered by these standard profiles. See the Caution box at the top of the article, and **Supported OPC UA Security Policies (Section 9.1.3)**).

- *Obsolete* SecurityPolicy - Basic256
- SecurityPolicy - None
- SecurityPolicy [A] - Aes128-Sha256-RsaOaep **(not in .NET Framework, COM or Excel, i.e. only in .NET Standard)**
- SecurityPolicy [B] - Basic256Sha256
- SecurityPolicy - Aes256-Sha256-RsaPss **(not in .NET Framework, COM or Excel, i.e. only in .NET Standard)**

TransportSecurity

- TransportSecurity - TLS 1.2

User Token

- User Token - Anonymous Facet
- User Token – Issued Token Windows Server Facet **(not in .NET Standard)**
- User Token - User Name Password Client Facet
- User Token - X509 Certificate Client Facet

Transport Category

Client-Server

(Note: More transports are available than just those covered by these standard profiles. See the Caution box at the top of the article).

- HTTPS UA-Binary
- HTTPS UA-XML **(not in .NET Standard)**
- UA-TCP UA-SC UA-Binary

PubSub

- PubSub UDP UADP *(note: IPv4 and IPv6)*
- PubSub MQTT UADP
- PubSub MQTT JSON

9.1.2 Supported OPC UA Services (Client-Server)

Sometimes, a list of supported services is used to compare SDKs, libraries or toolkits. For this reason, we provide the list of OPC UA services OPC Data Client supports, here. It should be noted, however, that comparison of supported services should not be done blindly. Always take following concerns into consideration:

- The number of services is intentionally kept low in OPC UA. The distinguishing factors lie deeper - in the various arguments that these services accept. Many services provide very rich functionality depending on the arguments they are passed, and the fact that a particular service is supported does not tell much by itself, without further qualification. You need to compare the actual capabilities of the toolkits, as seen from the user's perspective, instead.
- Many toolkits provide very thin layer over the OPC UA service calls, and that's all they do. They then leave it onto you to assemble the arguments, then call the services in the proper way and sequence, and interpret the results. That's easy for the toolkit author, but does not give you any added value. And, it's easy, but kind of useless, to expose many OPC UA services in this way, if all the toolkit does is to make it possible to invoke them. OPC Data Client has a very different approach. The individual OPC UA service are not exposed to you directly. Instead, you are given high level, task-oriented methods, that in turn call the necessary OPC UA services internally - sometimes, many of them for one OPC Data Client method.

In This Topic

The OPC UA services that OPC Data Client supports are listed below.

Discovery Service Set

- FindServers
- FindServersOnNetwork
- GetEndpoints

SecureChannel Service Set

- OpenSecureChannel
- CloseSecureChannel

Session Service Set

- CreateSession
- ActivateSession
- CloseSession

NodeManagement Service Set

(none)

View Service Set

- Browse
- BrowseNext
- TranslateBrowsePathsToNodeIds

Query Service Set

(none)

Attribute Service Set

- Read
- Write

Method Service Set

- Call

MonitoredItem Service Set

- CreateMonitoredItems
- ModifyMonitoredItems
- DeleteMonitoredItems

Subscription Service Set

- CreateSubscription
- Publish
- Republish
- DeleteSubscriptions

9.1.3 Supported OPC UA Security Policies

Client-Server

OPC Data Client supports following OPC UA security policies for Client-Server:

Name	Security Policy URI
Aes128_Sha256_RsaOaep **(not in .NET Framework, COM or Excel, i.e. only in .NET Standard)**	http://opcfoundation.org/UA/SecurityPolicy#Aes128_Sha256_RsaOaep
Aes256_Sha256_RsaPss **(not in .NET Framework, COM or Excel, i.e. only in .NET Standard)**	http://opcfoundation.org/UA/SecurityPolicy#Aes256_Sha256_RsaPss
Basic128Rsa15 **(obsolete)**	http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15
Basic256	http://opcfoundation.org/UA/SecurityPolicy#Basic256
Basic256Sha256	http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256
None	http://opcfoundation.org/UA/SecurityPolicy#None

9.2 OPC Interoperability

The OPC Data Client components have been extensively tested for OPC interoperability, with various OPC servers from many vendors, and on different computing environments. The tests also include regular participation in OPC Foundation's Interoperability Workshops, where OPC software is intensively "grilled" for interoperability.

Having tried so many different OPC servers to connect to, we have encountered different (though still correct) interpretations of the same OPC specifications, and also certain common (and less common) divergences from the specifications. OPC Data Client components do not try to "turn down" any OPC server for compliance problems. Just the opposite: Wherever possible, we have taken a "relaxed" approach to how the OPC servers are written, and allow and accept the above mentioned variations. This gives OPC Data Client even wider interoperability scope.

Based on the interoperability results (which can be viewed on OPC Foundation's web site), OPC Data Client.NET and OPC Data Client-COM have also been granted the OPC Foundations' "Self-tested for Compliance" logo. Note that in contrast with the logo title, the conditions of this logo program actually require the OPC client software be tested in presence and with cooperation of OPC Foundation's Test Lab delegate.

9.2.1 OPC-UA via UA Proxy


This functionality is not available under (or the text does not apply to) .NET Standard development platform.

The Unified Architecture (UA) is the next generation OPC standard that provides a cohesive, secure and reliable cross platform framework for access to real time and historical data and events.

The OPC Data Client-UA product allows native connections to OPC Unified Architecture servers.

OPC Data Client-Classic is not a native OPC UA client, but you can still use it to connect to OPC UA servers, using so-called UA COM Proxy (available e.g. from OC Foundation as part of OPC UA COM Interop Components).

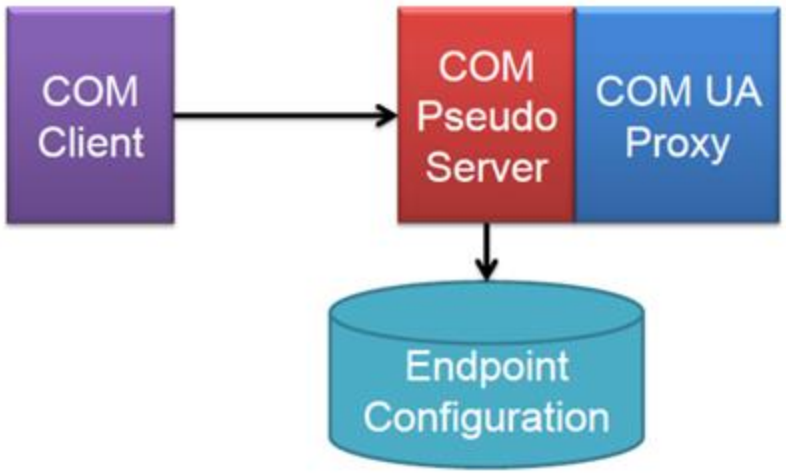
The OPC UA COM Interop Components from OPC Foundation make it possible for existing COM-based applications to

communicate with UA applications. OPC Labs is using them to add UA support to existing products. Support for OPC UA COM Interop Components is not currently provided.

The components that allows COM clients (such as OPC Data Client-COM and OPC Data Client.NET) to talk to UA server is called the UA Proxy (or UA COM Proxy). It is a DCOM server that implements the different OPC COM specifications. A COM client (OPC Data Client) uses DCOM to talk to this proxy, usually on the local machine (i.e. the same machine where the client is), and the proxy translates OPC COM calls into UA calls, and fetches the information as required from the UA server. It is a dynamic operation – all the information about address space and data values is retrieved dynamically from the UA server.

The UA COM Proxy is a DCOM server, meaning that it has its own ProgID and CLSID (class ID). However, there needs to be some mapping between the particular ProgID and a particular UA endpoint. The UA COM proxy relies on a concept of a Pseudo-server, which maps a ProgID to a specific UA endpoint, which is stored in the configuration file. The configuration tool that is made available as part of OPC UA COM Interop Components has the ability to select a UA endpoint and create one of these pseudo-servers that a COM client can then connect to. This endpoint configuration file is stored on disk in XML format.

This file also stores state information, which is necessary for replicating COM client configuration across multiple machines. If you set up a client on a particular machine, talking to a particular UA server, and do all the necessary configuration, and you then want to take the configuration and install it on multiple other machines, you can simply copy that endpoint configuration file along. The file contains the ProgID and CLSID of the pseudo-server, and the endpoint information.



The configuration tool for OPC UA COM Proxy can be found in your Start menu, under OPC Foundation → UA SDK 1.01 → UA Configuration Tool. In order to make a UA server visible to COM clients through the UA COM Proxy, select the "Manage COM Interop" tab, press the "Wrap UA Server" button, and fill in the information required by the program. You need to specify the endpoint information for a UA server (possibly using a UA discovery mechanism), the UA connection

parameters, and finally the COM pseudo-server CLSID and ProgID (the tool offers reasonable defaults). After you have finished this step, the pseudo-server appears in the list, and OPC COM clients (OPC Data Client-COM, OPC Data Client.NET) can connect to it.

9.2.2 OPC "Classic" via UA Wrapper

OPC “Classic” refers to a set of older OPC specifications that use Microsoft DCOM as their underlying communication mechanism. These are specifications such as OPC Data Access, OPC Alarms and Events, and others.

OPC Data Client-Classic products (OPC Data Client.NET and OPC Data Client-COM) allow native connections to OPC “Classic” servers.

OPC Data Client-UA is not a native OPC “Classic” client, but you can still use it to connect to OPC “Classic” servers, using so-called UA Wrapper (for OPC COM Servers) that is available from e.g. from the OPC Foundation (as part of OPC UA COM Interop Components). The OPC UA COM Interop Components from OPC Foundation make it possible for existing COM-based applications to communicate with UA applications.

9.3 Instrumentation

9.3.1 Application Configuration

OPC Unified Architecture is a rich specification that allows the participating software use different approaches and parameters to achieve their goals. As a result, there is large number of variable settings that can be tweaked to influence the performance, or achieve full interoperability.

OPC Data Client-UA attempts to determine many of these setting automatically, and also exposes the most important ones to the developer by means of properties on the various OPC Data Client-UA objects. As long as this approach works well for your application, you do not have to do anything extra to configure the UA application.

One important piece of information is the subject name of the application instance certificate. The component uses the subject name to locate the instance certificate in the certificate store, and if it is not found, it attempts to create a new instance certificate, and store it with the subject name. The default subject name is created automatically, based on the description (title) of the application assembly, or (if the preceding yields a name that is too generic, such as ‘mscorlib’ in many hosted scenarios) from other sources of information, such as the executable’s FileVersionInfo, or a main module name.

If you want to use a specific subject name for your application instance certificate, set the EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateSubject property accordingly.

Other parameters that influence the application certificate creation are EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName, ApplicationUriString, and ProductUriString.

If, for any reason, you need to change parameters that cannot be influenced using various properties on OPC Data Client-UA objects, you can decide to provide the whole set of parameters yourself. To do so, make sure that the EasyUAClient.SharedParameters.EngineParameters.ConfigurationSources property contains the UAConfigurationSources.AppConfig bit set (in the default value setting, this bit is set, together with UAConfigurationSources.Internal).

With this setting on, the whole set of parameters for OPC UA .NET Stack and SDK becomes accessible to you. The UA configuration file can be placed alongside the application’s executanle, and given the same, but a file extension of

"**.config.xml**". Such file will automatically be found and used, with no extra settings needed. If you want to place the UA configuration file elsewhere or give it a different name, you need to make several other steps yourself then:

1. Add an entry for a new configuration section, named "UAClientEngine", into the application configuration file. In Visual Studio, you typically design the contents of the application configuration file is in the **app.config** file in your project. If you do not have this file in your project, add it first. After making this change, the beginning of this file may look like this:

   UAClientEngine section declaration in application configuration file

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <configuration>
     <configSections>
       <section name="UAClientEngine"
   type="Opc.Ua.ApplicationConfigurationSection,Opc.Ua.Core" />
     </configSections>

   ...
   ```

2. Add a "UAClientEngine" element into the application configuration file. The main purpose of this element is to refer to an external XML file that contains the UA configuration. The element may look like this:

   UAClientEngine section definition in application configuration file

   ```
   <UAClientEngine>
     <ConfigurationLocation
   xmlns="http://opcfoundation.org/UA/SDK/Configuration.xsd">
       <FilePath>MyApplication.Config.xml</FilePath>
     </ConfigurationLocation>
   </UAClientEngine>
   ```

3. Create the UA configuration file with the name specified above, and fill it with settings you need. Unless specified otherwise, the file should be placed alongside the executable of your application.

Please refer to the OPC UA .NET Stack and SDK documentation for information on sections and parameters in the **app.config** file and in the UA configuration file. Following are examples of what can be changed using the UA configuration file:

- Transport configurations
- Transport quotas and timeouts
- Trace configuration

If you switch your application so that its UA configuration is placed in an external file like described above, you can then also use the "UA Configuration Tool" from OPC Foundation to configure certain parameters for the application (mainly, the security-related settings).

When the UAConfigurationSources.Internal bit is set as well, a failure to load the UA configuration from file causes OPC Data Client-UA to determine and provide all configuration parameters automatically, which is therefore the default behavior if no UA configuration file is present.

9.3.2 Event Logging

9.3.2.1 Event Logging for OPC "Classic"

Currently, OPC Data Client for OPC “Classic” specifications does not have support for event logging.

9.3.2.2 Event Logging for OPC UA

The OPC Data Client-UA does not log events on itself, but allows you to implement such functionality. It generates notifications with log entries that contain the necessary data. You can receive these notifications, and implement any kind of event logging with them in your code.

The EasyUAClient component has a static LogEntry event. This event is raised for loggable entries originating in the OPC-UA client engine and the EasyUAClient component. Each event notification carries a LogEntryEventArgs object, which contains information such as the source of the event, the event message, event type, timestamp, etc.

Note that you should not programmatically use the logging event notifications for any kind of direct actions on the OPC subsystem. For example, the notifications that inform you about the connections and disconnections to/from OPC servers have no direct relation to your OPC method calls, because the component internally merges the requests from multiple objects, and also makes optimizations to prevent unnecessary disconnections etc. The purpose of event logging is different from OPC communication.

In COM, you cannot access static members of the EasyUAClient object, and therefore you cannot directly use its LogEntry event. To get around this, create an instance of the EasyUAClientConfiguration object instead, and use its (non-static) LogEntry event to achieve the same result.

C#

```
// This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

using System;
using OpcLabs.BaseLib.Instrumentation;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class LogEntry
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            // Hook static events
            EasyUAClient.LogEntry += EasyUAClientOnLogEntry;

            try
            {
                // Do something - invoke an OPC read, to trigger some loggable entries.
                var client = new EasyUAClient();
                try
                {
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                    return;
                }

                Console.WriteLine("Processing log entry events for 1 minute...");
                System.Threading.Thread.Sleep(60 * 1000);

                Console.WriteLine("Done.");
            }
            finally
            {
                // Unhook static events
                EasyUAClient.LogEntry -= EasyUAClientOnLogEntry;
            }
        }

        // Event handler for the LogEntry event. It simply prints out the event.
        private static void EasyUAClientOnLogEntry(object sender, LogEntryEventArgs
logEntryEventArgs)
        {
            Console.WriteLine(logEntryEventArgs);
        }
    }
}
```

VB.NET

```
' This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

Imports OpcLabs.BaseLib.Instrumentation
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class LogEntry
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
```

```
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Hook static events
            AddHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry

            Try
                ' Do something - invoke an OPC read, to trigger some loggable entries.
                Dim client = New EasyUAClient()
                Try
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Exit Sub
                End Try

                Console.WriteLine("Processing log entry events for 1 minute...")
                Threading.Thread.Sleep(60 * 1000)

                Console.WriteLine("Done.")
            Finally
                ' Unhook static events
                RemoveHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry
            End Try

        End Sub


        ' Event handler for the LogEntry event. It simply prints out the event.
        Private Shared Sub EasyUAClientOnLogEntry(ByVal sender As Object, ByVal
logEntryEventArgs As LogEntryEventArgs)
            Console.WriteLine(logEntryEventArgs)
        End Sub
    End Class
End Namespace
```

C++

```
// This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include "LogEntry.h"

namespace _EasyUAClientConfiguration
{
    // CEasyUAClientConfigurationEvents

    class CEasyUAClientConfigurationEvents : public IDispEventImpl<1,
CEasyUAClientConfigurationEvents>
    {
    public:
    BEGIN_SINK_MAP(CEasyUAClientConfigurationEvents)
        // Event handlers must have the __stdcall calling convention
        SINK_ENTRY(1, 1 /*DISPID_EASYUACLIENTCONFIGURATIONEVENTS_LOGENTRY*/,
```

```
&CEasyUAClientConfigurationEvents::LogEntry)
    END_SINK_MAP()

    public:
        // Event handler for the LogEntry event. It simply prints out the event.
        STDMETHOD(LogEntry)(VARIANT varSender, _LogEntryEventArgs* pEventArgs)
        {
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(pEventArgs->ToString));
            return S_OK;
        }
    };



    void LogEntry::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // The configuration object allows access to static behavior - here, the
shared LogEntry event.
            _EasyUAClientConfigurationPtr
ClientConfigurationPtr(__uuidof(EasyUAClientConfiguration));

            // Hook events
            CEasyUAClientConfigurationEvents* pClientConfigurationEvents = new
CEasyUAClientConfigurationEvents();
            AtlGetObjectSourceInterface(ClientConfigurationPtr,
&pClientConfigurationEvents->m_libid,
                &pClientConfigurationEvents->m_iid,
                &pClientConfigurationEvents->m_wMajorVerNum,
&pClientConfigurationEvents->m_wMinorVerNum);
            pClientConfigurationEvents->m_iid =
_uuidof(DEasyUAClientConfigurationEvents);
            pClientConfigurationEvents->DispEventAdvise(ClientConfigurationPtr,
&pClientConfigurationEvents->m_iid);

            // Do something - invoke an OPC read, to trigger some loggable entries.
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));
            ClientPtr->ReadValue(L"http://opcua.demo-this.com:51211/UA/SampleServer",
L"nsu=http://test.org/UA/Data/;i=10853");

            _tprintf(_T("Processing log entry events for 1 minute...\n"));
            Sleep(60*1000);

            // Unhook events
            pClientConfigurationEvents->DispEventUnadvise(ClientConfigurationPtr,
&pClientConfigurationEvents->m_iid);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example demonstrates the loggable entries originating in the OPC-UA
```

```
// client engine and the EasyUAClient component.

type
  TClientConfigurationEventHandlers = class
    procedure OnLogEntry(
      Sender: TObject;
      sender0: OleVariant;
      eventArgs: _LogEntryEventArgs);
  end;

// Event handler for the LogEntry event. It simply prints out the event.
procedure TClientConfigurationEventHandlers.OnLogEntry(
  Sender: TObject;
  sender0: OleVariant;
  eventArgs: _LogEntryEventArgs);
begin
    WriteLn(eventArgs.ToString);
end;

class procedure LogEntry.Main;
var
  Client: EasyUAClient;
  EvsClientConfiguration: TEvsEasyUAClientConfiguration;
  ClientConfiguration: EasyUAClientConfiguration;
  ClientConfigurationEventHandlers: TClientConfigurationEventHandlers;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior - here, the
  // shared LogEntry event.
  EvsClientConfiguration := TEvsEasyUAClientConfiguration.Create(nil);
  ClientConfiguration := EvsClientConfiguration.ComServer;
  ClientConfigurationEventHandlers := TClientConfigurationEventHandlers.Create;
  EvsClientConfiguration.OnLogEntry := @ClientConfigurationEventHandlers.OnLogEntry;

  // Do something - invoke an OPC read, to trigger some loggable entries.
  Client := CoEasyUAClient.Create;
  Value := Client.ReadValue(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853');

  WriteLn('Processing log entry events for 1 minute...');
  PumpSleep(60*1000);
end;
```

Object Pascal

```
// This example demonstrates the loggable entries originating in the OPC-UA
// client engine and the EasyUAClient component.

type
  TClientConfigurationEventHandlers130 = class
    procedure OnLogEntry(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _LogEntryEventArgs);
  end;
```

```
// Event handler for the LogEntry event. It simply prints out the event.
procedure TClientConfigurationEventHandlers130.OnLogEntry(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _LogEntryEventArgs);
begin
    WriteLn(eventArgs.ToString);
end;

class procedure LogEntry.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  ClientConfigurationEventHandlers: TClientConfigurationEventHandlers130;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior - here, the
  // shared LogEntry event.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfigurationEventHandlers := TClientConfigurationEventHandlers130.Create;
  ClientConfiguration.OnLogEntry := ClientConfigurationEventHandlers.OnLogEntry;
  ClientConfiguration.Connect;

  // Do something - invoke an OPC read, to trigger some loggable entries.
  Client := CoEasyUAClient.Create;
  try
    Value := Client.ReadValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      //Exit;
    end;
  end;

  WriteLn('Processing log entry events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Finished.');
  FreeAndNil(ClientConfiguration);
  FreeAndNil(ClientConfigurationEventHandlers);
end;
```

PHP

```
// This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

class ClientConfigurationEvents {
    // Event handler for the LogEntry event. It simply prints out the event.
    function LogEntry($Sender, $E)
    {
    printf("%s\n", $E);
    }
}
```

```
// The configuration object allows access to static behavior - here, the shared
LogEntry event.
$ClientConfiguration = new COM("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration");
$ClientConfigurationEvents = new ClientConfigurationEvents();
com_event_sink($ClientConfiguration, $ClientConfigurationEvents,
"DEasyUAClientConfigurationEvents");

// Do something - invoke an OPC read, to trigger some loggable entries.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}


printf("Processing log entry events for 1 minute...");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

Visual Basic (VB 6.)

```
REM This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

' The configuration object allows access to static behavior - here, the shared LogEntry
event.
'Public WithEvents ClientConfiguration1 As EasyUAClientConfiguration

Private Sub LogEntry_Main_Command_Click()
    OutputText = ""

    Set ClientConfiguration1 = New EasyUAClientConfiguration

    ' Do something - invoke an OPC read, to trigger some loggable entries.
    Dim Client As New EasyUAClient
    On Error Resume Next
    Dim value As Variant
    value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    OutputText = OutputText & "Processing log entry events for 1 minute..." & vbCrLf
    Pause 60000
```

```
    Set ClientConfiguration1 = Nothing
    OutputText = OutputText & "Finished..." & vbCrLf
End Sub

' Event handler for the LogEntry event. It simply prints out the event.
Private Sub ClientConfiguration1_LogEntry(ByVal sender As Variant, ByVal eventArgs As
OpcLabs_BaseLib.LogEntryEventArgs)
    OutputText = OutputText & eventArgs & vbCrLf
End Sub
```

VBScript

```
Rem This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

Option Explicit

' The configuration object allows access to static behavior - here, the shared LogEntry
event.
Dim ClientConfiguration: Set ClientConfiguration =
CreateObject("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration")
WScript.ConnectObject ClientConfiguration, "ClientConfiguration_"

' Do something - invoke an OPC read, to trigger some loggable entries.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Processing log entry events for 1 minute..."
WScript.Sleep 60*1000



' Event handler for the LogEntry event. It simply prints out the event.
Sub ClientConfiguration_LogEntry(Sender, e)
    WScript.Echo e
End Sub
```

9.4 Debugging

In OPC-UA, the component attempts to detect whether a debugger is attached to the application, and if so, it uses a different, much slower keep-alive interval (this keep-alive interval is controlled by the UAClientSessionParameters.KeepAliveIntervalDebug property, and defaults to 1 day). This allows the developer to break into the debugger and examine the status of the program, without causing the OPC-UA sessions be disconnected due to the fact that keep-alives are not being exchanged while the execution is suspended.

9.5 Object Serialization

OPC Data Client.NET and OPC Data Client-UA allows you to easily store objects (and object graphs, i.e. interconnected objects) into files and streams, and also to load them back.

Two types of serialization are supported:

- Basic serialization using Serializable attribute and/or ISerializable interface. This serialization type is typically used with BinaryFormatter for storing objects in a binary format.
- XML serialization using XmlSerializer and IXmlSerializable. This serialization provides objects storage in XML format.

Practically all OPC Data Client objects (and their collections and dictionaries) can be serialized and deserialized. For example:

- You can load and store EasyDAClient**,** EasyAEClient and EasyUAClient objects (their instance properties, i.e. various parameters, are serialized).
- You can load and store parameter objects, such as DAGroupParameters.
- You can load and store arguments (and arrays of arguments) passed to functions, such as lists of items to be subscribed to (DAItemGroupArguments). This functionality can be used e.g. with storing lists of subscribed items in file, outside your application code.
- You can load and store results of browsing and querying (e.g. DANodeElementCollection, AECategoryElementCollection, and UANodeElementCollection).
- You can load and store results of reading (e.g. DAVtq, or UAAttributeData).
- You can load and store condition states (AEConditionState) and event data (AEEventData).
- You can load and store all notification data contained in event arguments, for OPC Data Access item changes (EasyDAItemChangedEventArgs), OPC Alarms and Events notifications (EasyAENotificationEventArgs), or OPC Unified Architecture monitored item changes (EasyUADataChangeNotificationEventArgs). This functionality can be used e.g. for easy logging of significant changes and events in the underlying system.

9.6 Internal Optimizations

OPC is quite “sensitive” to proper usage, with regard to efficiency. OPC Data Client performs many internal optimizations, and uses the knowledge of proper approaches and procedures to effectively handle the communication with OPC servers.

Here are some of the optimizations implemented in OPC Data Client.NET and OPC Data Client-COM:

- Wherever possible, OPC operations are performed on multiple elements at once.
- OPC items with similar update rates are put into a common OPC group.
- OPC items with similar percentage deadbands are put into a common OPC group.
- OPC items are not removed from OPC groups immediately, but only if not used for certain amount of time.
- OPC item data is held in memory, and if fresh enough, the value from memory is taken, and no OPC call is made to satisfy the Read request.
- OPC asynchronous calls are preferred over synchronous calls.
- Minimum update rates are enforced, so that the system cannot be easily overloaded.
- Multiple uses of the same OPC server or same OPC item in the user application are merged into a single request to the OPC.
- Internal queues are used to make sure that OPC callbacks cannot be blocked by user code.

Here are some of the optimizations implemented in OPC Data Client-UA:

- Wherever possible, OPC operations are performed on multiple elements at once.
- OPC UA sessions are not closed immediately, but only if not used for certain amount of time.
- Multiple uses of the same OPC server endpoint in the user application are merged into a single request.

In OPC Data Client-UA, you can call a static method EasyUAClient.CloseAll to close all unused sessions that are open to OPC-UA servers.

9.7 Failure Recovery

The OPC communication may fail in various ways, or the OPC client may get disconnected from the OPC server. Here are some examples of such situations:

- With OPC Classic, the OPC server may not be registered on the target machine – permanently, or even temporarily, when a new version is being installed.
- With OPC UA, the OPC server may not be running or registered with the discovery service on the target machine – permanently, or even temporarily, when a new version is being installed.
- The (DCOM, TCP, Web service or other) communication to the remote computer breaks due to unplugged network cable.
- The remote computer running the OPC server is shut down, or restarted, e.g. for security update.
- The configuration of the OPC server is changed, and the OPC information (item in OPC Classic, node and attribute in OPC UA) referred to by the OPC clients no longer exists. Later, the configuration could be changed again and the OPC item may reappear.
- The OPC server indicates a serious failure to the OPC client.
- The OPC Classic server asks its clients to disconnect, e.g. for internal reconfiguration.

OPC Data Client handles all these situations, and many others, gracefully. Your application receives an error indication, and the component internally enters a "wait" period, which may be different for different types of problems. The same operation is not reattempted during the wait period; this approach is necessary to prevent system overload under error conditions. After the wait period elapses, OPC Data Client will retry the operation, if still needed at that time.

All this logic happens completely behind the scenes, without need to write a single line of code in your application. OPC Data Client maintains information about the state it has created inside the OPC server, and re-creates this state when the OPC server is disconnected and then reconnected. In OPC Classic, objects like OPC groups and OPC items are restored to their original state after a failure. In OPC UA, objects like OPC subscriptions and OPC monitored items are restored to their original state after a failure.

Even if you are using the subscriptions to OPC items (in OPC Classic) or to monitored items (in OPC UA) or events, OPC Data Client creates illusion of their perseverance. The subscriptions outlive any failures; you do not have to (and indeed, you should not) unsubscribe and then subscribe again in case of error. After you receive event notification which indicates a problem, simply stay subscribed, and the values will start coming in again at some future point.

In OPC Data Client-UA, you can call a static method EasyUAClient.RetryAll to force a retrial on all objects that are in a failure state. This method terminates the waiting delay, and causes all such objects to be retried on next opportunity.

9.8 Timeout Handling

In OPC Data Client-UA, timeout values for certain OPC Data Client tasks are accessible via properties on the EasyUAClient object. The explanation of the individual timeout values is provided in the Reference documentation.

The remainder of this chapter applies mainly to OPC Data Client "Classic".

The core OPC Data Client methods (ReadItem, ReadItemValue, WriteItemValue, and their counterparts that work with multiple items at once) are all implemented as synchronous function calls with respect to the caller, i.e. they perform some operation and then return, passing the output to the caller at the moment of return. However, this does not mean that the component makes only synchronous calls to OPC servers while you are calling its methods. Instead, OPC Data Client works in background (in separate threads of execution) and only uses the method calls you make as "hints" to perform proper data collection and modifications.

Internally, OPC Data Client maintains connections to requested OPC servers and items, and it establishes the connections when you ask for reading or writing of certain OPC item. OPC Data Client eventually disconnects from these servers and items if they are no longer in use or if their count goes beyond certain limits, using its own LRU-based algorithm (Least Recently Used).

When you call any of the core OPC Data Client methods, the component first checks whether the requested item is already connected and available inside the component. If so, it uses it immediately (for reading, it may provide a cached value of it). At the same time, the request that you just made by calling the method is used for dynamic decisions on how often the item should be updated by the server etc.

If the item is not available, OPC Data Client starts a process to obtain it. This process has many steps, as dictated by the OPC specifications, and it may take some significant time. The method call you just made does not wait indefinitely until the item becomes available. Instead, it uses certain timeout values, and if the item does not become available within these timeouts, the method call returns. The connection process is totally independent of the method that was called, meaning that no problem in the connection process (even an ill-behaved server, or a broken DCOM connection) can cause the calling method to wait longer than the timeouts dictate.

The timeout values are accessible via InstanceParameters.Timeouts property on the EasyDAClient object. The explanation of the individual timeout values is provided in the Reference documentation.

Note that if the nature of the situation allows the component to determine that the item will NOT be available, the method will return earlier (before the timeouts elapse) with the proper error indication. This means that not every connection problem causes the method to actually use the full value of the timeouts. For example, when the server refuses the item because the item has an incorrect name, this error is passed immediately to the caller.

It is important to understand that even if the method call times out because the connection process was not finished in time, the connection process itself is not cancelled and may continue internally. This means that next time the same item is requested, it may be instantly available if the connection process has finished. In other words, the timeouts described above affect the way the method call is executed with respect to the caller, but do not necessarily affect at all the way the connection is performed.

When you create an EasyDAClient object, the timeout values are set to reasonable defaults that work well with reporting or computation type of OPC applications. In these applications, you know that you MUST obtain certain value within a timeout, otherwise the application will not be doing what is intended to do: e.g. the report will not contain valid data, or the computations will not be performed. When the requested item is not instantly available (for example, the server is not started yet), the application can afford delays in processing (method calls made to EasyDAClient object may block the execution for certain time). For this kind of applications, you may leave the default timeout values, or you may adjust them based on the actual configuration and performance of your system.

There is also a different kind of applications, typically an HMI screen, which wants to periodically update the values of controls displayed to the user. The screen usually contains larger number of these controls that are refreshed in a cyclic way by the application. If possible, you should use subscription-based updated for these applications, and in such case the timeouts are of much lesser importance. But, in some application the subscriptions are not practical, and you resort to periodic reading (polling). The fact that SOME data is not instantly available should not be holding the update of others. It is perfectly OK to display an indication that the data is not available momentarily, and possibly display them in some future refresh cycle when they become available. For this kind of application, you may prefer to set all the above mentioned timeout properties to lower values. This assures that the refresh loop always goes quickly over all the controls on the screen, no matter whether the data is available for them immediately, or only in a postponed fashion.

To simplify this explanation, you can also say that if you need the OPC values for further *sequential* processing, reasonably long timeouts are needed (and the defaults should serve well in most situations). If you are refreshing the data on a *cyclic* basis by polling, you will probably need to set the timeouts to lower values.

9.9 Data Types

OPC specifications have their own rules about data types, and these have to be reflected in the components and applications. This chapter describes the details of the data type handling.

9.9.1 Data Types in OPC Classic

OPC Data Access specification is based on COM, and uses Windows VARIANT type (from COM Automation) for representing data values.

Note: Some OPC servers even use certain VARIANT types that are not officially supported by Microsoft.

Microsoft .NET has a different concept, and all data is represented using an Object type. Conversions between the two are available, but not always fully possible.

In addition, not everything that can be stored in an Object can later be processed by all .NET tools and languages. Microsoft has created so-called Common Language Specification (CLS), which has certain rules and restrictions that, if followed, guarantee cross-language compatibility. Public OPC Data Client.NET components (assemblies) are fully CLS compliant, and that includes the way the data types are converted to and from OPC types.

OPC Data Client.NET converts data from COM to .NET according to following table:

COM type (VARIANT)	.NET type (Object)
VT_EMPTY	System.Object (null reference)
VT_NULL	System.DBNull (singleton class)
VT_I2	System.Int16
VT_I4	System.Int32
VT_R4	System.Single
VT_R8	System.Double
VT_CY	System.Decimal
VT_DATE	System.DateTime
VT_BSTR	System.String
VT_DISPATCH	System.Object (not tested)
VT_ERROR	System.Int32
VT_BOOL	System.Boolean
VT_VARIANT	converted type of the target VARIANT
VT_DECIMAL	System.Decimal
VT_I1	System.Int16

VT_UI1	System.Byte
VT_UI2	System.Int32
VT_UI4	System.Int64
VT_I8	System.Int64
VT_UI8	System.Decimal
VT_INT	System.Int32
VT_UINT	System.Int64
VT_ARRAY | *vtElement*	System.Array of the converted *vtElement* type

Types that are highlighted do not convert from COM to their "natural" .NET counterparts, because the corresponding .NET type is not CLS compliant. Instead, a "wider" type that is CLS compliant is chosen.

Types not listed in the above table at all are not supported.

Strings are internally represented in Unicode wherever possible.

OPC Data Client-COM is meant to be used from applications based on COM Automation, and in general, any valid VARIANT can be processed by such application. Some automation tools and programming languages, however, have restrictions on types of data they can process. If your tool does not support the data type that the OPC server is using, without OPC Data Client, you would be out of luck.

In order to provide the ability to work with widest range of OPC servers and the data types they use, OPC Data Client-COM converts some data types available in OPC. We have made a research into the data types supported by various tools, and OPC Data Client-COM uses a subset of VarType types that is guaranteed to work in most tools that are in use today (one of the most restrictive languages appears to be VBScript).

Note that the OPC Data Client-COM only converts the data that it passes to your application – either in output arguments of property accessors or methods, or input arguments in event notifications. In the opposite direction, i.e. for data that your application passes to OPC Data Client-COM, we use very "relaxed" approach, and accept the widest range of possible data types.

OPC Data Client-COM converts data from OPC Data Access according to following table:

VARTYPE in OPC Data Access	**VARTYPE In OPC Data Client-COM**
VT_EMPTY	VT_EMPTY
VT_NULL	VT_NULL
VT_I2	VT_I2
VT_I4	VT_I4
VT_R4	VT_R4
VT_R8	VT_R8
VT_CY	VT_CY
VT_DATE	VT_DATE
VT_BSTR	VT_BSTR

VT_DISPATCH	VT_DISPATCH
VT_ERROR	VT_R8
VT_BOOL	VT_BOOL
VT_VARIANT	VT_VARIANT
VT_DECIMAL	VT_DECIMAL
VT_I1	VT_I2
VT_UI1	VT_UI1
VT_UI2	VT_I4
VT_UI4	VT_R8
VT_I8	VT_R8 (may lose precision)
VT_UI8	VT_R8 (may lose precision)
VT_INT	VT_I4
VT_UINT	VT_R8
VT_ARRAY | *vtElement*	VT_ARRAY | VT_VARIANT(*vtElement*) *(see note below)

Types that are highlighted are converted to a different data type. If a precise match does not exist, a "wider" type is chosen.

Types not listed in the above table at all are not supported.

Strings are internally represented in Unicode wherever possible.

9.9.2 Data Types in OPC-UA

OPC Unified Architecture has its own type system, based on a set of built-in types, and a type graph that supports inheritance and creation of structured types.

The types in Microsoft .NET are different, and all data is represented using an Object type and its derivatives. Conversions between the two are available, but not always fully possible.

In addition, not everything that can be stored in an Object can later be processed by all .NET tools and languages. Microsoft has created so-called Common Language Specification (CLS), which has certain rules and restrictions that, if followed, guarantee cross-language compatibility. Public OPC Data Client components (assemblies) are fully CLS compliant, and that includes the way the data types are converted to and from OPC types.

OPC Data Client-UA converts data from OPC-UA to .NET according to following table:

OPC-UA type	**.NET type (Object)**
Boolean	System.Boolean
Byte	System.Byte
ByteString	System.Byte[]
DataValue	OpcLabs.EasyOpc.UA.UAAttributeData

DateTime	System.DateTime (UTC)
Double	System.Double
ExpandedNodeId	OpcLabs.EasyOpc.UA.AddressSpace.UANodeId
ExtensionObject	when the OPC UA Complex Data extension is enabled (the default): UAGenericObject Class when the OPC UA Complex Data extension is disabled: OpcLabs.EasyOpc.UA.UAExtensionObject
Float	System.Single
Guid	System.Guid
Int16	System.Int16
Int32	System.Int32
Int64	System.Int64
LocalizedText	System.String (see note)
NodeId	OpcLabs.EasyOpc.UA.AddressSpace.UANodeId
QualifiedName	OpcLabs.EasyOpc.UA.AddressSpace.UAQualifiedName
SByte	System.Int16
StatusCode	OpcLabs.EasyOpc.UA.UAStatusCode
String	System.String
UInt16	System.Int32
UInt32	System.Int64
UInt64	System.Decimal
XmlElement	System.Xml.XmlElement
array of *elementType*	System.Array of the converted *elementType*

Note: LocalizedText is converted to its text part.

Types that are highlighted do not convert from OPC-UA to their "natural" .NET counterparts, because the corresponding .NET type is not CLS compliant. Instead, a "wider" type that is CLS compliant is chosen.

Types not listed in the above table at all are not supported.

Strings are internally represented in Unicode.

In OPC Data Client-UA for COM, the same types as in OPC Data Client-UA for .NET are used. The type conversions between COM and .NET and performed by the Microsoft interoperability layer. Reference to Microsoft documentation for correspondences between the .NET and COM types.

9.10 Multithreading and Synchronization

The EasyDAClient, EasyAEClient and EasyUAClient objects and all their related helper objects are thread-safe.

In OPC Data Client.NET and OPC Data Client-UA, if you are hooking to the event notifications provided by the EasyDAClient , EasyAEClient or EasyUAClient component, or processing these notifications in your callback methods, make sure that you understand how the component generates these events and what threading issues it may involve. This is how it works:

The EasyDAClient, EasyAEClient or EasyUAClient object has a SynchronizationContext property. This property can either be set to a null reference, or to an instance of System.Threading.SynchronizationContext class. When the SynchronizationContext is set to a null reference, EasyDAClient, EasyAEClient or EasyUAClient calls any event handlers or callback methods on its own internal thread, which is possibly different from any threads that you have in your application. When the SynchronizationContext is set to a concrete instance, the synchronization model implemented by this object is used. The EasyDAClient, EasyAEClient or EasyUAClient then typically uses the Post method of this synchronization context to invoke event handlers in your application.

When the EasyDAClient, EasyAEClient or EasyUAClient object is initially created, it attempts to obtain the synchronization context from the current thread (the thread that is executing the constructor). As a result, if the thread constructing the EasyDAClient, EasyAEClient or EasyUAClient object has a synchronization context associated with it, it will become the value of the SynchronizationContext property. Otherwise, the SynchronizationContext property will be set to a null reference. This way, the synchronization context propagates from the constructing thread.

Access to Windows Forms controls or WPF controls is not inherently thread safe. If you have two or more threads manipulating the state of a control, it is possible to force the control into an inconsistent state. Other thread-related bugs are also possible, such as race conditions and deadlocks. It is important to make sure that access to your controls is performed in a thread-safe way. Thanks to the mechanism described above, this is done automatically for you, provided that the constructor of the EasyDAClient, EasyAEClient or EasyUAClient object is called on the form's main thread, as is the case if you place the EasyDAClient , EasyAEClient or EasyUAClient component on the form's design surface in Visual Studio. This works because by default, Windows Forms set the synchronization context of the form's main thread to a properly initialized instance of System.Windows.Forms.WindowsFormsSynchronizationContext object.

Similarly, Windows Presentation Foundation (WPF) applications use System.Windows.Threading.DispatcherSynchronizationContext to achieve the same thing.

If your application is not based on the above frameworks, or is using them in an unusual way, you may have to take care of the synchronization issues related to event notification yourself, either by directly coding the synchronization mechanism, or by implementing and using a class derived from System.Threading.SynchronizationContext.

In OPC Data Client.NET and OPC Data Client-UA, objects in the Forms namespaces follow the general Windows Forms rules and conventions, meaning that any public static (Shared in Visual Basic) type members are thread-safe. Any instance members are not guaranteed to be thread safe.

In OPC Data Client-COM and OPC Data Client-UA for COM, if you are hooking to the event notifications provided by the EasyDAClient, EasyAEClient or EasyUAClient component, make sure that you understand how the component generates these events and what threading issues it may involve. The event notifications generated by EasyDAClient, EasyAEClient or EasyUAClient object originate from a thread that may be (and generally is) different from the thread that you used to create an instance of the object or call its methods. The code in your event handler must be prepared to deal with it.

A typical issue that arises is that access to Windows controls is not inherently thread-safe, and should be done from a dedicated thread only. It is important to make sure that access to your controls is performed in a thread-safe way. This typically involves setting up some communication mechanism between the event handler code, and a thread dedicated to handling the user interface of your application.

9.11 64-bit Platforms

You can create 32-bit or 64-bit, or platform-independent applications with OPC Data Client. 32-bit applications can also

run on 64-bit systems. 64-bit applications can only run on 64-bit systems. Normally, you will target your application to "Any CPU", and the same code will then run on both x86 and x64 platforms.

Supported platforms are x86 (i.e. 32-bit), and x64 (i.e. 64-bit). The product has not been tested and is not supported on the Itanium platform (IA-64).

When you use OPC Data Client for COM development, the components are compiled using the "just-in-time" by the .NET runtime for either 32-bit or 64-bit execution, depending on the bitness of the process that loads the OPC Data Client components.

9.11.1 32-bit and 64-bit Code

OPC Data Client.NET (for OPC Classic; not OPC Data Client-UA) assemblies contain certain parts in native 32-bit code (for x86 platform) and in native 64-bit code (for x64 platform). OPC Data Client.NET uses a special technique to load the so-called mixed mode assemblies (assemblies that contain both managed and native code) for a proper processor architecture.

OPC XML-DA support in OPC Data Client is written fully in managed code, and is not using the mixed mode assemblies.

Any application built with OPC Data Client.NET or OPC Data Client-UA assemblies can also be run on 32-bit Windows, or on 64-bit Windows for x64 processors. By default, such applications run as 32-bit processes or 32-bit machines and as 64-bit processes on 64-bit machines. You can also build your code specifically for x86 or x64 platform, if you have such need.

9.11.2 Classic OPC on 64-bit Systems


This functionality is not available under (or the text does not apply to) .NET Standard development platform.

Classic OPC is based on Microsoft COM/DCOM, which has originally been designed for 32-bit world, and later ported to and enhanced for 64-bit systems. There are several issues with COM/DCOM on 64-bit systems and some additional issues specific to OPC.

The most notable issue is the fact that browsing for OPC servers does not always fully work between 32-bit and 64-bit worlds. This is because the old OPCEnum component (provided earlier by OPC Foundation and still being installed with some servers) runs in 32-bit process and only enumerates 32-bit OPC servers. Consequently, native 64-bit OPC servers may be "invisible" for browsing from 32-bit OPC clients, although it may still be possible to connect to them, provided that the OPC client has OPC server's ProgID or CLSID.

Depending on how the OPC server registers itself, the opposite behavior is also possible: The server might be visible in the list of servers, but no connection to it possible from 64-bit processes.

Local out-of-process server (.EXE-s) are, in generable, usable across bitness boundaries (32-bit servers from 64-bit clients and vice versa), but proper setup and registration is needed (see above), which is not always the case. Local in-process servers (.DLL-s), which are relatively rare, cannot be directly used across bitness boundaries. A possible workaround is to use COM+ configuration and set them to run in a surrogate process.


Some OPC servers have a licensing approach that differentiates between local and remote usage of the OPC server, and require you to obtain a different (typically more expensive) license for the remote usage. Quite often these are 32-bit OPC servers, and their internal logic that detects the type of usage is flawed in such a way that a local usage of this 32-bit OPC server from an OPC client running in a 64-bit process is considered a "remote" usage. This results in a licensing error from the OPC server when you try to connect to it from an OPC client built with OPC Data Client and run on a 64-bit computer. It may seem that this a OPC Data Client fault, because other OPC clients work well - but that is because they are typically 32-bit clients only. The remedy is to build your OPC client for the **"x86"** target

(instead of **"AnyCPU"** or **"x64"**), or to use the **ClrFlags** tool from Microsoft to modify your executable accordingly - so that it runs as a 32-bit process.

9.12 Prerequisites Boxing

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

OPC Data Client uses “boxing”, a special technique based on file system and registry virtualization, which allows is to transparently load and use dependent software that would otherwise have to be installed (and uninstalled separately). With boxing, OPC Data Client libraries can simply be placed on disk, registered (for COM) and referenced from .NET or COM code, without installing other software.

The boxing encompasses following software:

1. Microsoft Visual C++ 2015 Redistributable (x86, x64). From Microsoft Corporation.
   Needed because parts of our software are written in managed C++ (for OPC Classic only).
2. OPC Core Components 3.00 Redistributable (x86 or x64). From OPC Foundation.
   Provides proxies/stubs that allow OPC "Classic" components to communicate, and a server enumeration facility (OPCEnum) for a discovery of OPC "Classic" servers.
3. OPC UA Certificate Generator utility.

In order to improve application startup speed, conserve memory, or in edge cases where the “boxing” approach fails, a user can still install any of the above prerequisites, and OPC Data Client will use the software installed on the disk. The download links for the individual prerequisites for such case are installed with OPC Data Client, under the Redist folder (“Redistributables” from the program group under Start menu).

9.13 Version Isolation

Different product versions can be installed on the same computer in parallel. For example, version 5.12 can be installed together with version 5.02 or even version 3.03.

The above rule only applies to the effects of the installation on the target system using the **Setup Program (Section 3.1.1)** – things like Start menu folders and shortcuts, files installed into the Program files directory, etc. As to the actual component assemblies, you can have many versions and even different builds and revisions on the same computer, and they will all work in mutual isolation.

The simulation OPC server (for OPC “Classic”) is not subject to the versioning rules described above for the OPC Data Client product. Just one instance of simulation OPC server can exist on a computer. Features are being added to the simulation OPC server with newer versions of OPC Data Client. Always install the simulation OPC server from the latest OPC Data Client version in order to guarantee that all examples are functional.

10 Best Practices

This section describes best practices that aid in developing applications and libraries that use the OPC Data Client library.

All information described in this section is merely a set of guidelines to aid development. These guidelines do not constitute an absolute truth. They are patterns that we found helpful; not rules that should be followed blindly. There are situations where certain guidelines do not apply. It is up to each individual developer to decide if a certain guideline makes sense in each specific situation.

The guidelines in this section are listed in no particular order.

10.1 Do not write any code for OPC failure recovery

As described in Failure Recovery, problems in communications with target OPC server are automatically recognized, and the component performs the recovery automatically. Attempts to duplicate this functionality in your code will interfere with this mechanism and won't work.

It is fine (and indeed, suggested) to catch the errors and handle them appropriately. But the error handling should not include attempts to remedy the situation. If you are doing "one-shot" operations (such as reads, writes etc.), it is OK to re-try them later. If you are subscribing, you should not attempt to un-subscribe in case of a problem. The component does this by itself. Making a subscription in OPC Data Client means an intent to subscribe and stay subscribed whenever possible, until you explicitly unsubscribe, and the component works in the background to fulfill this. This is avalid even if the very first (initial) subscription attempt fails; the component will keep re-trying internally, unless you unsubscribe.

In OPC Data Client-UA, you can call a static method EasyUAClient.CloseAll to close all unused sessions that are open to OPC-UA servers.

10.2 With single-operation synchronous methods, only catch OpcException or UAException

Methods that operate on a single element, e.g. an OPC Data Access or OPC XML-DA item, throw an OpcException exception for all OPC-related errors. In OPC-UA, UAException is used for the same purpose. The only other exceptions that can be thrown are program execution-related errors (such as OutOfMemoryException, or StackOverflowException), and argument validation exceptions, such as ArgumentException.

When you catch OpcException or UAException, you are completely safe that all OPC-related situations that you cannot control will be covered. This includes errors related to arguments that can only be discovered in run-time and depend on the state of environment. For example, an invalid OPC-DA item ID does not throw ArgumentException, but still an OpcException. So, even if something changes in the target OPC server, you can handle it by catching OpcException (or UAException). For more information, see Error Handling.

By catching exceptions other than OpcException (or UAException) in this case, you could improperly hide programming or catastrophic errors that should not be handled in this way.

The SubscribeXXXX and UnsubscribeXXXX methods, such as EasyDAClient.SubscribeItem, EasyDAClient.UnsubscribeItem, or EasyAEClient.SubscribeEvents, work asynchronously. This rule therefore does not apply to them; instead, see **Do not catch any exceptions with asynchronous or multiple-operation methods (Section 10.3)**.

10.3 Do not catch any exceptions with asynchronous or multiple-operation methods

Asynchronous methods (such as SubscribeXXXX), or methods that work on multiple elements at once (such as EasyDAClient.ReadMultipleItems) only throw program execution-related errors (such as OutOfMemoryException, or StackOverflowException), and argument validation exceptions, such as ArgumentException. The OPC-related exceptions are reported in a different way: By event notifications or callbacks (for asynchronous methods), or in the OperationResult.Exception properties of the methods return value array (for multiple-operation methods).

By catching exceptions in this case, you could hide programming or catastrophic errors that should not be handled in this way. For more information, see Errors and Multiple-Element Operations.

10.4 Always test the Exception property before accessing the actual result (Value, Vtq, or AttributeData property)

Whenever OPC Data Client return an OperationResult object, or an object derived from it, such as ValueResult, DAVtqResult or UAAttributeDataResult, the actual result properties are only valid when the result represents a success, i.e. its Exception property is a null reference. You should therefore always test the Exception property for null-ness first. Only the properties available in the base OperationResult class are always accessible.

The same applies to the Exception property in event notifications, for all event arguments that are derived from the OperationEventArgs class. For example, the Exception property must be checked in event handlers for the EasyDAClient.ItemChanged event (in EasyDAItemChangedEventArgs), and in event handlers for the EasyUAClient.DataChangeNotification event (in EasyUADataChangeNotificationEventArgs).

If you attempt to get other properties while OperationResult.Exception or OperationEventArgs.Exception is not null, the outcome is undefined; most likely an exception will be thrown, but that would be a serious exception indicating a programming error. For more information, see Error Handling.

10.5 Always test the HasValue property before accessing DAVtq.Value or UAAttributeData.Value

The controlling member in the DAVtq class (OPC Data Access Value/Timestamp/Quality) is the Quality property. When the quality is not sufficient, the Value property is not available (and OPC Data Client assures that is a null reference in this case). Getting the Value property when it is not available makes little sense.

The same applies to the UAAttributeData class (in OPC Unified Architecture) with regard to its StatusCode and Value properties.

It is commonly said that the value is available whenever the quality (or status code) is not Bad, but precisely speaking, this is not true, as there are some border cases (per OPC specifications) in which this rule does not apply. Your best choice is to test the HasValue property, which has the precise, OPC-compliant functionality built in. Only if HasValue is true, the Value property contains valid data.

For more information, see Data Objects.

10.6 Use multiple-operation methods instead of looping

Due to way OPC internally works, it is significantly more efficient to perform more operations at once. Whenever your application logic allows it, use methods with the word **Multiple** in their name, i.e. methods that work on multiple elements (such as OPC items) at once, and try to group together a bigger number of operations. This approach gives much better performance.

10.7 Do not block inside OPC event handlers or callback methods

You should not block for excessive or significant time inside the OPC event handlers (such as an event handler for EasyDAClient.ItemChanged) or callback methods. Doing so delays further event processing, and in serious cases, can lead to excessive memory usage and eventually loss of events.

OPC Data Client internally queues the events, and therefore any blocking in your code cannot have negative influence on the target OPC server. In the long-term, however, the average processing time for your event handlers or callback methods must be equal to or less than the average rate the OPC events are incoming.

10.8 Use generic or type-safe access wherever possible

OPC "Classic" represents data using OLE Automation VARIANT-s, which are transformed to .NET (CLR) objects. As such, a value of any type can appear in place of OPC data. Your code typically knows that certain piece of data (such as OPC item) is of specific type (such as 32-bit signed integer), and expects that the value indeed is of this type.

Instead of trying to put the type conversions and safety checks in your code, you should use the features already available in OPC Data Client.NET.

If your language supports generics and you feel comfortable working with them, it is the best choice to use generic types and generic access (see e.g. Generic Types and Generic Access). The second best choice is type-safe access, which provides similar functionality, but without use of generics (see e.g. Type-safe Access).

10.9 Use the state instead of handles to identify subscribed entities

The handles returned from any EasyXXClient.SubscribeXXXX methods are only meant for use as input arguments to other methods that manipulate the existing subscriptions – such as EasyXXClient.ChangeXXXX or EasyXXClient.UnsubscribeXXXX.

If you need to identify the subscribed entities inside event handlers, use the 'state' argument that can be passed to EasyXXClient.SubscribeXXXX methods (or a State property on XXXXArguments objects). You can pass any type of object in the state, as it is of type System.Object (and then cast the System.Object back to the original type in the event handler).

Why shouldn't the handles be used? There are multiple reasons for that:

1. The SubscribeXXXX methods are asynchronous. There is no guaranteed relation between the moment the SubscribeXXXX method returns, and the moment when events start flowing into your event handler. The events can start flowing before the methods returns. Or, they can start flowing after the method returns, but before you get a

chance to process the returned handles, and store them somewhere where the event handler can access them.
2. Similarly, UnsubscribeXXXX methods are asynchronous as well, and therefore you cannot know precisely when any event handle became invalid. You may (temporarily) receive events for a previously returned handles, even after the UnsubscribeXXXX method finishes.
3. With handles, you will need to set up and properly maintain an additional data structure – something that keeps a correspondence between the handles returned (whose values you do not know in advance), and your application data that relate to these handles. Typically, this data structure is a dictionary. Maintaining and accessing this additional data structure means extra code, and inefficient access (application data must be looked up by handle from inside the event handler).
4. Using the handles returned from SubscribeXXXX in an event handler requires coordination of the access to a data structure where the handles are stored – i.e. an inter-thread synchronization. This leads to complicated and often buggy code, with risk of deadlocks or invalid data being accessed.

All these disadvantages can simply be overcome by passing the application data needed into the SubscribeXXXX method as a state object, and then casting the State to the appropriate type and using directly inside the event handler.

Note that earlier versions passed the handle to your code in the event arguments, making it relatively easy to make a mistake of using them from inside the handler. This is no longer the case with the current version, and the likeliness of violating this best practice is therefore much lower. With extra effort, you might be able to somehow "stick" the handle into the event notification, possibly by back-filling the State property after the subscription was made. It is strongly recommended that you do not do that for the reasons explained above.

C#

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// integer.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void StateAsInteger()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification +=
ClientOnDataChangeNotification_StateAsInteger;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
```

```
                        "nsu=http://test.org/UA/Data/;i=10845", 1000)
                        {State = 1},    // An integer we have chosen to identify the
subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000)
                        {State = 2},    // An integer we have chosen to identify the
subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                        {State = 3}     // An integer we have chosen to identify the
subscription
                });

            for (int i = 0; i < handleArray.Length; i++)
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray[i]);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Done.");
        }

        static void ClientOnDataChangeNotification_StateAsInteger(object sender,
EasyUADataChangeNotificationEventArgs eventArgs)
        {
            // Obtain the integer state we have passed in.
            var stateAsInteger = (int) eventArgs.Arguments.State;

            // Display the data
            if (eventArgs.Succeeded)
                Console.WriteLine("{0}: {1}", stateAsInteger, eventArgs.AttributeData);
            else
                Console.WriteLine("{0} *** Failure: {1}", stateAsInteger,
eventArgs.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item
' and display each change, identifying the different subscription by an
' integer.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub StateAsInteger()
```

```
            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
ClientOnDataChangeNotification_StateAsInteger

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10845", 1000) _
                         With {.State = 1}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10853", 1000) _
                         With {.State = 2}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10855", 1000) _
                        With {.State = 3} _
                } _
             ) ' An integer we have chosen to identify the subscription

            For i As Integer = 0 To handleArray.Length - 1
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray(i))
            Next i

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Done.")
        End Sub

        Private Shared Sub ClientOnDataChangeNotification_StateAsInteger(ByVal sender
As Object, ByVal eventArgs As EasyUADataChangeNotificationEventArgs)
            ' Obtain the integer state we have passed in.
            Dim stateAsInteger As Integer = CInt(eventArgs.Arguments.State)

            ' Display the data
            If eventArgs.Succeeded Then
                Console.WriteLine("{0}: {1}", stateAsInteger, eventArgs.AttributeData)
            Else
                Console.WriteLine("{0} *** Failure: {1}", stateAsInteger,
eventArgs.ErrorMessageBrief)
```

```
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// integer.

type
  TIntegerClientEventHandlers124 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TIntegerClientEventHandlers124.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
var
  stateAsInteger: Integer;
begin
  // Obtain the integer state we have passed in.
  stateAsInteger := eventArgs.Arguments.State;
  if eventArgs.Succeeded then
      WriteLn(stateAsInteger, ': ', eventArgs.AttributeData.ToString)
  else
      WriteLn(stateAsInteger, ' *** Failure: ', eventArgs.ErrorMessageBrief);
end;


class procedure SubscribeMultipleMonitoredItems.StateAsInteger;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TIntegerClientEventHandlers124;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TIntegerClientEventHandlers124.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;

  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
```

```
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments1.SetState(1);  // An integer we have chosen to identify the
subscription

  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2.SetState(2);  // An integer we have chosen to identify the
subscription

  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3.SetState(3);  // An integer we have chosen to identify the
subscription

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// integer.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Obtain the integer state we have passed in.
        $stateAsInteger = $E->Arguments->State;
        if ($E->Succeeded)
            printf("%d: %s\n", $stateAsInteger, $E->AttributeData);
        else
            printf("%d *** Failure: %s\n", $stateAsInteger, $E->ErrorMessageBrief);
    }
}


// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;

$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments1->SetState(1);  // An integer we have chosen to identify the
subscription

$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2->SetState(2);  // An integer we have chosen to identify the
subscription

$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3->SetState(3);  // An integer we have chosen to identify the
subscription

$arguments[0] = $MonitoredItemArguments1;
```

```
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;

$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

PHP: We have found no way of retrieving the original PHP object from State (COM VARIANT). Use other data types (such as integers) for State in PHP: **Examples - OPC Unified Architecture - Identify subscriptions by an integer state (Section 13.2.13.24)** .

C#

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// object.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        class CustomObject
        {
            public CustomObject(string name)
            {
                Name = name;
            }

            public string Name { get; private set; }
        }


        public static void StateAsObject()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
```

```
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification +=
ClientOnDataChangeNotification_StateAsObject;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000)
                        {State = new CustomObject("First")},    // A custom object that
corresponds to the subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000)
                        {State = new CustomObject("Second")},    // A custom object
that corresponds to the subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                        {State = new CustomObject("Third")},    // A custom object that
corresponds to the subscription
                });

            for (int i = 0; i < handleArray.Length; i++)
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray[i]);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Done.");
        }

        static void ClientOnDataChangeNotification_StateAsObject(object sender,
EasyUADataChangeNotificationEventArgs eventArgs)
        {
            // Obtain the custom object we have passed in.
            var stateAsObject = (CustomObject) eventArgs.Arguments.State;

            // Display the data
            if (eventArgs.Succeeded)
                Console.WriteLine("{0}: {1}", stateAsObject.Name,
eventArgs.AttributeData);
            else
                Console.WriteLine("{0} *** Failure: {1}", stateAsObject.Name,
eventArgs.ErrorMessageBrief);
        }
    }
```

```
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item
' and display each change, identifying the different subscription by an
' object.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Class CustomObject
            Public Sub New(ByVal name As String)
                _Name = name
            End Sub
            Public ReadOnly Property Name As String
                Get
                    Return _Name
                End Get
            End Property
            Private ReadOnly _Name As String
        End Class

        Public Shared Sub StateAsObject()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
ClientOnDataChangeNotification_StateAsObject

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10845", 1000) _
                         With {.State = New CustomObject("First")}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10853", 1000) _
                         With {.State = New CustomObject("Second")}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10855", 1000) _
                        With {.State = New CustomObject("Third")} _
                } _
            ) ' A custom object that corresponds to the subscription

            For i As Integer = 0 To handleArray.Length - 1
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray(i))
```

```
            Next i

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Done.")
        End Sub

        Private Shared Sub ClientOnDataChangeNotification_StateAsObject(ByVal sender As
Object, ByVal eventArgs As EasyUADataChangeNotificationEventArgs)
            ' Obtain the custom object we have passed in.
            Dim stateAsObject As CustomObject = CType(eventArgs.Arguments.State,
CustomObject)

            ' Display the data
            If eventArgs.Succeeded Then
                Console.WriteLine("{0}: {1}", stateAsObject.Name,
eventArgs.AttributeData)
            Else
                Console.WriteLine("{0} *** Failure: {1}", stateAsObject.Name,
eventArgs.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// object.

type
  TCustomObject = class
    Name: string;
    constructor Create(name: string);
  end;

constructor TCustomObject.Create(name: string);
begin
  Self.Name := name;
end;

type
  TObjectClientEventHandlers125 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;
```

```
procedure TObjectClientEventHandlers125.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
var
  stateAsObject: TCustomObject;
begin
  // Obtain the custom object we have passed in.
  stateAsObject := TCustomObject(INT_PTR(eventArgs.Arguments.State));
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(stateAsObject.Name, ': ', eventArgs.AttributeData.ToString)
  else
      WriteLn(stateAsObject.Name, ' *** Failure: ', eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleMonitoredItems.StateAsObject;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TObjectClientEventHandlers125;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  CustomObject1, CustomObject2, CustomObject3: TCustomObject;
begin
  CustomObject1 := TCustomObject.Create('First');
  CustomObject2 := TCustomObject.Create('Second');
  CustomObject3 := TCustomObject.Create('Third');

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TObjectClientEventHandlers125.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;

  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments1.SetState(INT_PTR(CustomObject1));  // A custom object that
corresponds to the subscription

  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
```

```
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2.SetState(INT_PTR(CustomObject2));  // A custom object that
corresponds to the subscription

  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3.SetState(INT_PTR(CustomObject3));  // A custom object that
corresponds to the subscription

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
  FreeAndNil(CustomObject1);
  FreeAndNil(CustomObject2);
  FreeAndNil(CustomObject3);
end;
```

11 Tools and Online Services

Besides offering components to integrate within your project, OPC Data Client also offers several tools and online service that can assist you in development. They include:

- **License Manager (Section 11.1)** - GUI, and **LMConsole Utility (License Manager Console) (Section 11.1.1)**
- **Connectivity Explorer (Section 11.2)**
- **The Launcher application (Section 11.3)**
- **OpcCmd Utility (Section 11.4)**
- **Demo Servers and Publishers (Section 11.5)** (some installable locally, some are online)

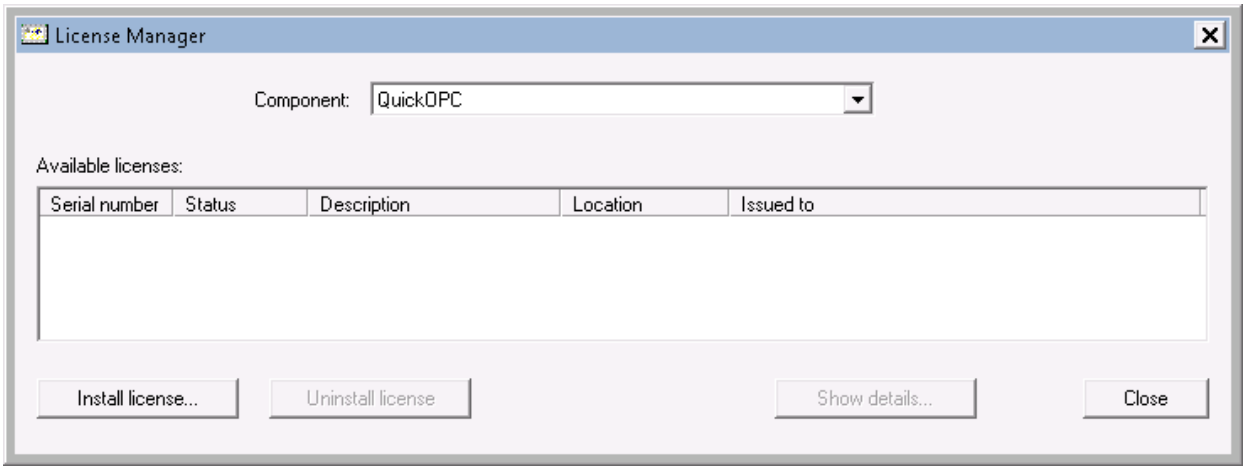
11.1 License Manager

The **License Manager** is a utility that allows you to install, view and uninstall licenses for OPC Data Client and other products which use the same licensing scheme. The **License Manager** is used:

- **On the development computer**, if you are developing on Windows and are targeting .NET Framework, COM or Excel development platform.
- **On the production (runtime) computer**, if your project is for Windows, and uses the license located in **Registry License Store (Section 8.5.2)** (as opposed to a license located e.g. in the **Managed Resource License Store (Section 8.5.1)**).

When you are targeting solely the .NET Standard development platform, you do not need to use the **License Manager** utility.

In order to install a license, invoke the License Manager application (from the Start menu or the **Launcher** application), select QuickOPC in the **Component** drop-down, and press the Install license button. Then, point the standard **Open File** dialog to the license file (typically with .BIN or .TXT extension) provided to you by your vendor.



If you have purchased some one or more of the **Product Options (Section 12)** with the base product, you may have been provided with additional license files for the product option(s). In such case, you need to install them using **License Manager** as well, under the same **Component**.

> ⚠ Note: You need administrative elevated privileges to successfully install and uninstall licenses.

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

11.1.1 LMConsole Utility (License Manager Console)

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

In This Topic

The **LMConsole** utility is a command-line equivalent of the **License Manager**, for scripting/automation (e.g. embedded installations), and folks that dislike GUIs in general. The command-line switches are described here, and a brief help is also available from the utility itself.

The utility is installed in the **Bin** subdirectory under the product folder.

Command Line Syntax

Following table summarizes the command-line switches available:

Switch	Description
-u *string* **--uninstall** *string*	Uninstall license
-s *string* **--show** *string*	Show license details
-q **--query**	Query available licenses
-l **--list**	List components
-i *string* **--install** *string*	Install licenses. The parameter is the license key file name, including extension (.BIN or .TXT, typically).
-c *string* **--component** *string*	Component name
-- **--ignore_rest**	Ignores the rest of the labeled arguments following this flag.
--version	Displays version information and exits.
-h **--help**	Displays usage information and exits.

The switches are combined to achieve the desired effect.

Typical Commands

You will typically need following LMConsole commands:

Task	Command
Query available OPC Data Client licenses	**LMConsole --component QuickOPC --query**
Install OPC Data Client license (from a file)	**LMConsole --component QuickOPC --install** *fileName*
Show OPC Data Client license details	**LMConsole --component QuickOPC --show Multipurpose**
Uninstall OPC Data Client license	**LMConsole --component QuickOPC --uninstall Multipurpose**

The license for the base product uses ID "Multipurpose". Licenses for specific **Product Options (Section 12)** have different IDs, documented with each particular product option.

Exit Codes

The LMConsole utility returns following exit codes:

Exit Code	Description
0	The command has completed successfully.
1	Command parsing error.
2	Command processing error.

11.2 Connectivity Explorer

The **Connectivity Explorer** application (for Windows only) allows the user to navigate through hierarchy of OPC Data Access and OPC Unified Architecture servers and data nodes. The user can subscribe to data changes, and view the results live.

Features of Connectivity Explorer include:

- Selected data can be transferred to Excel, where it continues to stay subscribed, giving cell values that are dynamically updated with live OPC data.
- Individual points, or a selected set of points, can be read or written to.
- Point parameters can be edited before a subscription, read or write is made.
- Multiple points can be easily subscribed to at once, using the "Bulk Add" command.
- The live view can be turned online/offline with a single push of a button.
- Single cell, or any range of cells can be copied to clipboard and pasted to other programs.
- Data can be copied to clipboard either statically, using their current value, or (for Excel) dynamically, using RTD function (see **Excel Option (Section 12.1)**).
- The currently displayed list of points can be saved to an XML file, or retrieved from a file.
- Various connectivity options can be configured, and the configuration can be in turn used when the same data is accessed by Excel.

The Connectivity Explorer uses the **Connectivity Model (Section 6.2)** as basis for its operation. Please refer to the

Connectivity Model documentation whenever necessary.

11.2.1 Main Window and Its Parts

The main window appears after the Connectivity Explorer is started. It contains three subwindows:

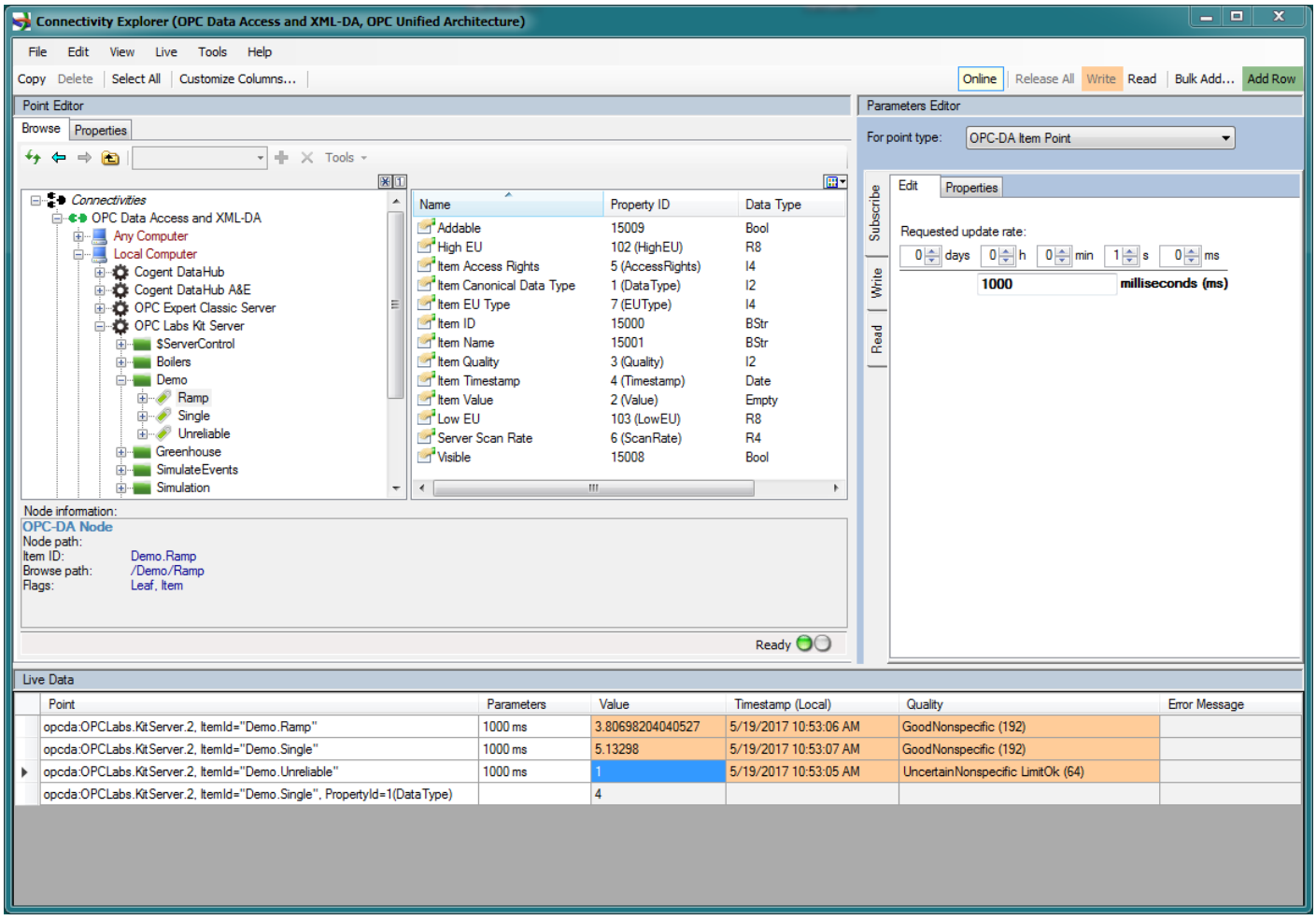
- **Point Editor (Section 6.2.3)**
- **Parameters Editor Window (Section 11.2.1.2)**
- **Live Point Data Window (Section 11.2.1.3)**

In This Topic

The usual flow of work within the Connectivity Explorer is as follows:

1. You select a point in the **Point Editor**.
2. If needed, you modify the associated parameters in the Parameters Editor.
3. Using the Add Row command (or simply by double-clicking a point in the Point Editor), you create a corresponding row in the **Live Point Data** window.
4. You observe the incoming data in the **Live Point Data** window, and possibly transfer them (using clipboard or drag-and-drop) to other applications, suich as Excel.
5. Optionally, you can also perform one-time Reads or Writes.

Below is a picture of main window, with some data already present in the **Live Point Data** subwindow.



The following articles describe the individual subwindows.

> For OPC UA, the Connectivity Explorer automatically generates its application certificate first time when it is started. This may fail if it is not running with elevated privileges ("Run as administrator"), and you may receive error message similar to "The requested key container was not found" (or other). To resolve this, run the Connectivity Explorer once with elevated privileges.

Workspace

The subwindows described above are arranged in a workspace that consists of individual cells where the subwindows can be placed. You can change the layout of the workspace in several ways:

- The splitters between the cells can be used to resize the cells.
- You can drag the cell page onto another cell and drop it there, or dock it against a border of a cell or the whole workspace. Start the drag-and-drop on the caption area of the cell page.
- Right-click on the cell page gives a context menu with commands such as Close, Close All But This, or Maximize.

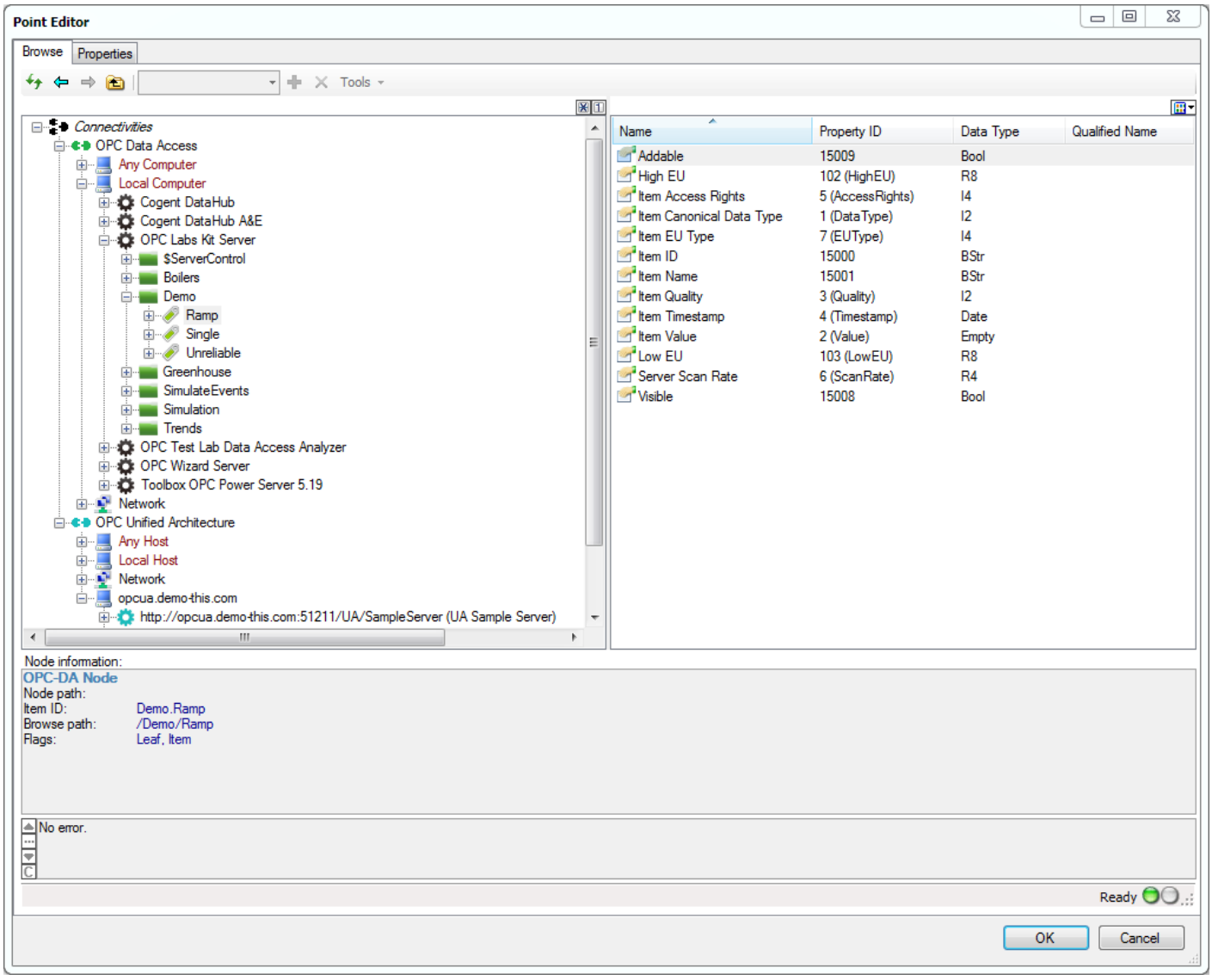
Closed subwindows can be reopened using corresponding commands on the View menu (see **Application Commands (Section 11.2.3)**).

11.2.1.1 Point Editor

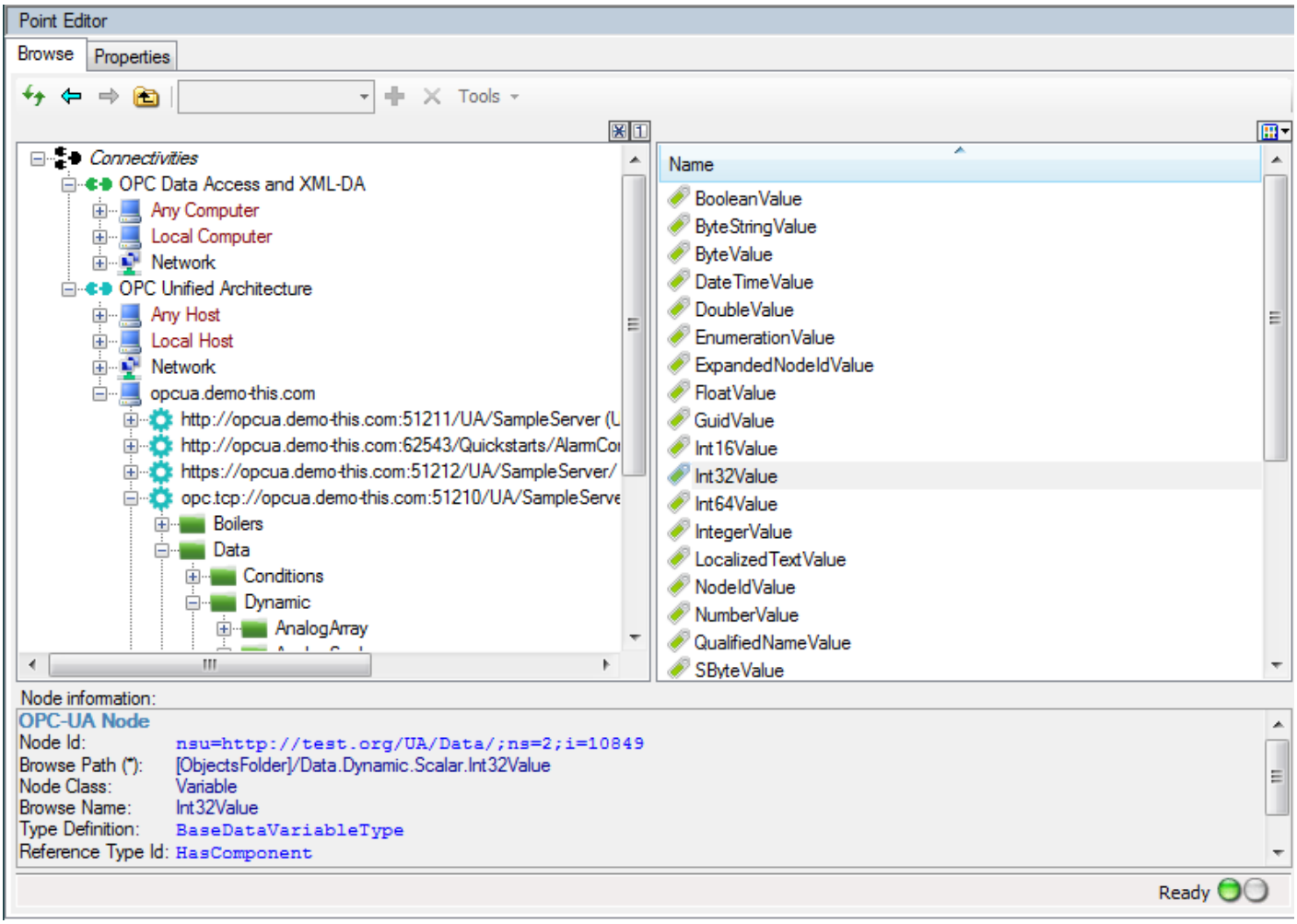
When you are working with the Connectivity Model (such as with the Live Binding in Visual Studio designer, or the Connectivity Explorer tool), you often need to specify the point you'd like to bind to or display.

For this purpose, you will use the **Point Editor**. The Point Editor can be a modal dialog (e.g. in Live Binding), or a subwindow in a larger set (as e.g. in Connectivity Explorer tool), but its main parts are always the same; what differs is the "framing".

In Live Binding, a **Point Editor** dialog is displayed. An example of it is on the following picture:



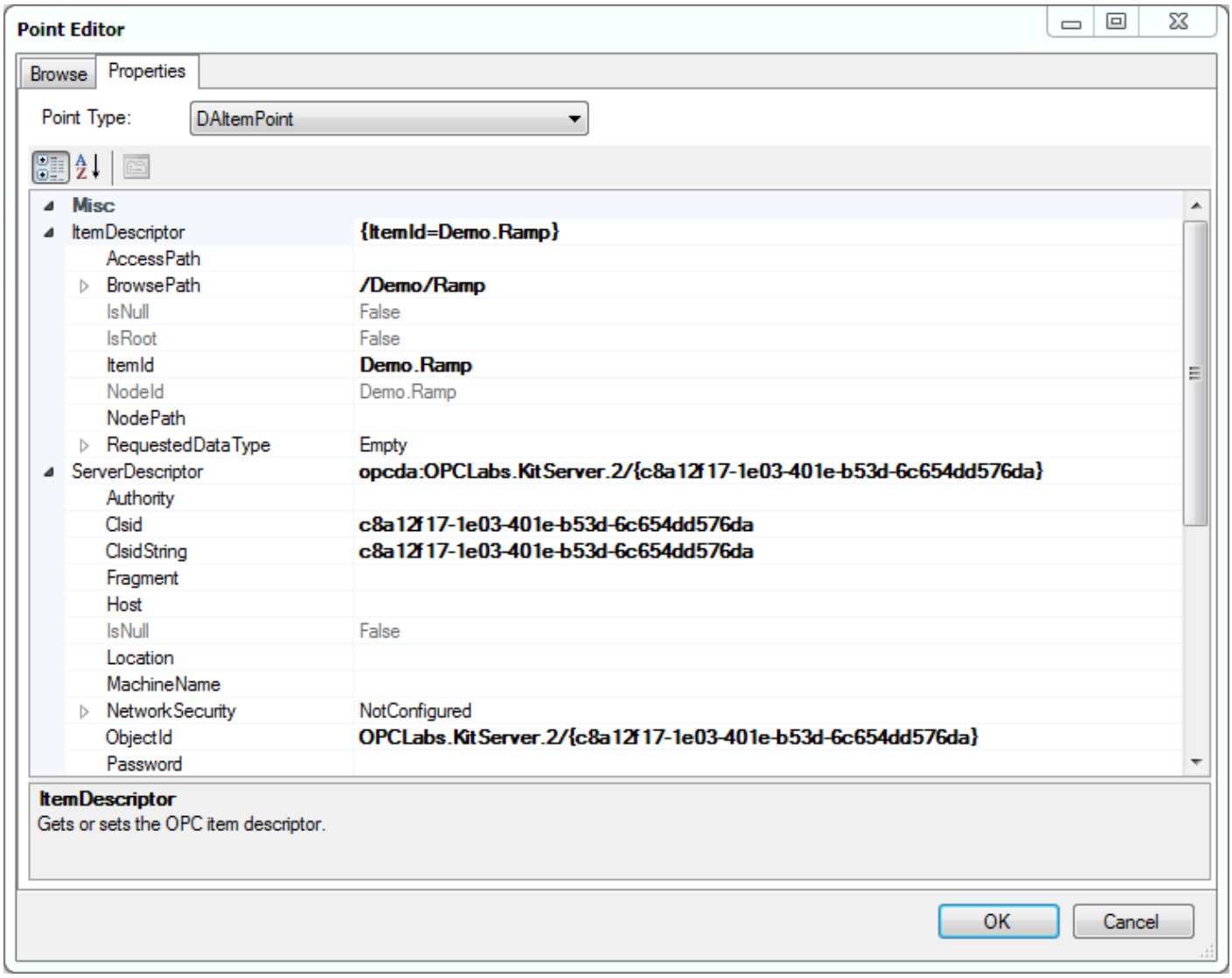
In Connectivity Explorer tool, the main window contains a **Point Editor** subwindow. An example of it is on the following picture:



The actual contents of the dialog or window will of course differ, depending on your configuration. This particular example is with a composite connectivity configured with both OPC Data Access and OPC Unified Architecture, showing multiple binding possibilities in the same dialog.

The point editor has two tabs, labeled **Browse** and **Properties**. Initially, the **Browse** tab is selected, and you are given an opportunity to browse the network, the OPC servers present, and the nodes in them. You can simply select the desired node and press OK (in Live Binding) or double-click on it (in Connectivity Explorer tool).

Sometimes the browsing is not possible, or you may need to view and possibly the details of the selected point. In such cases, you can switch to the **Properties** tab of the dialog:



In the **Properties** tab, the data that describe the point are displayed in a detailed property grid. In addition, in the upper part of the tab, you can select any type of point supported by the connectivity in use, and then fill in its properties as needed.

11.2.1.2 Parameters Editor Window

The **Parameters Editor** window allow you to specify **Parameters (Section 6.2.4)** that will be used with the (subscription) operation on the selected point. The parameters apply to the point being added to the **Live Point Data** window e.g. by the Add Row command or the Bulk Add menu command.

In This Topic

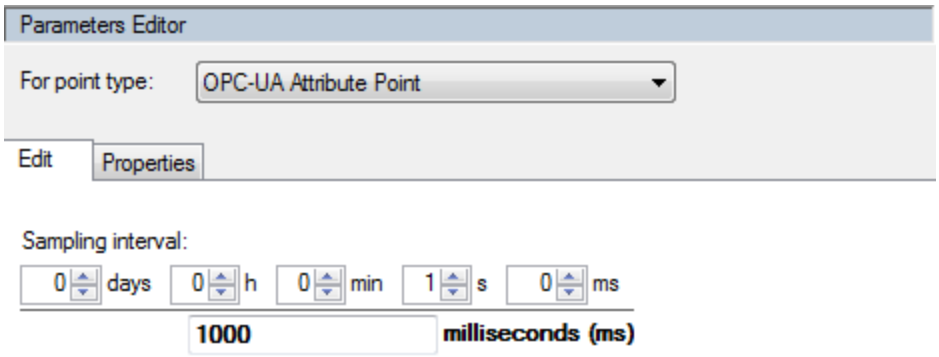
For point type

Different types of points require different parameters to be specified. This drop-down selector determines to which point type the parameters below apply. When you select a point in the **Point Editor** window, this drop-down is automatically

set for you correctly.

Edit Tab

In this tab, you can comfortably edit all or the most important parameters.

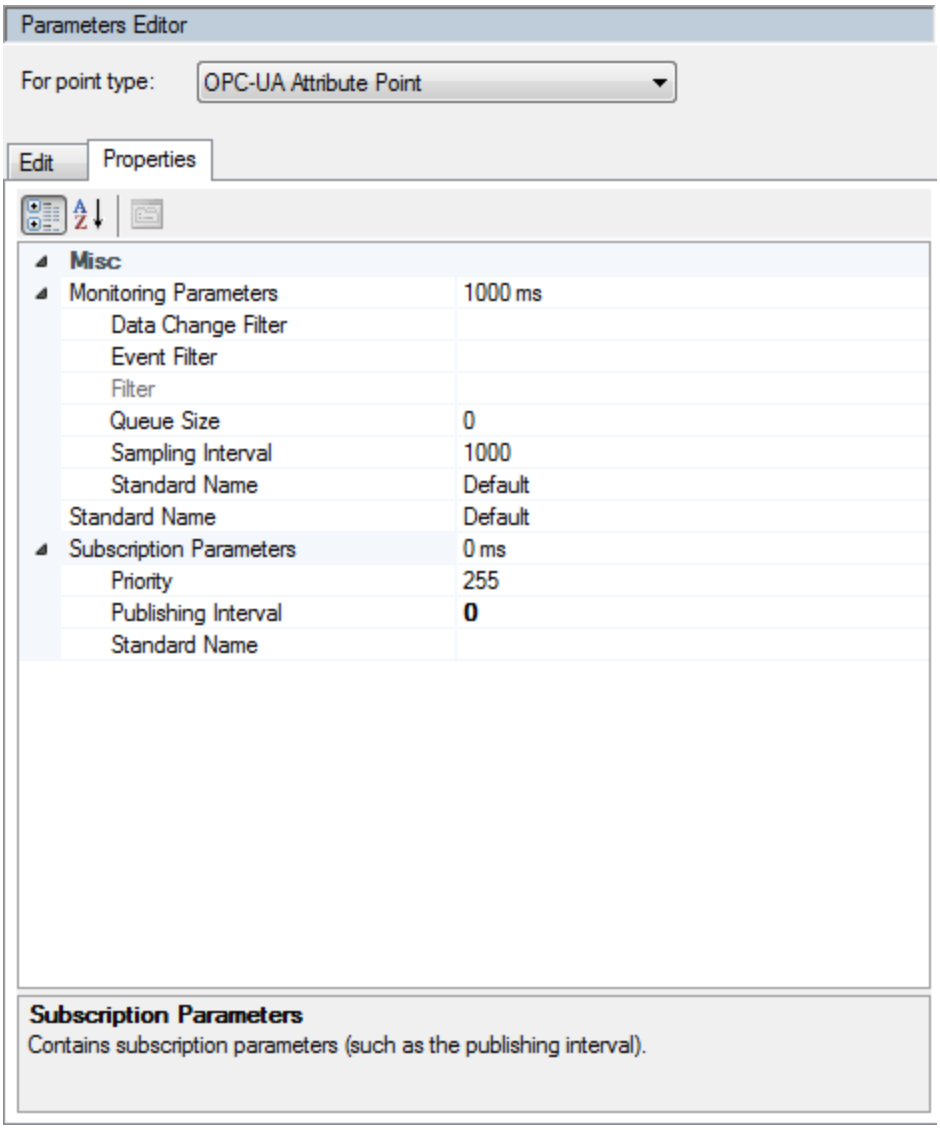


For some point types, not all parameters that exist for a point are accessible through the Edit tab. In such case, and if you need to view or change the remaining parameters, you need to switch to the **Properties** tab.

The **Edit** tab is not visible if there are no parameters to be configured for the operation on the point, or the specific point type does not support this type of parameter editing.

Properties Tab

This tab displays all configurable parameters for the operation on the point in a property grid.

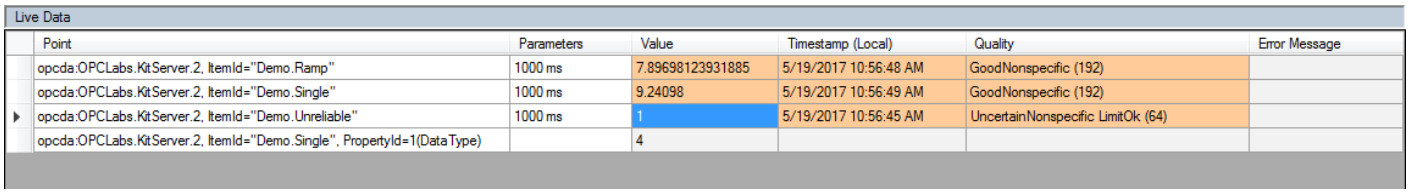


A description to the selected property is displayed in the information box at the bottom of the property grid.

11.2.1.3 Live Point Data Window

The **Live Point Data** window contains a grid, where each row represents data for a particular combination of point and parameters, and the columns represent various aspects of the data, including the value itself. You can add, remove, and modify the information contained in the grid. In addition, you can also transfer the information from the grid to other places and programs, including Excel.

In This Topic


Live Data

	Point	Parameters	Value	Timestamp (Local)	Quality	Error Message
	opcda:OPCLabs.KitServer.2, ItemId="Demo.Ramp"	1000 ms	7.89698123931885	5/19/2017 10:56:48 AM	GoodNonspecific (192)	
	opcda:OPCLabs.KitServer.2, ItemId="Demo.Single"	1000 ms	9.24098	5/19/2017 10:56:49 AM	GoodNonspecific (192)	
▶	opcda:OPCLabs.KitServer.2, ItemId="Demo.Unreliable"	1000 ms	1	5/19/2017 10:56:45 AM	UncertainNonspecific LimitOk (64)	
	opcda:OPCLabs.KitServer.2, ItemId="Demo.Single", PropertyId=1(DataType)		4			

Selecting Information

Some **Live Point Data** window commands work with a selected set of cells. The **Live Point Data** window supports a wide range of selection methods and modes, using both the mouse and the keyboard. Here are some tips:

- A single cell can be selected by clicking on it, or by navigating to it using the arrow keys on the keyboard.
- You can also use the ***Home***, ***End***, ***PageUp*** and ***PageDown*** keys for navigation.
- A whole row can be selected by clicking an empty area of the leftmost column.
- All cells can be selected by clicking an empty cell in the upper left corner of the grid.
- You can easily extend a selection to a rectangular range with a help of the ***Shift*** key.
- A non-continuous set of cells can be selected with a help of the ***Ctrl*** key.

There are also commands on the Edit menu that provide selection features.

Modifying Information

Writeable cells are visually distinguihed (by a different background color). You can navigate to the cells, and enter a new information into it. By doing so, the whole row stops displaying the information from Subscribe, and changes the font to ***bold italic***. After you have modified the cells you want, you can use the Write command (***Ctrl+W***) to perform a Write operation on all rows that in Write state.

Use the Release (***Esc***) or Release All command to revert the current row, or all rows, back to normal state. This is done either because you have changed your mind and do not want to write the row(s), or after you have viewed the results of the last Write operation and want to switch back to receiving updates from Subscribe.

Viewing Information

You can resize columns by dragging in the area of the vertical divider lines between them.

Commands

For information about the available commands, see **Live Point Data Commands (Section 11.2.1.3.1)**. The commands can be invoked e.g.

- From the main application menu.
- From the main application toolbar.
- From the **Live Point Data** context menu, accessible by right mouse click in the **Live Point Data** window area, or by pressing the ***Menu*** key when the window is focused.
- By keyboard shortcuts.

11.2.1.3.1 Live Point Data Commands

In This Topic

Edit Menu

Copy Command

Copies the contents of the selected cells to clipboard. See **Options Command (Section 11.2.3.1)** for settings that influence how the copy is done. For example, dynamic cells can be copied in such a way that they continue showing dynamic information from the data source after being pasted to Excel.

You can also use ***Ctrl+C*** to invoke this command.

Delete Command

Deletes the selected rows from the grid. You can also use ***Del*** to invoke this command.

Select All Command

Selects all cells in the grid. You can also use ***Ctrl+A*** to invoke this command.

Select None Command

Empties the selection set in the grid. You can also use ***Ctrl+N*** to invoke this command.

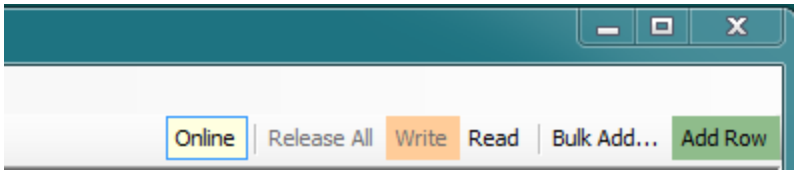
View Menu

Customize Columns Command

The **Customize Columns** command allows you influence which columns are displayed in the grid, which information they contain and how they look like. For details, see **Customize Columns Command (Section 11.2.1.3.1.1)**.

Live Menu

Commands under this menu allow you to control what gets displayed in the **Live Point Data** window. For easy access, they are also grouped as toolbar buttons near the upper right corner of the main window.



Add Row Command

Adds a new row to the **Live Point Data** window, for point selected in the Point Editor Window and with parameters selected in the **Parameters Editor Window (Section 11.2.1.2)**. The command also automatically determines the columns needed to display the relevant aspects of the data, and adds missing columns to the **Live Point Data** grid as

necessary.

The command is disabled when no point is selected in the **Point Editor** window.

Bulk Add Command

This command allows you to add many points at once. It invokes a point selection dialog with multi-select ability. Make your selection, and accept it in the dialog. All selected points will be added to the **Live Point Data** window. Each point will use parameters from the **Parameters Editor** window that correspond to its type.

Read Command

This command performs a Read operation on all currently selected rows (or rows that contain any selected cells, including the current cell). The Read is perfomed with parameters that are currently set for it in the **Parameters Editor** window.

Each row that was read stops displaying the information from Subscribe, and changes the font to **bold**. Use the Release (***Esc***) or Release All command to revert the current row, or all rows, back to normal state. This is done after you have viewed the results of the last Read operation and want to switch back to receiving updates from Subscribe.

You can also use ***Ctrl+R*** to invoke this command.

Write Command

This command performs a Write operation on all rows that are currently in a Write state (as indicated by ***bold italic*** font). A row enters the Write state when you start typing into any writable cell of it.

You can also use ***Ctrl+W*** to invoke this command.

The values you enter are, by default, parsed (evalued) before they are actually written. To configure this behavior differently, use the **Options Command (Section 11.2.3.1)**.

Release Command

Reverts the currently selected rows (or rows that contain any selected cells, including the current cell) back to the normal state, in which they display Subscribe updates.

Release All Command

Reverts all rows back to the normal state, in which they display Subscribe updates. You can also use ***Ctrl+E*** to invoke this command.

Online Command

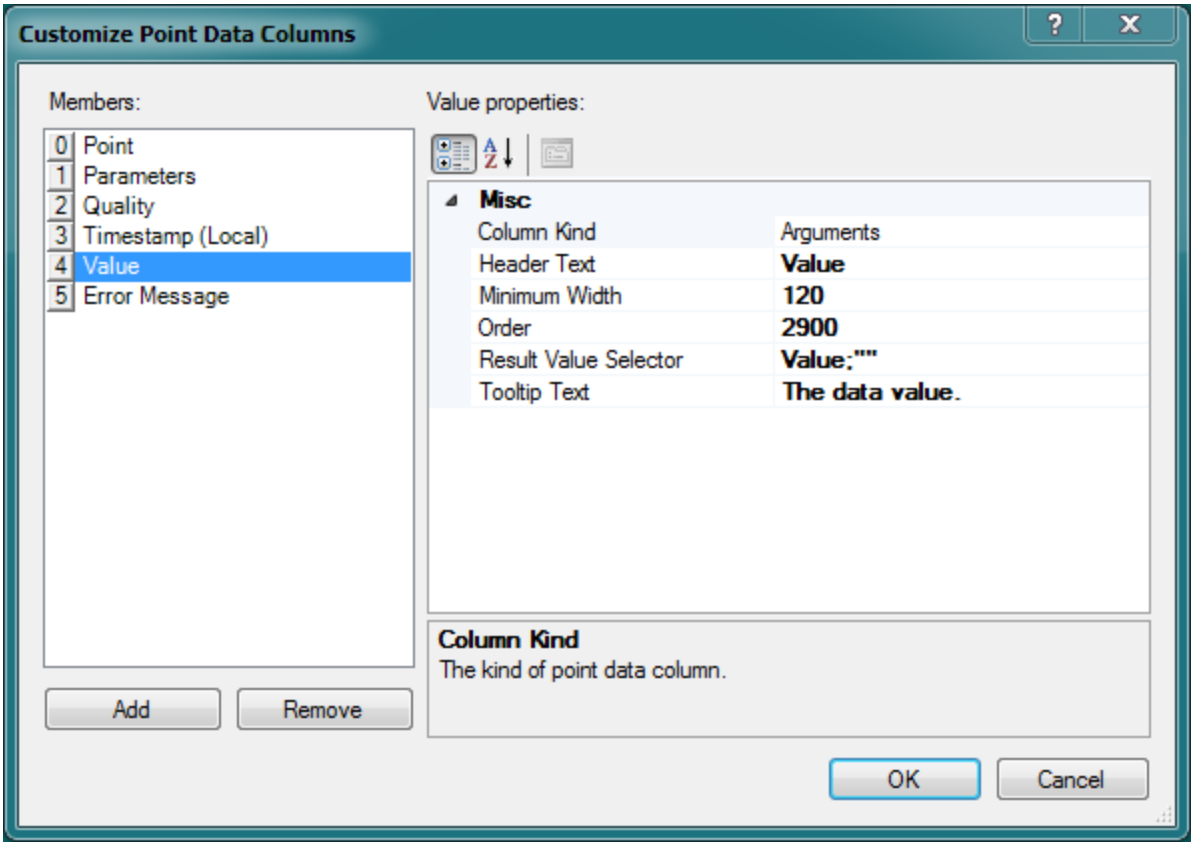
This is a toggle command. In the "on" state (online), the **Live Point Data** window is connected to the data source, and shows live data. Subscriptions are performed in order to obtain the data from the data source. In the "off" state (offline). the **Live Point Data** window is not connected to the data source, and the latest obtained values are shown.

You can also use ***Ctrl+L*** to invoke this command.

> ⚠ The **Online** command does not influence the **Point Editor** window. It is therefore possible to make connections and browse the data in the **Point Editor** even when the status is "offline".

11.2.1.3.1.1 Customize Columns Command

The **Customize Columns** command allows you influence which columns are displayed in the grid, which information they contain and how they look like. The command invokes a **Customize Point Data Columns** dialog:



The list on the left contains the columns that are currently displayed in the grid. By selecting a column from this list, you can view and edit its properties in the grid on the right side of the dialog. You can also add and remove columns.

Most properties are self-explanatory, and there is also a description in the box below the property grid. Besides many properties that influence *how* the information will be shown, there are two important ones that dictate *what* will be shown: Column Kind and Result Value Selector.

Column Kind Property

Selects where the information in the column comes from. It can be:

- **Point**: Information about the point in the data source that is being addressed.
- **Parameters**: Information about the parameters associated with the operation on the point.
- **Arguments**: Specific arguments of the operation - typically, its outputs.

Result Value Selector Property

Determines the specific piece (or pieces) of information that will be extracted and displayed. For details, see **Result Value Selector (Section 11.2.2)**.

11.2.2 Result Value Selector

In This Topic

The **Result Value Selector** is a text string that determines the specific piece (or pieces) of information that will be extracted and displayed from a target object.

The result value selector can have one of two forms:

```
successAndFailureElementPath
```

or

```
successElementPath ; failureElementPath
```

where

- *successAndFailureElementPath* is an element path in the target object that will be used regardless of whether the operation has succeeded or failed.
- *successElementPath* is an element path in the target object that will be used when the operation has succeeded.
- *failureElementPath* is an element path in the target object that will be used when the operation has failed.

In the second case, the elements paths are separated by a semicolon.

> When used for defining columns, the *failureElementPath* is only effective when the **Column Kind** is set to **Arguments** (and not to other kinds such as **Point** or **Parameters**).

Each element path can be either

- Empty. In this case, the target object as such will be the result of the extraction.
- A string constant, enclosed in doublequotes. The string will become the extracted value.
- A name of a property of the target object, optionally followed by names of sub-properties and so forth; the names are separated by dots. A value of the selected property (or its subproperty) will be extracted.

The types of target objects (point, parameters, and arguments), and therefore the names and meaning of their properties, can be found in **Point Types Reference (Section 16.5)**.

Examples

Always extract the **ErrorMessage** property from the target object:

```
ErrorMessage
```

Always extract the **Value** property from the target object:

```
Value
```

Extract the **Value** property from the target object when the operation succeeded; return an empty string when the operation has failed:

```
Value;""
```

Extract the **Value** property from the target object when the operation succeeded; return "N/A" when the operation has failed:

```
Value;"N/A"
```

Extract the **Value** property from the target object when the operation succeeded; extract the **ErrorMessage** property when the operation has failed:

```
Value;ErrorMessage
```

Extract the **Vtq** property (OPC-DA) from the target object when the operation succeeded; extract the **ErrorMessage** property when the operation has failed:

```
Vtq;ErrorMessage
```

Extract the **AttributeData** property (OPC-UA) from the target object when the operation succeeded; extract the **ErrorMessage** property when the operation has failed:

```
AttributeData;ErrorMessage
```

11.2.3 Application Commands

This article describes Connectivity Explorer commands that are not related to any specific window.

In This Topic

File Menu Commands

New Command

Clears (removes) any existing rows in the **Live Point Data** window.

You will be prompted to save any modified unsaved data, if needed.

Open Command

Allows you to retrieve a previously saved data (row definitions in the **Live Point Data** window) from an XML file. You can also use ***Ctrl+O*** to use this command.

You will be prompted to save any modified unsaved data, if needed.

Insert From Command

Allows you to add previously saved data (row definitions) to the current data in the **Live Point Data** window.

Save Command

Saves the current data (row definitions in the **Live Point Data** window) into an XML file. It uses the name of the file from the last Save As or Open; it asks for the file name in case none is known yet.

You can also use ***Ctrl+S*** to invoke this command.

Save As Command

Asks you for a name of the file, and saves the current data (row definitions in the **Live Point Data** window) into an XML

file. The selected file will then be used in subsequent Save commands automatically.

Exit Command

Closes the main window and exits the application. The selected options and connectivity configuration are preserved between subsequent runs of the application. You can also use ***Alt+F4*** to invoke this command.

You will be prompted to save any modified unsaved data, if needed.

View Menu Commands

Point Editor Command

Opens and selects the **Point Editor (Section 6.2.3)** window, or simply selects it if it is already open.

Parameters Editor Command

Opens and selects the **Parameters Editor Window (Section 11.2.1.2)**, or simply selects it if it is already open.

Live Point Data Command

Opens and selects the **Live Point Data Window (Section 11.2.1.3)**, or simply selects it if it is already open.

Event Records Command

Opens a window that allows you to view the event records generated by the application.

Currently, only operations of the "OPC Unified Architecture" connectivity generate event records.

Tools Menu Commands

Options Command

The **Options** command allows you to specify options for the application, i.e. configure the environment to your needs. For details, see **Options Command (Section 11.2.3.1)**.

Configure Connectivity Command

The **Configure Connectivity** command allows you to view and modify various setting that influence how the application connects to its data sources. For details, see **Configure Connectivity Command (Section 11.2.3.2)**.

Windows Menu Commands

Reset Windows Layout Command

Restores the application windows back to their original structure.

Help Menu Commands

Contents Command

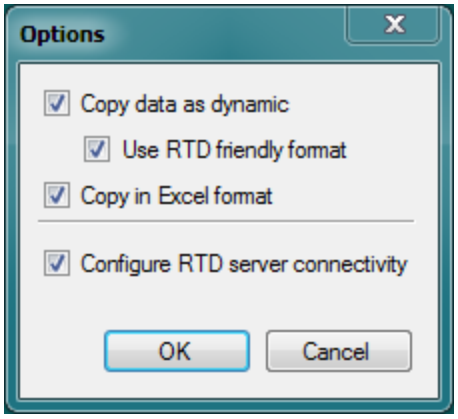
Opens a Web browser window (using your default browser) with online documentation to the application. You can also use ***F1*** to invoke this command.

About Command

Display a dialog box with information about the application, its purpose, version, and intellectual property-related matters.

11.2.3.1 Options Command

The **Options** command allows you to specify options for the application, i.e. configure the environment to your needs. The selected options are preserved between subsequent runs of the application.



Parse entered values

Determines whether the entered values will be parsed before use.

When this setting is turned off: The values are written as entered.

When this setting is turned on: If the entered value does not start with "{", it is written "as is", as a scalar. If the entered value starts with "{", it is considered an array, it must end with "}" and contain a list of elements separated by ",". For example, to write an integer array, enter ***{1,2,3}***. To write a string array, enter ***{abc,def,ghi}***. No provision is made for escaping the special cases yet.

Copy data as dynamic

When the **Copy data as dynamic** box in the **Options** dialog in **Connectivity Explorer** is checked, dynamic data transferred from the Connectivity Explorer through clipboard or drag-and-drop are transformed into RTD function calls, which allows them to remain dynamic when used in Excel.

When this box not checked, dynamic data transferred from the Connectivity Explorer simply contains their current value, static.

Use RTD friendly format

When the **Use RTD friendly format** box in the **Options** dialog in **Connectivity Explorer** is checked, and when the actual point and parameters used allow it, the generated RTD function calls use a friendly topic format, which is easier to read

and modify by a human. The friendly format uses a short series of arguments as topic strings in the RTD function.

When this box is not checked (or when the actual point and parameters used cannot be expressed in the friendly format), the generated RTD function call will specify the topic strings using XML (more lengthy, and less readable).

Copy in Excel format

When the **Copy in Excel format** box in the **Options** dialog in **Connectivity Explorer** is checked, (static) data transferred from the Connectivity Explorer through clipboard or drag-and-drop are formatted with Excel in mind. This setting tries to bypass some Excel "intelligence" that causes incorrect data imports.

Some examples of what this settings is trying to do:

- Prevent Excel from interpreting strings that begin with plus or minus sign, a decimal point, or a digit, as numbers.
- Prevent Excel from interpreting strings that "look like" dates or times as being actual dates or times.
- Assure proper transfer of Boolean values.
- Assure that date and time values will be transferred without misinterpretation, independent of any language or regional settings.
- Assure that time values do not lose milliseconds or even seconds.
- Assure that date and time values will be formatted by Excel as such.

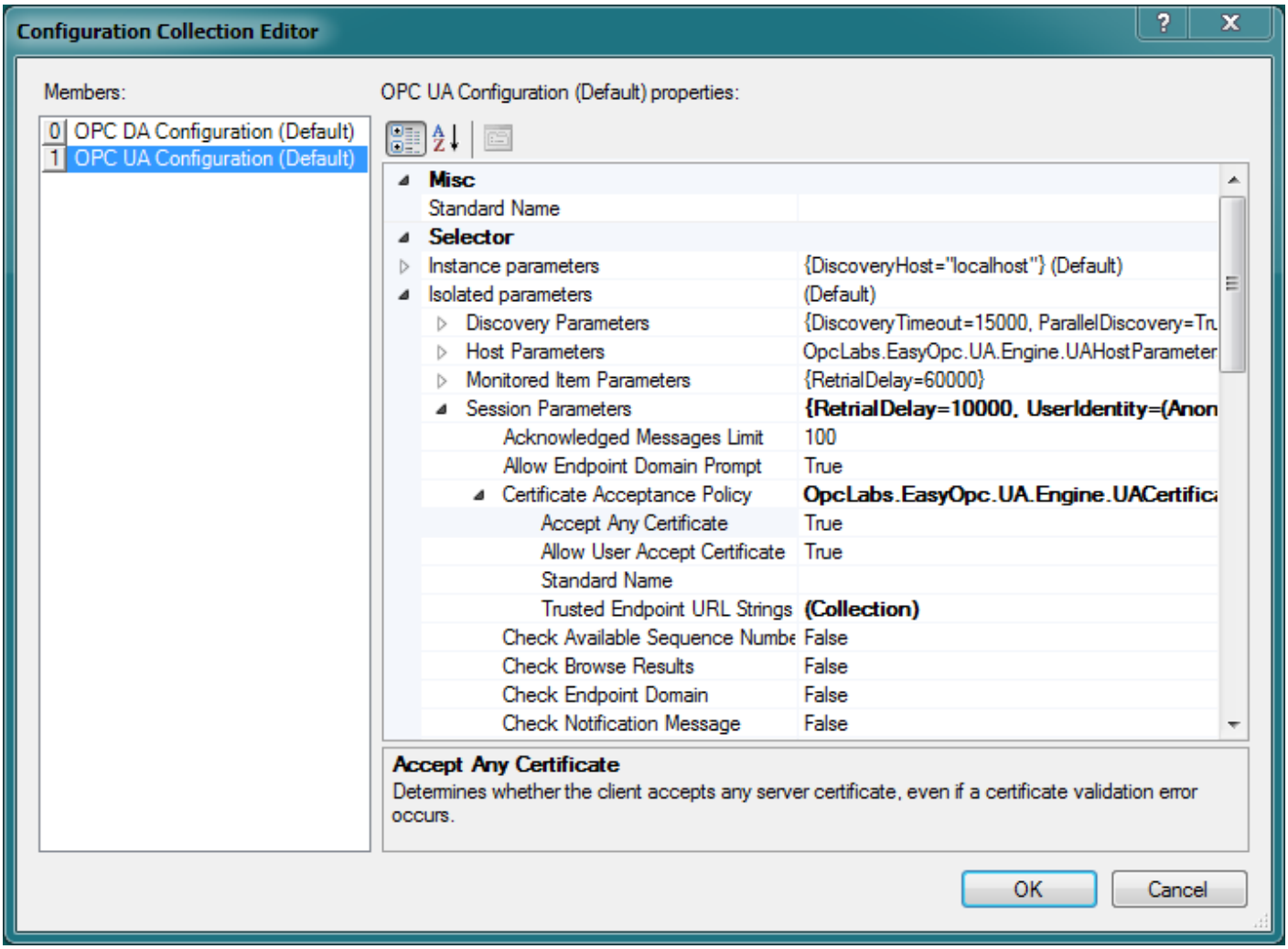
Uncheck this box if you need to transfer data to programs other than Excel.

Configure RTD server connectivity

When the **Configure RTD server connectivity** box in the **Options** dialog in **Connectivity Explorer** checked, any changes you make to the connectivity cofiguration (see **Configure Connectivity Command (Section 11.2.3.2)**) are also saved for use by the Connectivity RTD Server. This assures that your Excel spreadsheets with the RTD functions will use the same connectivity configuration as the one in the Connectivity Explorer. Note that you may have to restart Excel in order for any setting changes to take effect.

11.2.3.2 Configure Connectivity Command

The **Configure Connectivity** command allows you to views and modify various setting that influence how the application connects to its data sources. The connectivity configuration is preserved between subsequent runs of the application.



There is one group of configuration settings for OPC Data Access and XML-DA, and a separate group for OPC Unified Architecture. Each setting is briefly explained in the information box below the property grid. You will find further explanation to some settings elsewhere in this documentation, including the Reference part for the corresponding types.

In some cases (typically, after changing some value under Shared Parameters), you may be prompted to restart the application, so that the configuration changes can take effect.

If the **Configure RTD server connectivity** box is checked in the **Options** dialog (see **Options Command (Section 11.2.3.1)**), any connectivity configuration changes that you make will also be taken over by the RTD server, i.e. by the Excel worksheets that reference dynamic data using the RTD function.

The default settings are tuned for high interoperability.

For OPC UA, if you need settings that are tuned for best OPC compliance, select the OPC UA Configuration member, and then, in the property grid, click on the drop-down arrow next to the Standard Name, and select the OpcCompliance value from the list.

The default settings for OPC Unified Architecture allow (and prefer) non-secure connections. Therefore, a non-secure connection is made if the target OPC server has an endpoint that supports that.

You can change this behavior as follows:

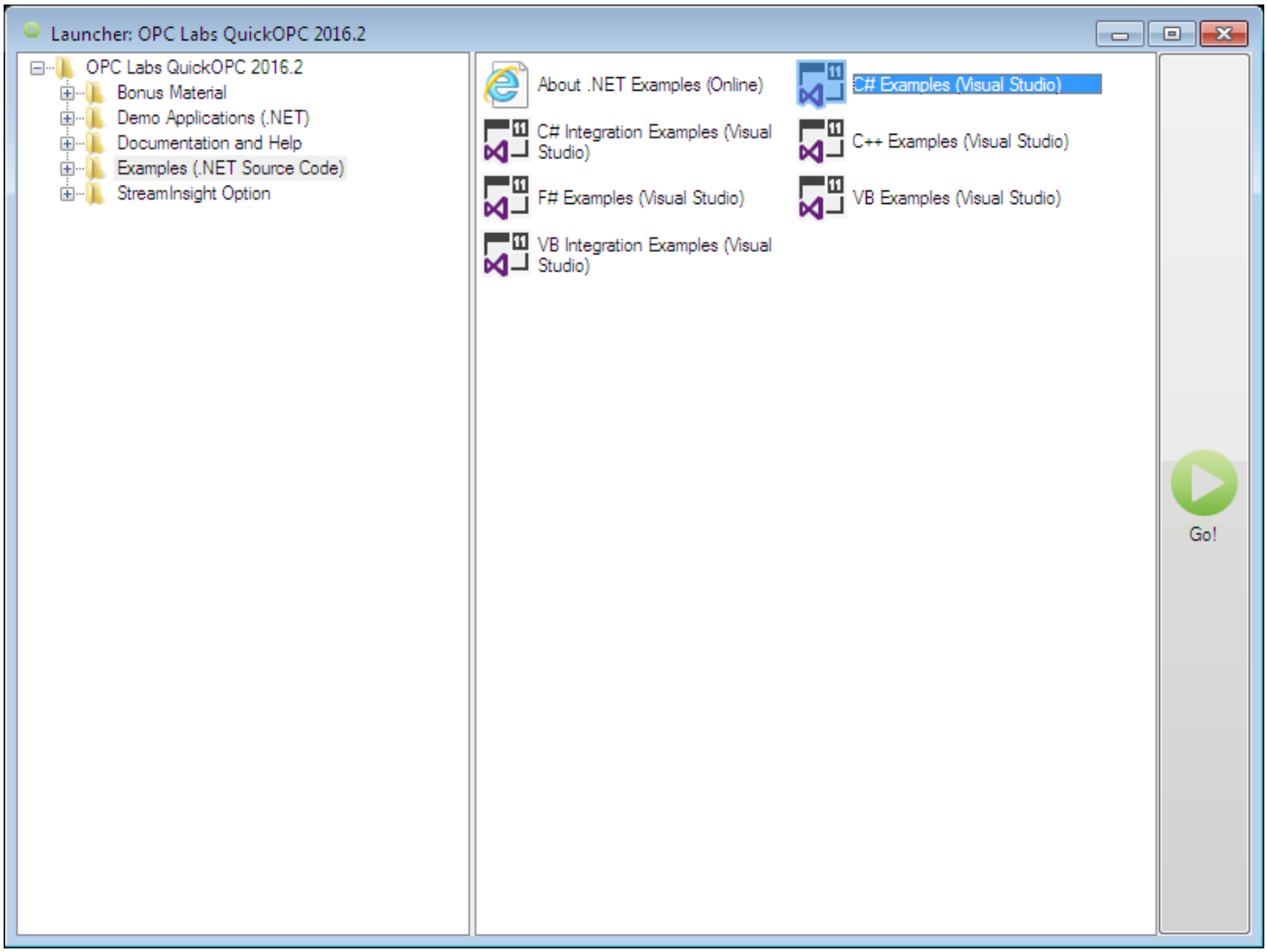
1. In the list on the left side, select the OPC UA Configuration member.
2. In the Selector category, expand the Isolated Parameters item, then expand the Session Parameters item, and then expand the Endpoint Selection Policy item.
3. Choose which security modes will be allowed: Click on the drop-down arrow next to the Allowed Message Security Modes, and select a combination of the modes. For example, for all modes except non-secure connections, make sure that Secure is checked, and SecurityNone is unchecked.
4. If both secure and non-secure modes are allowed (and available from the server), you may choose whether the secure or non-secure mode will be preferred: Click on the drop-down arrow next to the Message Security Preference, and select Negative to prefer the non-secure mode, None for no particular preference, or Positive to prefer the secure modes.

Note that if you select OpcCompliance for the Standard Name on Endpoint Selection Policy or higher in the hierarchy, the secure modes will be preferred, but non-secure mode will still be allowed.

11.3 The Launcher application

The Launcher application is a switchboard that presents the same shortcuts as those available from the Start menu. It serves as a starting point for work with QuickOPC. In addition, for users of Windows 8 and later, it allows them to easily navigate the various QuickOPC shortcuts, as the Start menu in Windows 8 and later is essentially broken for applications that install a hierarchical structure of shortcuts.

The Launcher is only available on Windows, when OPC Data Client has been installed with the **Setup Program (Section 3.1.1)**.



The Launcher application is also accessible from a desktop shortcut, from the Programs group (Start menu), and from the Quick Launch bar (note that Quick Launch isn't visible by default; to enable it, see https://support.microsoft.com/en-us/kb/975784).

You can use the tree on the left side to select a group of actions you are interested in, and then view the actions in the list on the right side. You can launch any action by double-clicking on its icon, or by selecting it first and then pressing the "Go!" button.

11.4 OpcCmd Utility

The **OpcCmd** utility is a program that allows performing various OPC operations from the command line. It can act as a generic OPC UA PubSub subscriber, and be used for evaluation, experiments, and testing.

For more information, see:

- Using OpcCmd Utility as OPC UA PubSub Subscriber
- Category:OpcCmd

11.5 Demo Servers and Publishers

To demonstrate capabilities of OPC Data Client, some OPC server (or publisher, for PubSub) is needed. One such server ships with the product, and more are made available remotely (online).

For OPC Classic (COM/DCOM-based):

- **Simulation Server for OPC Classic (Section 11.5.1)**

For OPC XML:

- **OPC XML Sample Server (Section 11.5.2)**

For OPC UA (Unified Architecture):

- **OPC UA Sample Server (Section 11.5.3)**
- **QuickStart Alarm Condition Server (OPC UA) (Section 11.5.4)**

For OPC UA PubSub:

- **OPC UA Demo Publisher (Section 11.5.5)**

11.5.1 Simulation Server for OPC Classic

The demo application installed with the product, and most examples use an OPC Simulation Server that is installed together with OPC Data Client. The server's ProgID is **"OPCLabs.KitServer.2"**.

The demo application and the examples are designed to connect to the Simulation OPC Server in its default configuration (i.e. as shipped). In fact, some very simple examples connect to just one OPC item, named **"Demo.Ramp"**. There are various other OPC items in this server that you can use in your own experiments too.

Note 1: When OPC proxies/stubs are not installed on the system, the installation program installs them together with the simulation server.

Note 2: On 64-bit platforms, the installation program still registers the 32-bit (x86) binary of Simulation OPC Server. This configuration gives better OPC compatibility. The 64-bit binary of Simulation OPC Server is also installed to the disk, and can be registered manually if needed.

This functionality is not available under (or the text does not apply to) .NET Standard development platform.

11.5.2 OPC XML Sample Server

The OPC XML Sample Server is not installed with the product, but is made available on the Internet. You can use it for your explorations and tests. The server's URL is http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx . Our example code also refers to this server, so that the examples can be run without having to set up a server locally.

The server is based on the sample server code from OPC Foundation. It provides some OPC data of its own, and it wraps two OPC COM-based servers (OPC-DA 2.0 and OPC-DA 3.0). It contains a combination of items of various data types, static and dynamic, some readable and some also writeable.

OPC XML does not define a way for server discovery (browsing for servers). For this reason, if you are using user interface components such as the browsing dialogs or controls, you need to specify the server manually, using its URL.

> You are only licensed to use this online service with connections made from OPC Data Client, and not from any other software. Also, please do not use it for any kind of load or volume testing or tests that require substantial computing or network resources (such as subscribing to many variables at once), because that could have negative impact on other users, and on our ability to provide this online service. We reserve the right to deny access to online services when we detect that these conditions has not been followed.

11.5.3 OPC UA Sample Server

To demonstrate capabilities of OPC Data Client, some OPC server is needed. Most examples use an OPC UA Sample Server provide by OPC Labs (based on OPC Foundation sample server) that is available over Internet. The server can be accessed on opc.tcp://opcua.demo-this.com:51210/UA/SampleServer, http://opcua.demo-this.com:51211/UA/SampleServer or https://opcua.demo-this.com:51212/UA/SampleServer (these are so-called Discovery URIs).

Note that if there is a firewall between the client machine and the server, it needs to have the corresponding ports (in the above example, 51210, 51211 and 51212) open for outbound traffic.

The demo application and the examples are designed to connect to the OPC UA Sample Server in its default configuration. In fact, some very simple examples connect to just one OPC node. For example, a dynamically changing double-precision value has node Id **"nsu=http://test.org/UA/Data/;i=10854"**. You can either call the browsing methods to obtain other node Ids together with their descriptions, or you can simply borrow the node Ids used in our examples. There are various other OPC nodes in this server that you can use in your own experiments too.

> You are only licensed to use this online service with connections made from OPC Data Client, and not from any other software. Also, please do not use it for any kind of load or volume testing or tests that require substantial computing or network resources (such as subscribing to many variables at once), because that could have negative impact on other users, and on our ability to provide this online service. We reserve the right to deny access to online services when we detect that these conditions has not been followed.

11.5.4 QuickStart Alarm Condition Server (OPC UA)

We run a public Quickstart Alarm Condition Server (OPC UA) that you can use for your explorations and tests. The server is located on **opcua.demo-this.com**, and its endpoint URLs are:

- **opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer**
- http://opcua.demo-this.com:62543/Quickstarts/AlarmConditionServer

Our example code also refers to this server, so that the examples can be run without having to set up a server locally.

The server is based on the code from OPC Foundation.

> You are only licensed to use this online service with connections made from OPC Data Client, and not from any other software. Also, please do not use it for any kind of load or volume testing or tests that require substantial computing or network resources (such as subscribing to many variables at once), because that could have negative impact on other users, and on our ability to provide this online service. We reserve the right to deny access to online services when we detect that these conditions has not been followed.

11.5.5 OPC UA Demo Publisher

The **OPC UA Demo Publisher** is useful for demonstrating OPC UA PubSub. It is the publisher that we use in most of our examples for OPC UA PubSub.

For more information, see:

- UADemoPublisher Basics
- Category:UADemoPublisher

11.6 Test Tools

OPC Data Client ships with certain internal testing tools that might be of interest to advanced users, and were created partially in relation to OPC compliance certification process.

The test tools are not installed by default; you need to select “Custom Install”, and then specifically enable them on the “Select Components” page in the installation wizards. After installation, the test tools can be reached through the Start menu or the Launcher application, in the “Test Tools” group.

There is a separate document (online) that contains the list of test tools, their purpose, and instructions on how to operate the tools. When the Test Tools are installed, you can access it from the Start menu as well.

12 Product Options

12.1 Excel Option

Microsoft, MSDN, Excel, Windows, and Windows Server are trademarks of the Microsoft group of companies.
All other trademarks are property of their respective owners.

12.1.1 Introduction to Excel Option

With the Excel Option for OPC Data Client, you can set up a communication link between Excel and any OPC server. It is possible to subscribe to and view real-time data, and also write the data back.

OPC Classic, OPC XML and OPC UA servers are supported. No programming, macros or add-ins are necessary.

This functionality is not yet available for OPC UA PubSub.

In This Topic

Supported Excel Versions

The OPC Data Client Excel Option supports following Excel versions:

- Excel 2016 (32- and 64-bit)
- Excel 2013 (32- and 64-bit)
- Excel 2010 (32- and 64-bit)
- Excel 2007 (32- and 64-bit)

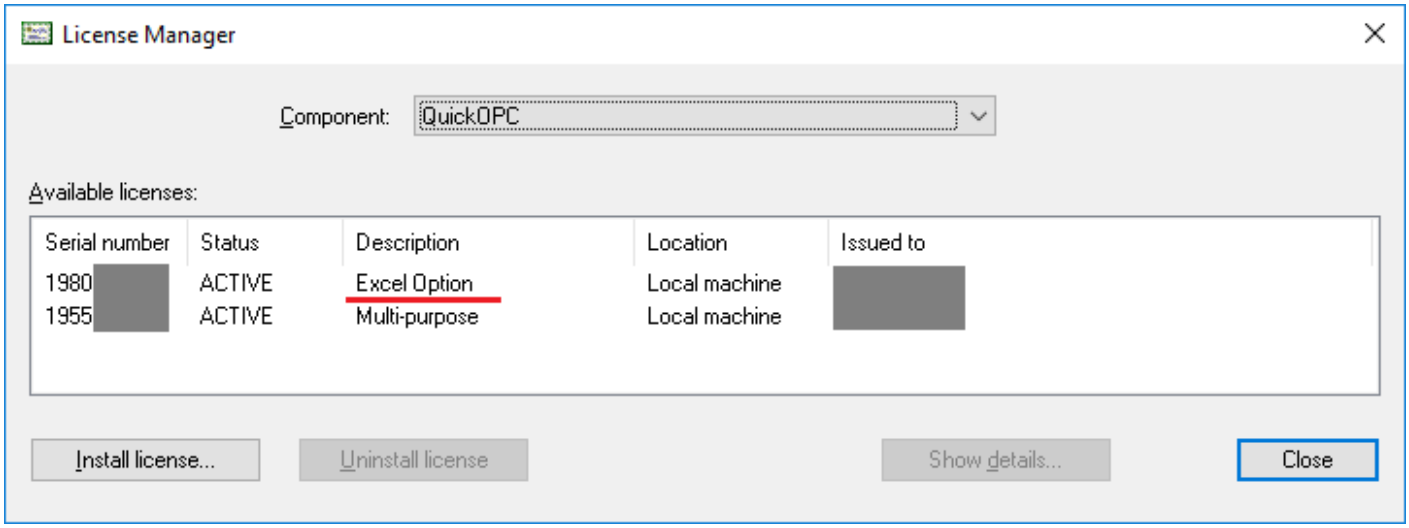
Excel 2003 may also work, but we have not tested with it and do not officially support it.

Note that Excel 2003, Excel 2007 and Excel 2010 are no longer on mainstream support by Microsoft.

Licensing

Excel Option is usually licensed separately from the base product, and in such case you will receive a separate license file for the Excel Option. See **License Manager (Section 11.1)** for more information.

The license is shown with Description "Excel Option". Its ID for use with the **LMConsole Utility (License Manager Console) (Section 11.1.1)** is "ExcelOption". When properly installed, you will have two licenses under the "QuickOPC" component name in License Manager, and the form will look similar to this:



If you are running the trial license, Excel Option is automatically included (enabled), with 30-minute runtime limitation per

each process (Excel) run.

12.1.2 Getting Started under Excel

OPC Data Client allows you to develop Excel spreadsheets that contain values and formulas that are dynamically updated from OPC data, without any programming.

In This Topic

Making a first real-time Excel spreadsheet

In this Getting Started procedure, we will create am Excel spreadsheet that subscribes to OPC Data Access (OPC DA) item or OPC Unified Architecture (OPC UA) node, and continuously displays it changes. With OPC Data Client., this can be achieved without any coding, and also without installing any Excel add-ins, only with help of so-called RTD server component that comes with OPC Data Client.

1. Install OPC Data Client. Make sure that you have selected an installation choice that includes Excel integration.
2. Start Microsoft Excel and create a new blank workbook.
3. From Windows Start menu, navigate to OPC Data Client 2020.2 Launcher, and run it.
4. In the **Launcher** application (when the root node is selected in the tree on the left side, which is the default state), select the Connectivity Explorer icon in the list on the right side, and press the green Go! button. Alternatively, simply double-click on the Connectivity Explorer icon.
5. In the **Connectivity Explorer** application, locate the **Point Editor** sub-window, and select a "point" to subscribe to, using one of the procedures described below.
   a. For OPC Data Access (OPC DA), on the **Browse** tab, navigate in the tree (on the left) to Connectivities -> OPC Data Access and XML-DA -> Local Computer -> OPC Labs Kit Server -> Demo -> Ramp.
   b. For OPC Unified Architecture (OPC UA), on the **Browse** tab, navigate in the tree (on the left) to Connectivites -> OPC Unified Architecture -> opcua.demo-this.com -> http:// opcua.demo-this.com:51211/UA/Sample Server (UA Sample Server) -> Data -> Dynamic -> Scalar -> DoubleValue.
6. Press the dark green Add Row button on the main toolbar (near the upper right corner of the **Connectivity Explorer** application). Alternatively, use the Live -> Add Row command form the main menu.

   In the **Live Point Data** sub-window (at the bottom), a new row will be displayed in the grid. The row contains information about the point selected, and dynamically changing value and timestamp(s), and other related information (e.g. diagnostics).
7. In the **Live Point Data** sub-window, click on the changing number under the Value column header. This should leave only this single cell selected (highlighted).
8. Using the mouse, drag the selected grid cell to the worksheet area of your Excel workbook, and drop it onto a cell of you choice. This will fill the cell with a special formula. The OPC value shows in the cell, and it updates itself dynamically.

   As an alternative to drag-and-drop, you can copy the grid cell from the **Live Point Data** sub-window using the Edit -> Copy command form the **Connectivity Explorer** menu, using the Copy button on the main toolbar, or with

the ***Ctrl+C*** keyboard shortcut. Then, paste the dynamic data to the Excel worksheet.

Where do I go from here?

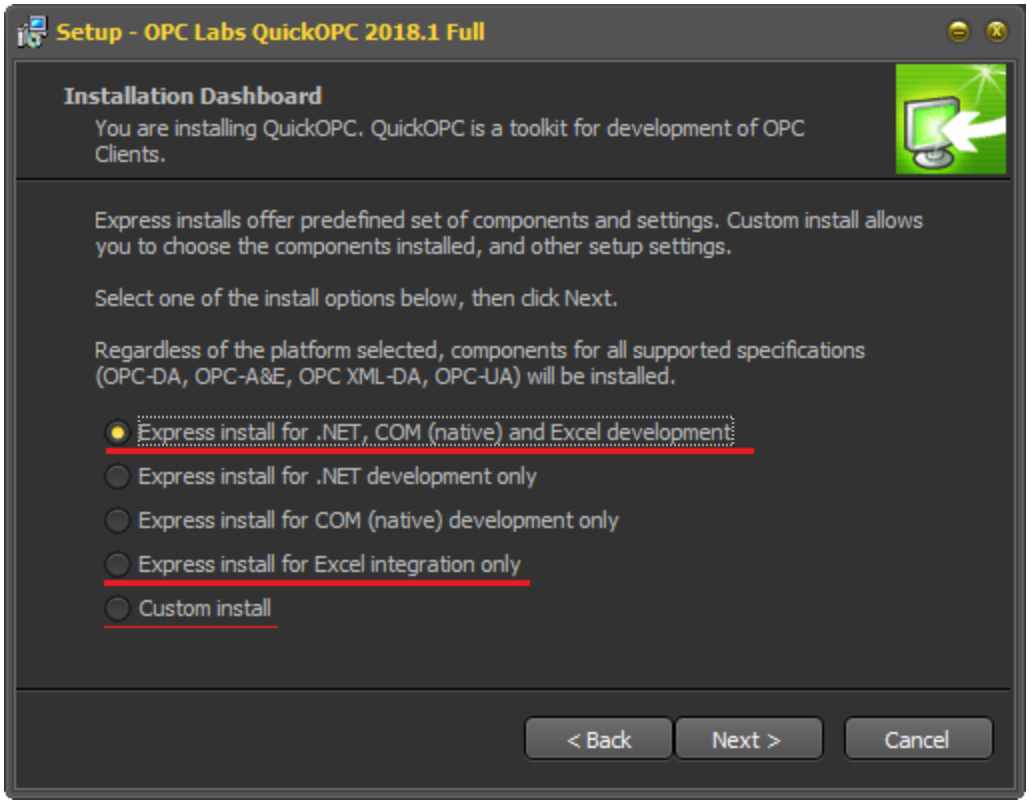
You now have a basic idea of how to use the OPC Data Client. There are many additional resources that you can use to gain additional knowledge about the product. Here are some of them:

- Play with the Demo application (accessible from the Start menu) and explore the various functionalities that the product provides.
- Read the User's Guide.
- Study the Reference documentation.
- Review the additional examples and tools included with the product.

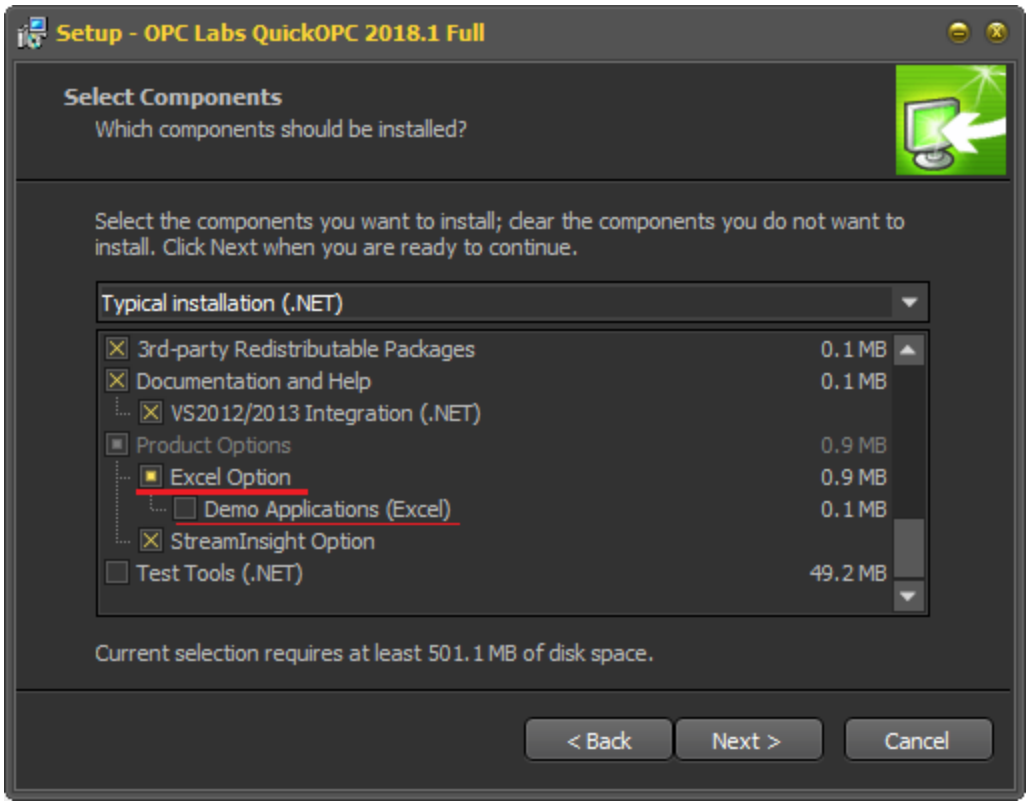
12.1.3 Excel Option - Installation

Excel Option is part of normal OPC Data Client standalone installer (full or production), but it needs to be properly enabled during installation. If you want to work with Excel Option, make sure you have selected the proper choice when installing the product. If you are not sure, run the installation again, making the choices as described below.

In order to have the Excel Option installed with OPC Data Client, select one of the red underlined options on the Installation Dahsboard page of the setup wizard:



If you opt to make a Custom Install, the Excel Option is then listed on the Select Components wizard page, and needs to be enabled there:



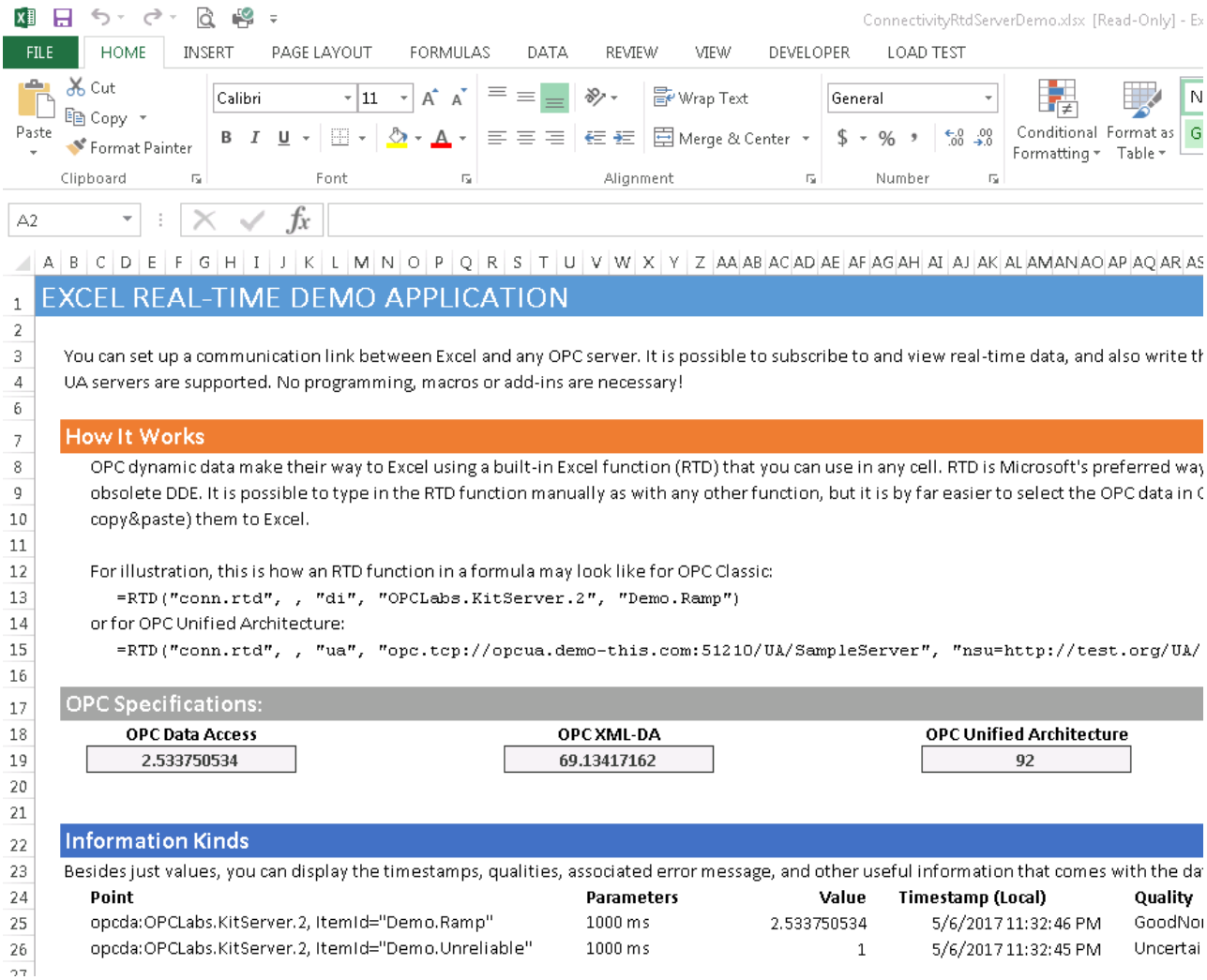
Excel Option is not contained in the OPC Data Client NuGet packages.

12.1.4 Excel Demo Applications

OPC Data Client installs with a demo workbook that shows the basic Excel integration features of the product (Excel Real Time Data). The demo workbook is available from the Start menu or the Launcher application. It shows:

- Principles of operation, and examples of RTD functions.
- OPC specifications that are supported, with live data coming in via each of the specifications.
- The kinds of information that can be accessed.
- Bidirectional communication (OPC writes).
- Examples of various data types supported.

For illustration, a part of the workbook is shown the following figure.



12.1.5 Excel Option Fundamentals

OPC dynamic data make their way to Excel using a built-in Excel function (RTD) that you can use in any cell. RTD is Microsoft's preferred way to get real-time data into Excel, instead of the obsolete DDE. It is possible to type in the RTD function manually as with any other function, but it is by far easier to select the OPC data in **Connectivity Explorer (Section 11.2)** tool, and drag-and-drop (or copy&paste) them to Excel. The RTD function addresses a so-called **Connectivity RTD Server (Section 12.1.5.1)**, a COM component that is part of the Excel Option, and this RTD server then performs the necessary communication with the data source.

In design-time, you would therefore typically use the **Connectivity Explorer (Section 11.2)** to locate the dynamic data and transfer them to the Excel worksheets you are creating, or you would type in the formulas containing the RTD functions directly into Excel.

In run-time, only the **Connectivity RTD Server (Section 12.1.5.1)** has to be installed on the computer, for your Excel worksheets to be functional.

12.1.5.1 Connectivity RTD Server Fundamentals

This article describes how the syntax of the Excel RTD function to be used with Excel Option. When used in the way described here, the RTD function retrieves real-time data using the Connectivity RTD Server.

In This Topic

RTD Function Syntax

Use the following syntax in Excel (the '=' designates a formula):

```
=RTD(RealTimeServerProgID, ServerName, Topic1, [Topic2], ...)
```

where the function arguments are as follows:

- *RealTimeServerProgID*
  A string that represents the Program ID of the Connectivity RTD server. *RealTimeServerProgID* is a required argument.
- *ServerName*
  A string that represents the name of the server on which the RTD server is to be run. If the RTD server is run locally, the *ServerName* should be an empty string ("") or omitted.
- *Topic1*, [*Topic2*], ...
  Strings that determine the data being retrieved. You can use from one to 28 topics, although only one topic is required.

RealTimeServerProgID Argument

For Connectivity RTD Server, use one of the following ProgIDs as the *RealTimeServerProgID*:

```
"opclabs.office.excel.connectivityrtdserver"
```

or

```
"conn.rtd"
```

Both these ProgIDs are equivalent. The second one is intentionally short, intended for manual typing into Excel.

> ProgIDs are case insensitive.

ServerName Argument

The Connectivity RTD Server is usually run locally (i.e. on the computer where Excel runs), and you should therefore omit the *ServerName* argument.

Topic Arguments

The topic arguments determine the data to be retrieved. See **Topic Syntax (Section 12.1.5.1.1)** for details.

Examples

For illustration, this is how an RTD function in a formula may look like for OPC Classic:

```
=RTD("conn.rtd", , "di", "OPCLabs.KitServer.2", "Demo.Ramp")
```

or for OPC Unified Architecture:

```
=RTD("conn.rtd", , "ua", "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=11218")
```

Related Links

More information about the Excel RTD function:

- RTD function
- How to set up and use the RTD function in Excel - Microsoft Support

12.1.5.1.1 Topic Syntax

As explained in **Connectivity RTD Server Fundamentals (Section 12.1.5.1)**, the RTD function syntax contains a number of topic arguments - at least one, but there can be more. The synatx and semantics of the topic arguments is given by the RTD server being used. The OPC Data Client Excel Option uses its internal Connectivity RTD Server, and the syntax of topic arguments is described here.

In This Topic

Topic Formats

The Connectivity RTD Server supports two formats of topics sequences in the RTD function.

The so-called friendly topic syntax is designed to be easily understandable by humans, and can also be manually typied in without too much effort. This format cannot, however, fully describe all possible requirements, due to the number of properties that can theoretically be set on different points and operation parameters.

For this reason, a second format, XML-based (XML Toipc Syntax) is also supported by the Connectivity RTD Server. This format can describe all requirements without loss, but is much less readable, and not really suitable for manual creation.

> By default, the Connectivity Explorer application creates RTD functions with topics formatted in the friendly topic syntax, whenever possible. The Connectivity Explorer automatically switches to the XML topic syntax if it detects that the given data cannot be described without loss in the friendly topic format.

Friendly Topic Syntax

With the friendly topic syntax, the overall sequence of topic arguments for Connectivity RTD Server is as follows:

```
typeString, pointIdentifier1, pointIdentifier2, ..., subscribeParameter1,
subscribeParameter2, ... [,
resultValueSelector]
```

Where:

- *typeString* denotes the type of point this RTD function is subscribing to. The type is given by an alias; it can be e.g. "opcdaitem", "opcdaproperty", or "opcuaattribute". See **Point Types Reference (Section 16.5)** for details on available point types and their type aliases. Type aliases are case insensitive.

- *pointIdentifier1*, *pointIdentifier2*, ... are topics that describe the point to be subscribed to. Their number, syntax and semantics depends on the type of the point. See **Point Types Reference (Section 16.5)** for details on point identifiers for each point type.
- *subscribeParameter1*, *subscribeParameter2*, ... are topics that describe the parameters of the Subscribe operation on the point. Their number, syntax and semantics depends on the type of the point. See **Point Types Reference (Section 16.5)** for details on point identifiers for each point type.
- *resultValueSelector* is a text string that determines the specific piece (or pieces) of information that will be extracted and displayed from a target object. For details, see **Result Value Selector (Section 11.2.2)**.

Further topics may be appended to the end of this sequence if the **Topic Write with RTD Server (Section 12.1.5.1.2)** feature is used.

XML Topic Syntax

The XML topic syntax is capable to fully describe the point and parameters. It is currently not documented, and is subject to change.

Topic Spanning

Some limitations of the RTD function make it problematic to pass the data to it. For example, each topic (string) must not be longer than 255 characters. This is normally not an issue with the friendly topic syntax, but can easily occur with XML topic syntax, where the resulting XML strings tend to be quite long. The Connectivity RTD Server (and the associated integrated functionality in Connectivity Explorer) overcome this limitation by spanning the topics strings over several RTD function arguments. The first topic contains information about how the arguments are laid out in the topic strings. The precise syntax and semantics of this topic spanning feature is currently not documented, and is subject to change.

Examples

Friendly topic syntax example for an OPC-DA item:

```
   "opcdataitem", "OPCLabs.KitServer.2", "Demo.Ramp"
```

Friendly topic syntax for OPC Unified Architecture:

```
   "opcuaattribute", "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=11218"
```

12.1.5.1.2 Topic Write with RTD Server

You can write values from Excel back to an OPC server too. This is usually done by appending a source cell reference to the list of RTD function arguments.

The RTD function in this mode works like normally, but in addition, it writes the value from the referenced cell into the data source (OPC server). The write occurs every time whenever the RTD function is evaluated (at the beginning, when explicitly invoked, or whenever its parameters change).

In This Topic

> ⚠ Use the **Topic Write** feature carefully.
>
> This feature is basically a "trick", it uses the RTD function of Excel for something it was not designed for. As such, it

has its risks and limitations.

One issue comes from the fact that the RTD function is evaluated anew (and therefore, a Write occurs) not only when the value to be written changes, but also initially - for example, when the worksheet with the RTD function is opened. If you use the Topic Write feature of the RTD server, your overall design must be such that this kind of behavior does not cause a problem.

Friendly Topic Syntax

In order to use the Topic Write feature, add the following sequence of topics to the end of the normal sequence, as described in **Topic Syntax (Section 12.1.5.1.1)**.

```
valueToWrite [, writeParameter1, writeParameter2, ... [, inputElementPath]]
```

Note that if there are optional arguments at the end of the original topic sequence, before the Topic Write arguments (for a particular point type and parameters, or for the result value selector), all such parameters must be given first (i.e. cannot be left out). In Excel, you can simply use commas to indicate their presence; the actual values (empty strings) are not necessary, though.

XML Topic Syntax

The XML topic syntax is capable to fully describe the write parameters. It is currently not documented, and is subject to change.

Examples

The following formula in Excel continuously displays values of the boiler level setpoint item. At the same time, any initial and new value from the cell A2 is written to the setpoint item.

```
=RTD("conn.rtd", , "da", "OPCLabs.KitServer.2", "Boilers.Boiler #1.LC1001.SetPoint", ,
"Value;""""", A2)
```

12.1.5.2 RTD Server Data Conversions

The usual conversions performed by OPC Data Client on the data take place. For details, see:

Data Types in OPC Classic (Section 9.9.1)

- **Data Types in OPC Classic (Section 9.9.1)**
- **Data Types in OPC-UA (Section 9.9.2)**

In This Topic

In addition, the Connectivity RTD Server makes following conversions before returning the data through the RTD function:

- Null values are converted to empty strings.
- Generic objects are converted to their string representations.
- 64-bit and wider integers and decimals are converted to strings (this avoids loss of precision).

Dates and Times

The dates and times are represented as floating point number in Excel internally, and are transferred as such to Excel from the Connectivity RTD Server. When dates or times from the Connectivoty RTD Server are displayed by Excel, by default the general number format is used. You need to set the cell format manually as needed. For example, you can use one of the pre-defined Date or Time formats that Excel offers, or a custom format, like (e.g.):

```
m/d/yyyy h:mm:ss AM/PM
```

You may also want to read: Format numbers as dates or times.

12.1.5.3 Connectivity Explorer Integration with Excel

The **Connectivity Explorer (Section 11.2)** application is designed to serve as a primary source of dynamic data items for use in Excel. As such, it contains several features that provide tight integration with the Excel Option.

In This Topic

Excel Clipboard Formats

Copy data as dynamic

When the **Copy data as dynamic** box in the **Options** dialog in **Connectivity Explorer** is checked, dynamic data transferred from the Connectivity Explorer through clipboard or drag-and-drop are transformed into RTD function calls, which allows them to remain dynamic when used in Excel.

When this box not checked, dynamic data transferred from the Connectivity Explorer simply contains their current value, static.

Use RTD friendly format

When the **Use RTD friendly format** box in the **Options** dialog in **Connectivity Explorer** is checked, and when the actual point and parameters used allow it, the generated RTD function calls use a friendly topic format, which is easier to read and modify by a human. The friendly format uses a short series of arguments as topic strings in the RTD function.

When this box is not checked (or when the actual point and parameters used cannot be expressed in the friendly format), the generated RTD function call will specify the topic strings using XML (more lengthy, and less readable).

Copy in Excel format

When the **Copy in Excel format** box in the **Options** dialog in **Connectivity Explorer** is checked, (static) data transferred from the Connectivity Explorer through clipboard or drag-and-drop are formatted with Excel in mind. This setting tries to bypass some Excel "intelligence" that causes incorrect data imports.

Some examples of what this settings is trying to do:

- Prevent Excel from interpreting strings that begin with plus or minus sign, a decimal point, or a digit, as numbers.
- Prevent Excel from interpreting strings that "look like" dates or times as being actual dates or times.
- Assure proper transfer of Boolean values.
- Assure that date and time values will be transferred without misinterpretation, independent of any language or regional settings.
- Assure that time values do not lose milliseconds or even seconds.

- Assure that date and time values will be formatted by Excel as such.

Uncheck this box if you need to transfer data to programs other than Excel.

Synchronized Connectivity Configuration

Configure RTD server connectivity

When the **Configure RTD server connectivity** box in the **Options** dialog in **Connectivity Explorer** checked, any changes you make to the connectivity cofiguration (see **Configure Connectivity Command (Section 11.2.3.2)**) are also saved for use by the Connectivity RTD Server. This assures that your Excel spreadsheets with the RTD functions will use the same connectivity configuration as the one in the Connectivity Explorer. Note that you may have to restart Excel in order for any setting changes to take effect.

12.1.6 Excel Option Application Deployment

This chapter describes how to deploy Excel solutions developed with use of OPC Data Client Excel Option.

For some uses, it is acceptable to perform the deployment manually. In other cases, you will find yourself writing an installer for your application, and you will probably want to include the installation of OPC Data Client components in it as well. A Production installer (part of OPC Data Client) can help with that.

> This chapter is only concerned with deploying RTD the server part of the Excel Option. That is sufficient for run-time. The Connectivity Explorer tool is used in design-time only, and therefore does not need to be deployed with the solution.

12.1.6.1 Excel Option Installation Elements

The elements described in this chapter need to be installed with your solution based on OPC Data Client Excel Option.

You need to deploy:

1. The Excel files (workbooks) that contain your solution, along with any associated files.
2. The Connectivity RTD Server.

12.1.6.1.1 Connectivity RTD Server Installation Elements

For Excel worksheets with RTD functions referring to Connectivity RTD Server, you need install following files:

- App_Web_OpcLabs.EasyOpcClassicRaw.amd64.dll
- App_Web_OpcLabs.EasyOpcClassicRaw.x86.dll
- OpcLabs.BaseLib.dll
- OpcLabs.BaseLibForms.dll
- OpcLabs.BaseLibPresentation.dll
- OpcLabs.EasyOpcClassic.dll
- OpcLabs.EasyOpcForms.dll

- OpcLabs.EasyOpcUA.dll
- **OpcLabs.OfficeConnector.dll**

You will find them under the **Bin** subdirectory in the installation directory of the product.

In addition, you need to register the **OpcLabs.OfficeConnector.dll** assembly using the Assembly Registration tool (**Regasm.exe**). The Assembly Registration tool is a part of .NET Framework, and is therefore available on every computer that has the .NET Framework installed.

For more information, please refer to http://msdn.microsoft.com/en-us/library/tzat5yw6(v=vs.110).aspx. It is recommended that you use the **/codebaseoption** when registering the assembly.

> ⚠ On 64-bit systems, a 32-bit and a 64-bit version of **RegAsm.exe** are present. Using the 32-bit version of **RegAsm.exe** registers the assembly for use by 32-bit Excel, and using the 64-bit of **RegAsm.exe** registers the assembly for use by 64-bit Excel. Depending on your Excel bitness, you need to select the proper version of **RegAsm.exe**.

12.1.6.2 Excel Option Configuration Elements

The elements described in this chapter need to be configured in your solution based on OPC Data Client Excel Option.

12.1.6.2.1 Connectivity RTD Server Configuration Elements

If you needed to change the connectivity configuration in order to develop your solution, you will also need to make sure that the same connectivity configuration is in effect for your users.

The connectivity configuration data for the Connectivity RTD Server is stored in the following file (optional):

- ***<CommonApplicationData>*\Connectivity RTD Server\ConnectivityConfiguration2.xml**

The ***<CommonApplicationData>*** folder is the directory that serves as a common repository for application-specific data that is used by all users. Its actual location depends on your system; it can be e.g. **C:\ProgramData**.

Most likely, you will not create this file manually. You will rather take the connectivity configuration file from the computer where you have developed your solution, and then place it onto user computers.

12.1.7 Excel Option Troubleshooting

If you do not get the expected value in Excel, you may be left without an error message that would guide towards the cause of the problem.

In order to see the related status and error messages, you can add cells that refer to this information. To do so, from Connectivity Explorer, drag-and-drop or copy-and-paste (in addition to the formula you already have) also the cells under "**Quality**" (OPC Classic), "**Status**" (OPC UA) and "**Error Message**" columns.

You can also create formulas that combine either the value, or the error message in the same cell, depending on a condition. Refer to **Topic Syntax (Section 12.1.5.1.1)** and **Result Value Selector (Section 11.2.2)** for more details.

12.1.8 Additional Resources for Excel Option

Study the Reference documentation (also available from Start menu).

Explore the demo installed with the product.

Check the vendor's Web page for updates, news, related products and other information.

In This Topic

Excel RTD function

How to set up and use the RTD function in Excel - Microsoft Support

RTD function

RTD.ThrottleInterval Property (Excel)

RTD Object (Excel)

Security settings and Excel RealTimeData servers

12.2 StreamInsight Option

12.2.1 Introduction to StreamInsight Option

OPC Data Client is a toolkit for writing OPC client applications. Microsoft StreamInsight is a platform for writing and running complex event processing (CEP) applications.

The StreamInsight Option for OPC Data Client allows you to bring in data from OPC sources into StreamInsight, analyze them and process them further, and even feed the results back to OPC servers. Data from OPC can be combined with data from multiple other sources. You can monitor the data for meaningful patterns, trends and exceptions. Data can be analyzed and correlated while they are in-flight.

The StreamInsight Option for OPC Data Client supports connections to OPC servers with following specifications:

- OPC Data Access
- OPC Alarms and Events
- OPC Unified Architecture (data access)

> This functionality is not yet available for OPC UA PubSub.

The StreamInsight support in OPC Data Client is not separate from its other parts. Instead, it builds on top of many features and components already available in OPC Data Client. Specifically, the StreamInsight Option makes ideal usage of the connection-less approach in OPC Data Client, and the reactive programming model (OPC Reactive Extensions, Rx/OPC). This means that you can also easily combine the StreamInsight development with all other options and features provided by OPC Data Client.

This part of the documentation provides information that is specific to the StreamInsight Option for OPC Data Client. You should be able to follow the main ideas presented here without advance knowledge of OPC Data Client, but for serious development, you will need OPC Data Client knowledge as well. For information about OPC Data Client itself, please refer to corresponding OPC Data Client documentation parts, such as the Users' Guide, or the Reference documentation.

> For in-memory streaming analytics, check out also Microsoft Trill (https://github.com/microsoft/Trill). OPC Data Client can interoperate with Trill seamlessly. See **Installed Examples - Console - SimpleTrillApplication (Section 13.1.4.1.9)**.

> The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

12.2.2 StreamInsight Option Installation and Getting Started

The StreamInsight Option for OPC Data Client is installed together with the OPC Data Client product, as described further below (there is no separate installation package). Please refer to the Installation section of this document if you need details about the OPC Data Client installation procedure.

12.2.2.1 StreamInsight Option Licensing

The StreamInsight Option for OPC Data Client is a licensed product. You must obtain an appropriate license to use it in development or production environment. For evaluation purposes, you are granted a trial license, which is in effect if no

other license is available.

With the trial license, the components only provide valid OPC data for 30 minutes since the application was started. After this period elapses, performing OPC operations will return an error. Restarting the application gives you additional 30 minutes, and so on. If you need to evaluate the product but the default trial license is not sufficient for you purposes, please contact the vendor or producer, describe your needs, and a special trial license may be provided to you.

12.2.2.2 Installing a StreamInsight Instance

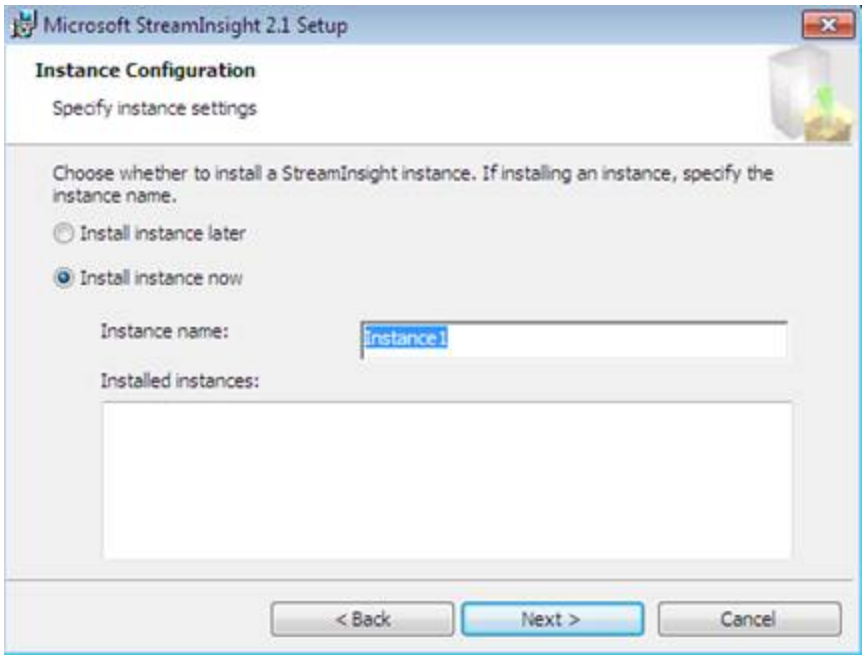
The Getting Started procedures assume an existence of a local StreamInsight instance named Instance1. You need to install Microsoft StreamInsight 2.3, and create this instance. General setup instructions to Microsoft StreamInsight can be found here: http://technet.microsoft.com/en-us/library/ee378749.aspx.

We have created our own summary of the steps needed, which you may find useful, and easier to follow:

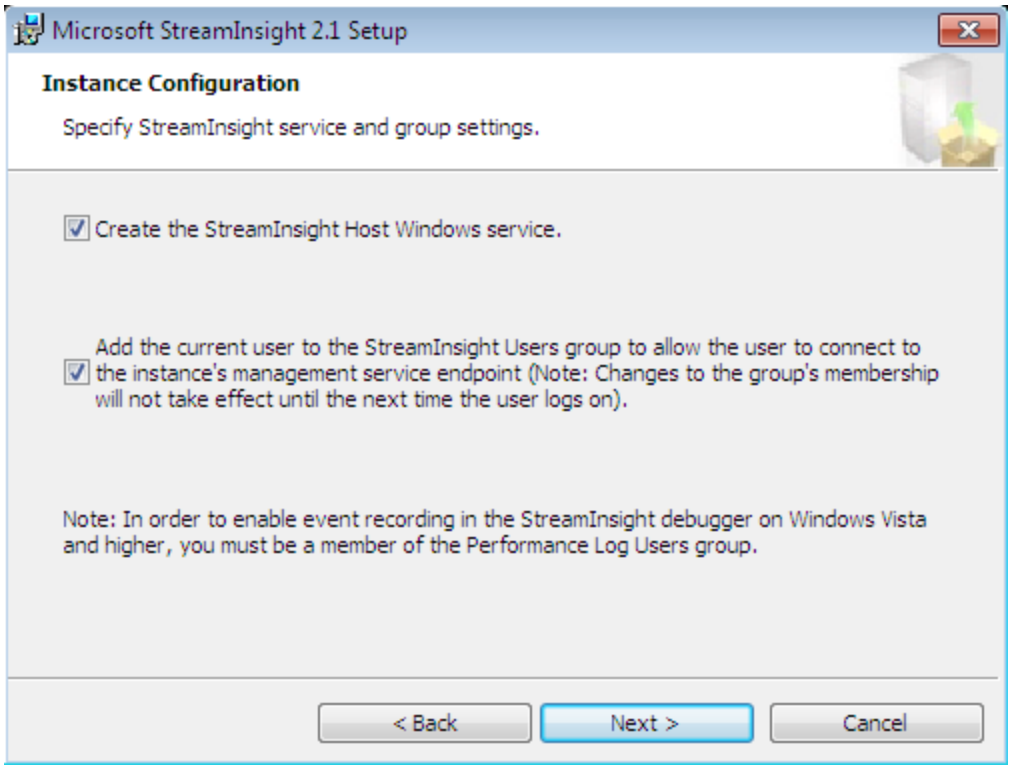
1. Download the x86 (for 32-bit operating systems) or x64 (for 64-bit operating systems) version of StreamInsight.msi (not StreamInsightClient.msi) from Microsoft (https://www.microsoft.com/en-us/download/details.aspx?id=42295, or search for "Microsoft SQL Server 2014 feature Pack"), and run it.

Note: Do not use StreamInsight installation from Microsoft SQL Server 2012 or earlier – it contains Microsoft StreamInsight 2.0 or older, which is not supported by StreamInsight Option for OPC Data Client. Also, StreamInsight 2.1 is not officially supported with this version of OPC Data Client.

2. Go through the installation wizard pages, accepting defaults or entering information as needed. On the "Instance Configuration" page, select "Install instance now" option, and enter "Instance1" (without the double-quotes) into the "Instance name" field. Creation of this instance is recommended now (although you can create it later), because it is the instance that we use in our examples.



3. Continue going through the installation wizard pages. We recommend that you check the "Add the current user to the StreamInsight Users group..." box. Finalize the installation wizard then.



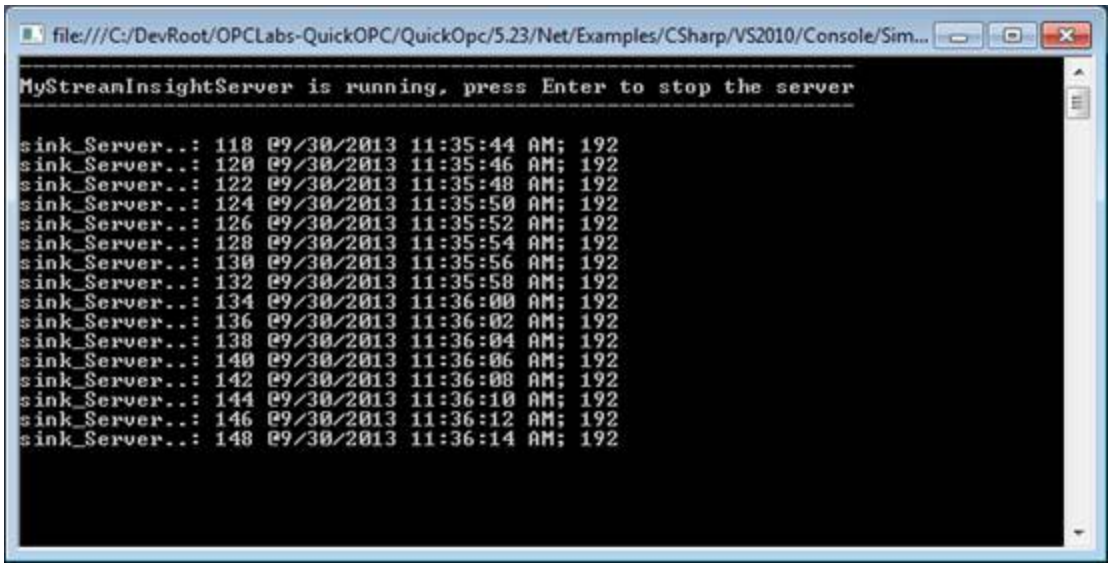
4. If you have checked the "Add the current user to the StreamInsight Users group..." box as instructed above, log out of Windows, and then log in again.

12.2.2.3 Building and Running a First StreamInsight Application with OPC

In this Getting Started procedure, we will build and run a console application in C# or Visual Basic that shows how to create an OPC Data Access event source, and query it for events carrying even data value.

We assume that you have already installed a StreamInsight instance named "Instance1", as described in the previous chapter. We also assume that you have Microsoft Visual Studio 2017 or later installed.

1. Install OPC Data Client. If you are reading this text, chances are that you have already done it. The "StreamInsight Option" installation component must be checked during the installation, but it is on by default, so unless you are doing a custom installation and choosing a non-default different installation type, you do not have to do anything special during the OPC Data Client installation.
2. From the Start menu (locate the OPC Data Client product submenu) or the Launcher application, and select "StreamInsight Option -> Examples Solution (Source Code)". This will open the solution in Visual Studio.
3. In the Solution Explorer of Visual Studio, double-click "Program.cs" under "SimpleDAStreamInsightApplication" project. This will give you a chance to review the one-page program we are going to build and run.
4. Right-click on "SimpleDAStreamInsightApplication" project in the Solution Explorer, and select "Debug -> Start new instance". This will build and run the program.
5. Observe the results in the console window, and after you receive enough events, press Enter to stop the server. The output should look similar to this:



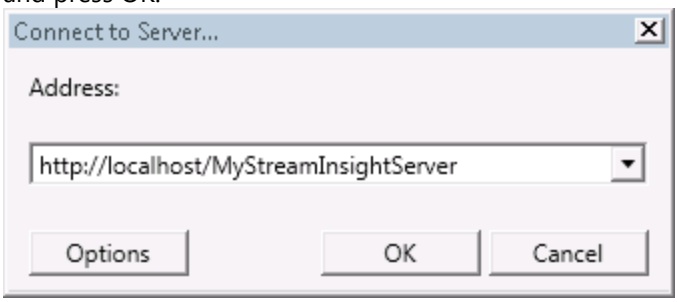
You can also try similar projects for other OPC Specifications:

- **SimpleAEStreamInsightApplication**: For OPC Alarms&Events (OPC-A&E), or
- **SimpleUAStreamInsightApplication**: For OPC Unified Architecture (OPC-UA).

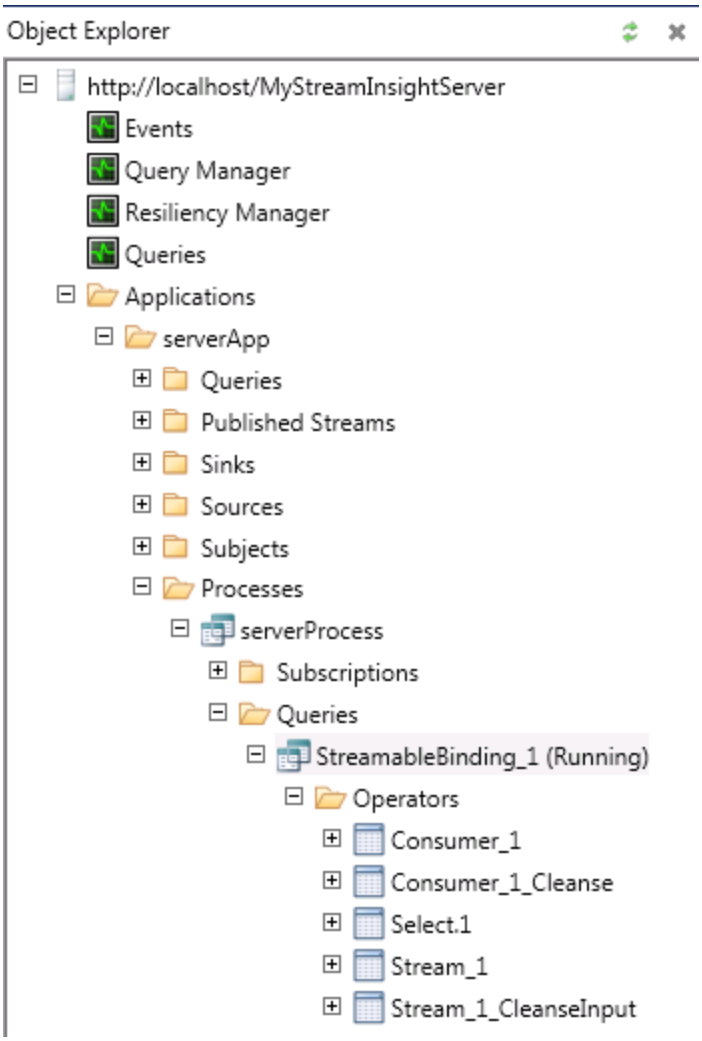
12.2.2.4 Debugging the StreamInsight Event Flow

In this Getting Started procedure, we will use the StreamInsight Event Flow Debugger to view how the events are internally processed in StreamInsight. We assume that you have performed the preceding Getting Started procedures already.

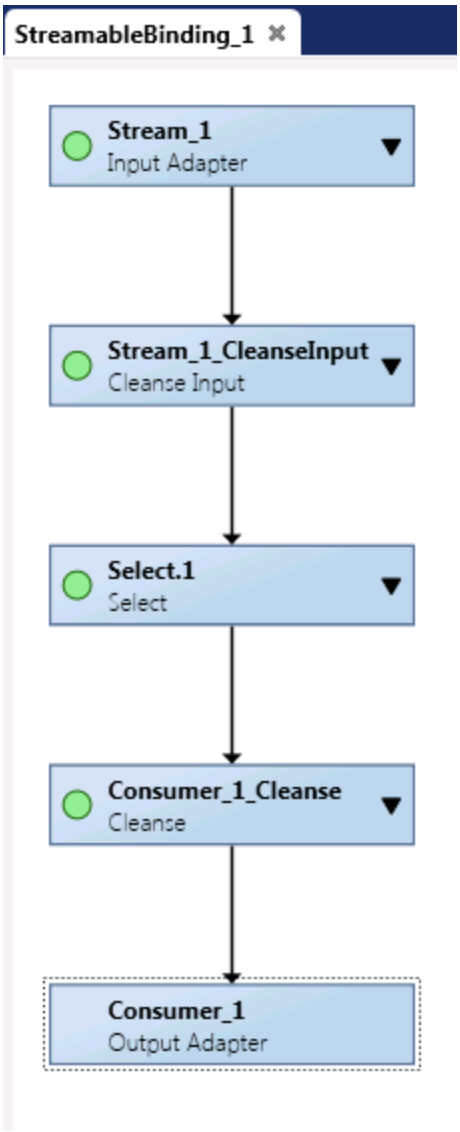
1. As before, right-click on "SimpleDAStreamInsightApplication" project in the Solution Explorer, and select "Debug -> Start new instance". This will build and run the program. Observe the results in the console window, but let the application run.
2. From the Start menu, select "Microsoft StreamInsight 2.3 -> StreamInsight Event Flow Debugger".
3. From the debugger menu, select "File -> Connect to Server... (Ctrl+K)" command.
4. In the "Connect to Server..." dialog, enter the address as http://localhost/StreamInsight/MyStreamInsightServer , and press OK.



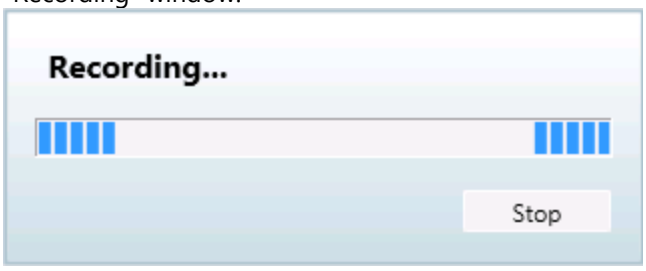
5. In the Object Explorer pane of the StreamInsight Event Flow Debugger, navigate and expand nodes as follows:



6. Under “Applications -> serverApp -> Processes -> ServerProcess -> Queries”, right-click the StreamableBinding_1 query node, and select the “Display Query” command. This will display the query flow chart:



7. Press the “Start Recording Events” button on the toolbar. If it happens to be disabled (grayed out), select the “Display Query” command again, as in the previous step. 
8. Let the events be recorded for some time (e.g. about a minute), and then press the “Stop” button on the “Recording” window:



Alternatively, you can press the “Stop Recording Events” button on the toolbar. 
9. You can now view the events at various stages of the query processing, by clicking the downwards-pointing triangle in any of the query operators. For example, events on the “Select.1” operator may look like this:

You can sort and/or filter the events according to various criteria, and the debugger also gives you analysis options, such as Root Cause Analysis or Event Propagation Analysis.

10. Close the StreamInsight Event Flow Debugger.
11. Switch to the console output window of the application, and press Enter to stop it and close the window.

12.2.3 StreamInsight Option Fundamentals

This chapter describes the fundamental concepts used within the StreamInsight Option for OPC Data Client.

12.2.3.1 StreamInsight Option Product Parts

12.2.3.1.1 StreamInsight Option Assemblies

The StreamInsight Option for OPC Data Client uses the core OPC Data Client assemblies described in the “OPC Data Client User's Guide”.

In addition, following assemblies are needed to develop programs with the StreamInsight Option for OPC Data Client:

Assembly Name or File	Title	Description
OpcLabs.OpcComplexEventProcessing	OPC Labs OPC Complex Event Processing	StreamInsight support for OPC-DA, OPC-A&E and OPC-UA (Unified Architecture) specifications

The assemblies are written to target Microsoft .NET Framework 4.5.

As with the core OPC Data Client Assemblies, the StreamInsight Option for OPC Data Client assemblies are also delivered with XML comment files and ReSharper annotations.

12.2.3.1.2 StreamInsight Option Examples

Example code, projects and solutions are available from the Start menu or the Launcher application. You can also access them from the file system, in the Examples-NET subfolder of the installation folder.

A separate Examples chapter in the StreamInsight Option documentation contain list of all examples and their purpose.

12.2.3.1.3 StreamInsight Option Documentation and Help

The StreamInsight Option documentation is part of the OPC Data Client User's Guide and Reference. You can also access it from the Start menu or the Launcher application.

In addition, there is IntelliSense and Object Browser information available from Visual Studio environment.

The help contents integrates Microsoft Visual Studio Help (Microsoft Help Viewer 2 format).

12.2.3.2 StreamInsight Option Usage

The StreamInsight Option for OPC Data Client is suitable for use from within wide range of tools or languages based on Microsoft .NET Framework. Most commonly, you will use from inside C# or Visual Basic project in Visual Studio.

The project can be e.g. a class library, a console application, Windows Forms or WPF application, or a Web service.

12.2.3.2.1 How the StreamInsight Option Works

The main purpose of StreamInsight Option for OPC Data Client is to make it easy to create StreamInsight event sources that generate data that come from underlying OPC servers. These event sources generate StreamInsight events that are then processed inside StreamInsight like any other events.

In order to bring OPC data to StreamInsight, you first reference the necessary OPC Data Client assemblies in your project. Then, in your code, you create instances of OPC “observables” – the reactive objects that call a specified method whenever new data is available. Several types of “observables” are available, for OPC Data Access, OPC Alarms and Events, and OPC Unified Architecture, respectively. The observables are then used to create event sources inside StreamInsight, by converting them to temporal streams.

12.2.3.2.2 Referencing the Assemblies

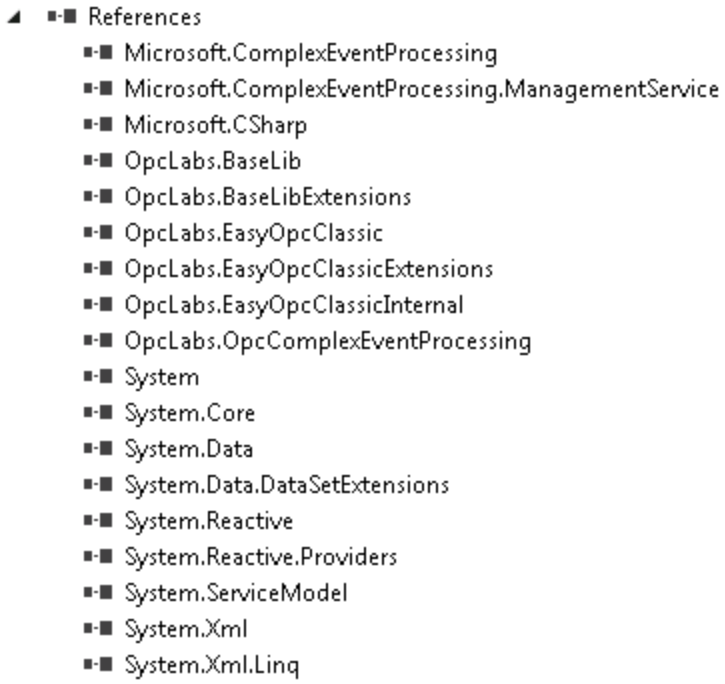
In order to develop applications with StreamInsight Option for OPC Data Client, your project first needs to reference the appropriate assemblies.

Besides the “standard” assemblies used in your project, and the assemblies you already have referenced for various specific purposes of yours, you will to reference:

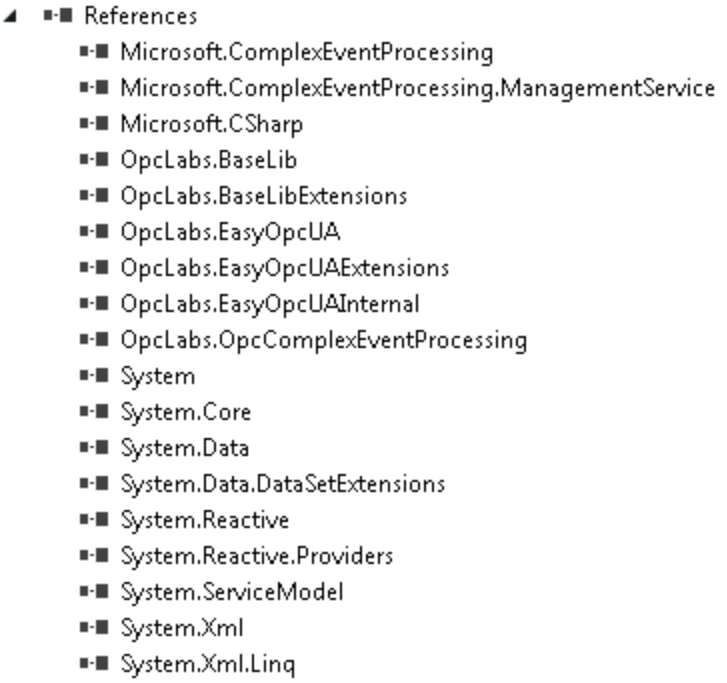
- The OPC Data Client assemblies needed for various tasks. They are described in the Product Parts section of the “OPC Data Client User's Guide” document.
- The StreamInsight Option for OPC Data Client assembly or assemblies (as listed in “Product Parts”) in this document
- Microsoft Reactive Extensions Assemblies (installed with Microsoft StreamInsight).
- Supporting Microsoft StreamInsight assemblies.

To make this a bit simpler, we have put together some illustrations below that summarize the assemblies that are typically referenced in your projects.

In a StreamInsight project that uses OPC “Classic”, the references list should look similarly to this:



In a StreamInsight project that uses OPC Unified Architecture, the references list should look similarly to this:



Note: In OPC Data Client version 5.40 and later, ignore the EasyOpcClassicExtensions, EasyOpcUAInternal and EasyOpcUAExtensions assemblies above – they are no longer needed, and no longer present.

You can also combine the two lists together, if you application needs both OPC "Classic" and OPC Unified Architecture servers.

12.2.3.2.3 Example Walkthrough

In this chapter, we will explain the usage of StreamInsight Option for OPC Data Client on one of the examples that comes with the product. It is the SimpleDAStreamInsightApplication project in C#. This example uses OPC Data Access; other examples, for OPC Alarms and Events, or OPC Unified Architecture, are quite similar.

In order to make it easy to understand the specifics of StreamInsight Option for OPC Data Client, and stay away from parts that common to all StreamInsight application, we have created this set of simple examples by deriving from the examples that Microsoft uses to teach StreamInsight. We can thus stay away from explaining the information that is already available by Microsoft.

Our example will expose an embedded server from inside a console application, and also run the query and display the results. It is based on Microsoft's "StreamInsight Example: Server - Exposing an Embedded Server", http://msdn.microsoft.com/en-us/library/hh995352(v=sql.111).aspx . It is recommended that you study the Microsoft's example first.

In This Topic

Introductory Steps

The first steps are the same:

- Create the (StreamInsight) server instance
- Create the (StreamInsight) application

Define and Deploy a Source

In the next step, an input source is defined and deployed to the (StreamInsight) server with a name so that it can be used by other StreamInsight clients. In this example, the data is a simple temporal stream of point events generated by the OPC Data Access server.

The DAItemChangedObservable.Create<int> method creates an observable that generates a sequence of EasyDAItemChangedEventArgs<int> objects. Each notification contains, among other information, the value of a given integer OPC item (if available), as it is delivered by the OPC server. Because the EasyDAItemChangedEventArgs<int> objects are not directly suitable as StreamInsight event payloads, we convert them to DAItemChangedPayload<int>, which is pre-made payload object provided by the StreamInsight Option for OPC Data Client.

Define and Deploy a Source

```
// DEFINE a simple SOURCE (returns a point event every second)
const string machineName = "";
const string serverClass = "OPCLabs.KitServer.2";
const string itemId = "Simulation.Incrementing (1 s)";
var observable = DAItemChangedObservable.Create<int>(machineName,
    serverClass, itemId, 100);
var mySource = myApp
    .DefineObservable(() => observable)
    .ToPointStreamable(
```

```
            eventArgs =>
                PointEvent.CreateInsert(DateTimeOffset.Now,
                (DAItemChangedPayload<int>)eventArgs),
            AdvanceTimeSettings.StrictlyIncreasingStartTime);
```

Compose a Query over the Source

Next, compose a query over the input source. The query uses LINQ as the query specification language. In this example, the query returns the events where the value of the OPC item is an even number.

Compose a Query over the Source

```
    // Compose a QUERY over the source
    // (return every event carrying even data value)
    var myQuery = from e in mySource
                  where e.VtqPayload.Value % 2 == 0
                  select e;
```

Technically, this definition translates to a filter operator that drops all events from the sequence that do not fulfill the filter predicate (where e.VtqPayload.Value % 2 == 0) and returns the event value. For more information about LINQ query operators, see Using StreamInsight LINQ.

Define and Deploy a Sink

Next, an output sink is created that can be bound to the query and process the resulting sequence. In this example, a simple function is created that simply writes the stream values to the console. The DAItemChangedPayload<int> class has a convenient ToString() method that returns the relevant payload information nicely formatted for output.

Define and Deploy a Sink

```
    // DEFINE a simple observer SINK (writes the value to the server console)
    var mySink = myApp.DefineObserver(() =>
        Observer.Create<DAItemChangedPayload<int>>(
            payload => Console.WriteLine("sink_Server..: {0}", payload)));
```

The sink is then deployed to the server with a name, in the same way as in the original Microsoft example.

Bind and Run the Query and Sink

The remainder of the example is not doing anything specific to OPC. The observable query is bound to the observer output sink, and the query is then run in a process in the server. This process continues to run until the user stops it by typing in the console.

12.2.3.3 Creating OPC Event Sources

The power of StreamInsight Option for OPC Data Client comes from its OPC event sources. On the surface, they are just

handful of relatively “small” objects, with uncomplicated interface. Below the surface, however, there is an enormous amount of functionality, providing things such as type conversions, error handling, connection pooling, connection persistence, optimizations, OPC compliance, wide interoperability, and security.

12.2.3.3.1 OPC Observables

The StreamInsight Option for OPC Data Client provides OPC event sources in form of OPC observables, i.e. .NET reactive object with IObservable interface. In order to simplify creation of these OPC event sources, static classes with various overloads of Create method are provided:

- In order to create OPC Data Access event source, call one of the DAItemChangedObservable.Create<TValue> method overloads, passing it arguments such as the OPC server ProgID, OPC item ID, and requested update rate.
- In order to create OPC Alarms and Events event source, call one of the AENotificationObservable.Create method overloads, passing it arguments such as the OPC server ProgID, notification rate, and subscription filter.
- In order to create OPC Unified Architecture (data) event source, call one of the UADataChangeNotificationObservable.Create<TValue> method overloads, passing it arguments such as the OPC server endpoint, OPC node ID, and sampling interval.

For more information and details on the OPC Observables, see the appropriate chapters (separately for OPC-DA, OPC-A&E, and OPC-UA) in the Reactive Programming Model section of this document.

12.2.3.3.2 Errors in OPC Sequences

When there is an error in communication with the target OPC server, the OPC observable sends a message (one of the XXXXEventArgs objects) with following characteristics:

- The Exception property is non-null, and contains an exception object that describes the error.
- The ErrorId property optionally contains a string that identifies the error.
- The ErrorMessage property contains a string that describes the error.
- The Succeeded property is equal to ‘false’.
- Other properties that normally contain the actual notification data may not contain valid data, or may be null.

Your StreamInsight application code needs to be prepared for such messages. As a minimum, if you want to ignore error situations, you need to filter out the error messages, and in case of error, do not access the fields of the XXXXEventArgs that do not contain valid data.

When you use the pre-defined OPC Event Payload classes (described further below), the conversion code already takes the possible errors into account, and behaves accordingly, by filling in the payload fields with default values, setting negative error code in the payload, etc.

Note that the errors described here are NOT errors in the sense of calling the OnError method of the IObserver interface. In fact, the OPC observables never call OnError, because that would mean an irrecoverable error and a completion of the sequence. The OPC errors are, however, potentially transient and recoverable, and must therefore be a part of normal notifications.

12.2.3.3.3 Other Special Cases in OPC Sequences

Leaving error situations aside, you still need to be careful when interpreting properties of XXXXEventArgs objects provided by the OPC observables. For precise description, study the corresponding topics int the “OPC Data Client User's Guide”

and the reference documentations. Here are some basic points to be aware of:

- In OPC Data Access, the EasyDAItemChangedEventArgs<T>.TypedVtq.TypedValue property does not contain valid data when the quality of the VTQ denotes a "Bad" quality.
- In OPC Alarms and Events, the EasyAENotificationEventArgs.EventData property does not contain valid data for in a notification that signifies a re-gained connection to the OPC server, or in a notification that indicates that a subscription refresh is complete.
- In OPC Unified Architecture, the EasyUADataChangeNotificationEventArgs<T>.TypedAttributeData.TypedValue property does not contain valid data when the status code of the attribute data denotes a "Bad" status.

When you use the pre-defined OPC Event Payload classes (described further below), the conversion code already takes these special cases into account, e.g. by filling in the payload fields with special values.

12.2.3.4 OPC Event Payloads

When you develop application for Microsoft StreamInsight, you are responsible for defining *event payloads*, i.e. .NET data structures that hold the data associated with StreamInsight events. The fields in the payload are user-defined and their types are based on the .NET type system.

There are various rules and limitations regarding the design of event payloads for Microsoft StreamInsight. For details, refer to StreamInsight documentation, e.g. the "Event Structure" article (http://msdn.microsoft.com/en-us/library/ee378905(v=sql.111).aspx).

The objects provided as notifications by OPC observables are not directly suitable as StreamInsight event payloads. You need to define the event payload structure for StreamInsight according to the needs of your application. There is no general rule as to what to use as StreamInsight event payload – it all depends on the specifics of the application. You can, for example

- Pick up a single field from the notification provided by the OPC observable – for example, the actual data value. This may be very useful in applications that only need the data values, but beware of special cases such as communication errors etc.
- Create a custom .NET data structure, and define its fields and organize the data in it precisely as your application requires. Then, convert the notifications provided by the OPC observable, to instances of this >NET event payload structure.
- Use ready-made OPC Event Payload classes provided by the StreamInsight Option for OPC Data Client. These classes are already designed to conform to StreamInsight rules, and payload instances can be easily created by converting them from the notifications coming from OPC observables.

The following paragraphs describe the OPC Event Payload classes for various OPC specifications.

12.2.3.4.1 Common Payload Characteristics

All OPC Event Payload classes contain a string ErrorId field. This field is empty for successes, and non-empty for errors.

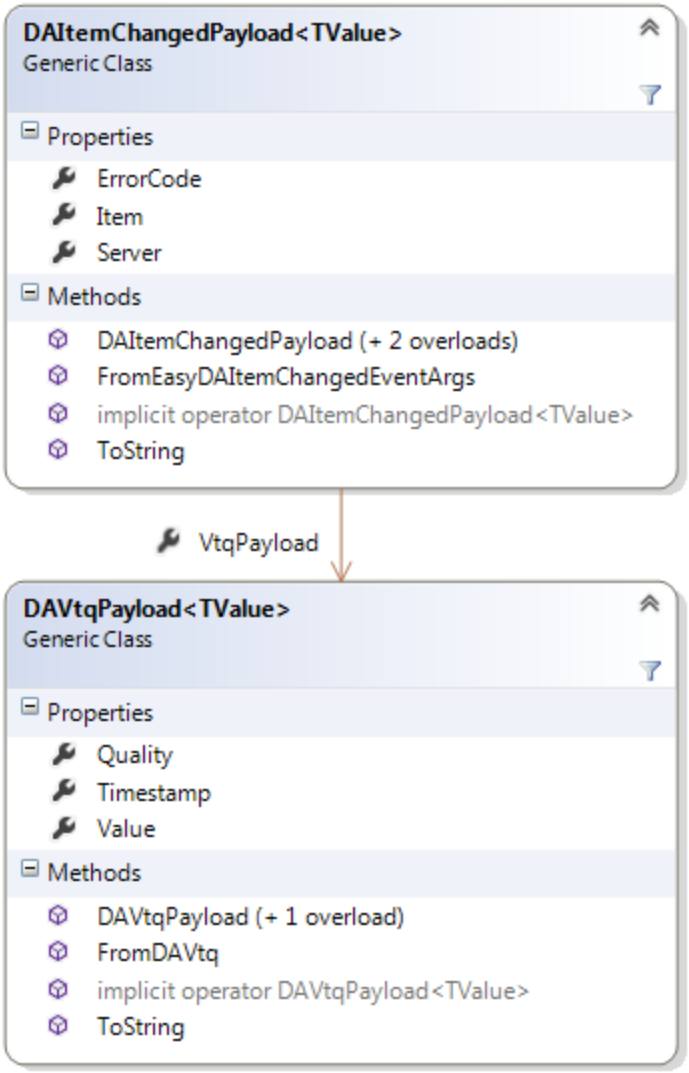
The OPC Event Payload classes contain no more information about the error (such as the error message), due to possible negative effect on the payload size, memory consumption, and performance. If your application needs extra information such as the error message strings, derive your payload from one of the OPC Event Payload classes and extend it, or use your own even payload structure.

Each OPC Event Payload class contains a method (called FromXXXX) that creates it from a non-payload object that comes natively from the OPC observable, and a corresponding implicit conversion operator.

For easy viewing, logging and similar purposes, all OPC Event Payload classes contain a ToString() method that formats the contents of relevant fields of the payload into a readable string.

12.2.3.4.2 Payloads for OPC Data Access

The event payload class for OPC Data Access is the DAItemChangedPayload<TValue>, where TValue is the type of the OPC values. This class contains an embedded DAVtqPayload<TValue> object, which contains the actual OPC value, timestamp, and quality. This object is always present; in case of errors, it contains default values.



In addition, there are Server and Item string properties, identifying the source of the event. For a single OPC observable, they always contain the same value, but they become useful once you start combining events from different OPC sources into one stream.

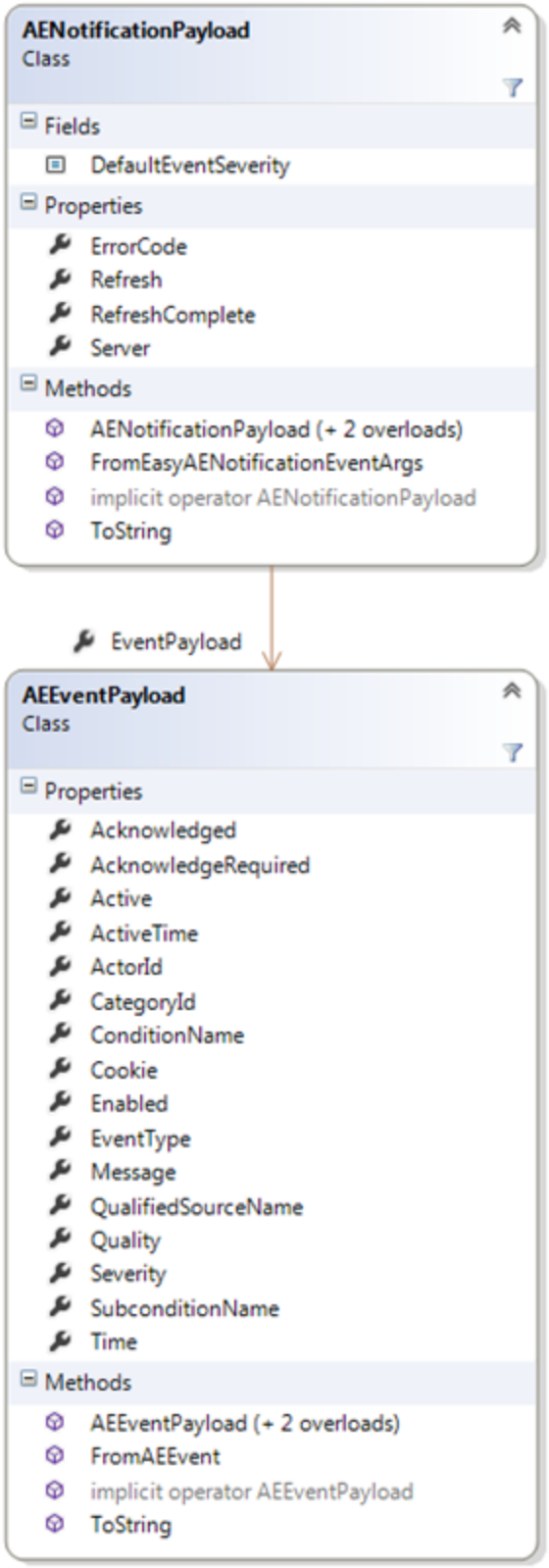
12.2.3.4.3 Payloads for OPC Alarms and Events

The event payload class for OPC Alarms&Events is the AENotificationPayload class. This class contains an

embedded AEEventDataPayload object, which contains the actual OPC event. This object is always present; in case of errors and special cases, it contains default or pre-defined values.

A special event priority value is used when no OPC event is present (in case of errors, but also re-gained connections, and a completed refresh). Normal OPC event priorities are in the range from 0(1) to 1000. When creating a payload for a notification where no actual OPC event is present, an event priority value of 9999 is used. The value 9999 is chosen to be higher than the severity of any OPC-A&E event, yet still in a 4-digit range.

The payload class does not contain any fields for OPC server-specific event attributes. If you need to process the OPC event attributes in StreamInsight, derive your own class from AEEventDataPayload, add the fields you need, and develop a code that extracts the attributes from the AEEventData object (in EasyAENotificationEventArgs.EventData).

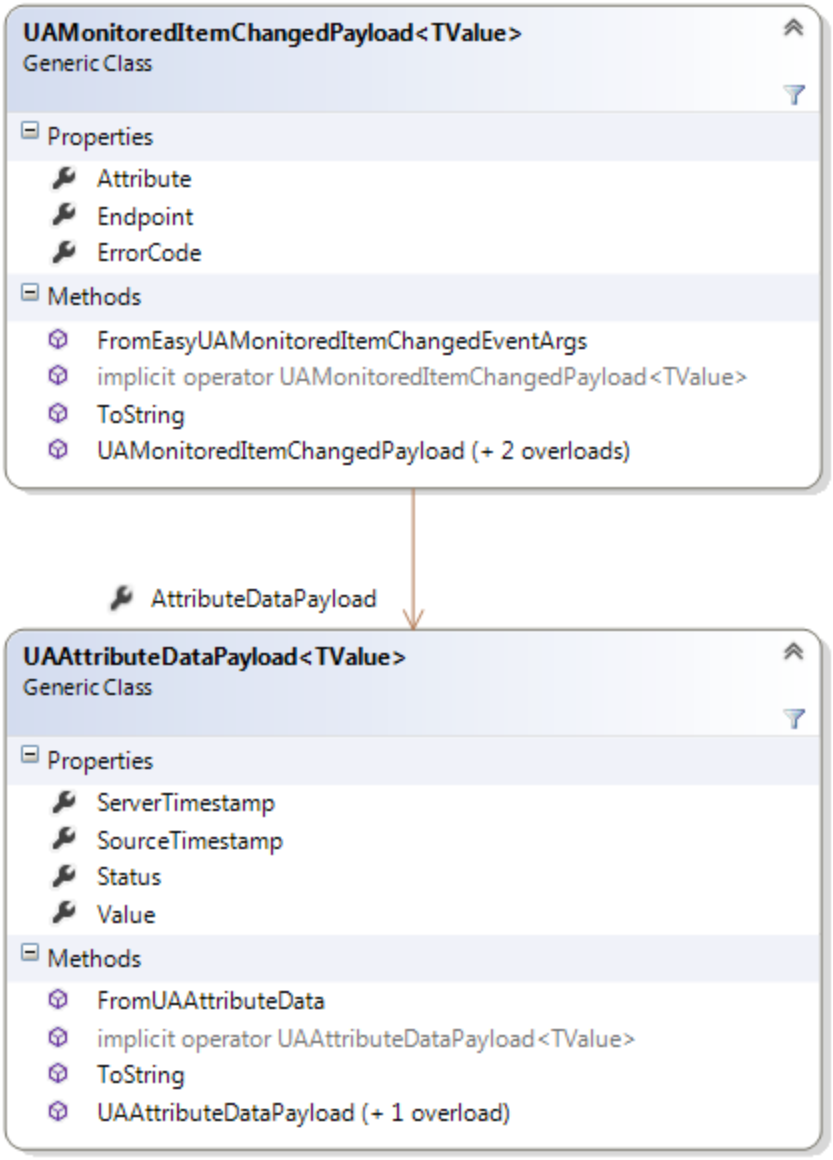


In addition, there is a Server string property, identifying the source of the event. For a single OPC observable, it always contains the same value, but it becomes useful once you start combining events from different OPC sources into one

stream.

12.2.3.4.4 Payloads for OPC Unified Architecture

The event payload class for OPC Unified Architecture is the UADataChangeNotificationPayload<TValue>, where TValue is the type of the OPC values. This class contains an embedded UAAttributeDataPayload<TValue> object, which contains the actual OPC value, timestamps, and status. This object is always present; in case of errors, it contains default values.



In addition, there are Endpoint and Attribute string properties, identifying the source of the event. For a single OPC observable, they always contain the same value, but they become useful once you start combining events from different OPC sources into one stream.

12.2.3.5 Time in OPC and StreamInsight

In StreamInsight, the time that comes with events has to be monotonic, i.e. it cannot go back. More precisely, at the moment a CTI (Current Time Increment) event with certain time is stored into a stream, your application has committed to this point in time, and cannot store any events into the stream that have time that precedes the last CTI time. And, you must generate the CTI events, because before committing to certain time (by storing a CTI), StreamInsight cannot perform processing on the incoming stream events, because it does not know for sure that they represent the entirety of the information to be processed.

The time presents a challenge in many StreamInsight applications, and the combination of StreamInsight and OPC brings some additional points to consider. We try to present a summary of them below.

12.2.3.5.1 Time Synchronization in OPC

In OPC "Classic" (COM-based) specifications, such as OPC Data Access or OPC Alarms and Events, there is little or nothing specified about the times the OPC servers should use, and the timestamps that come with the OPC data. Specifically, there is no requirement that the OPC servers must have proper and synchronized time.

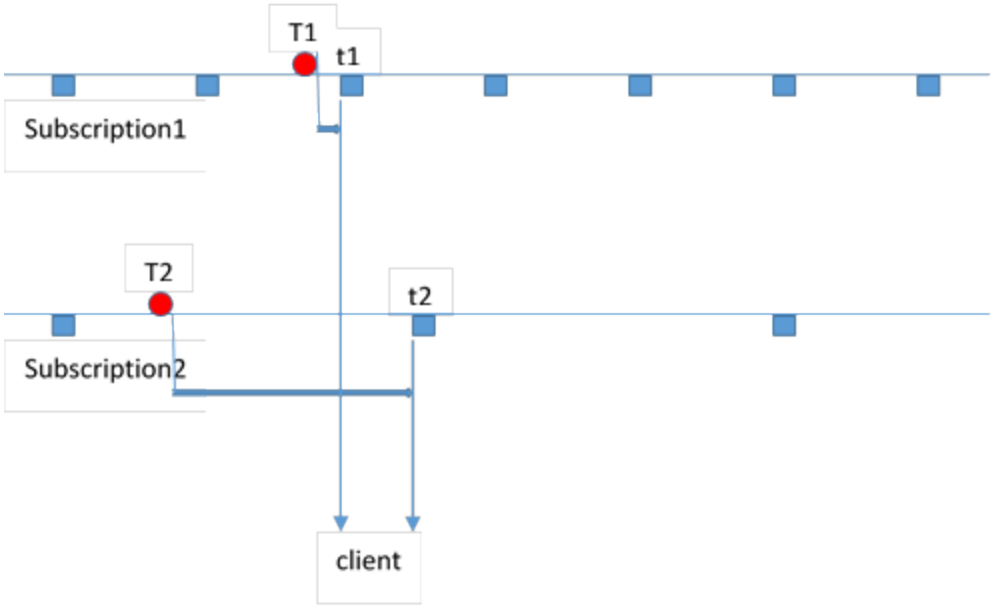
This becomes an issue when your application starts combining data from multiple OPC servers, and even more if they run on different computers. Unless you can guarantee their time synchronization by some application-specific means, you must assume that their times might be out of sync. Events that come in sequence from different servers may have their timestamps in decreasing order instead.

The OPC Unified Architecture specification has given more though to time, and (with some simplification), we may say that it requires the times be synchronized on both sides of the communication (between the OPC server and the OPC client), although it does not provide a mechanism to do, and again relies on external means to achieve.

The OPC-UA requirements consequently mean that if the OPC client (such as your application with StreamInsight Option for OPC Data Client) communicates with multiple OPC servers, all their times should be synchronized. In reality, however, not all systems will be set according to OPC specifications, and adding to that, the synchronization required by OPC-UA is not precise, and there may be several seconds' difference between the times on various computers. In StreamInsight, however, the time monotony requirement is a strict one – you cannot go back in time even by a fraction of a second.

12.2.3.5.2 Time in OPC Subscriptions

Even if the times used by all OPC servers were perfectly synchronized, one needs to consider how the OPC subscriptions are realized and what effect it has on the time monotony. In OPC, data change or event delivery is not usually immediate, for performance (throughput) reasons. Notifications are transferred in larger chunks, and only after certain time elapses, so that they can accumulate.



Let's have a look at following situation where we have two subscription, using different notification rates:

The red circles denote times when new source data appear. The blue squares denote times when the server delivers notifications to the client.

In this case, following occurrences happen in real time sequence:

1. At time T2, the new source data appear on Subscription2 (which has slower notification rate).
2. At time T1, the new source data appear on Subscription1 (which has faster notification rate).
3. At time t1, the data from Subscription1, carrying a T1 timestamp, are delivered to the client.
4. At time t2, the data from Subscription2, carrying a T2 timestamp, are delivered to the client.

As a result, the event that has appeared later (on T1) is delivered to the client earlier than the event that has appeared sooner (on T2). If the StreamInsight application wanted to use the timestamps of the source data, the time monotony requirement would be broken.

In addition, even with a single subscription, the OPC "Classic" server may send data with timestamps in reversed time order, albeit that would be quite rare and probably an indication of a buggy server.

12.2.3.5.3 Conclusions

The above findings can be summarized briefly as follows:

Microsoft StreamInsight has strict requirements on time monotony, which OPC cannot fulfill directly.

While the above statement may sound somewhat pessimistic, there are several ways how to adapt OPC times to StreamInsight, and we will mention some of them here. There is also material available from Microsoft and on the Web from various sources that deals with such issues, because other data sources that are used to feed events into StreamInsight deal with the same or similar issues.

Here are some options that can be used in various situations, either alone, or in combination:

- Use current system time on the client side for StreamInsight time, i.e. do not use OPC timestamps at all. This is the

approach used the in the simple examples shipped with the product, and in this document. The OPC timestamps become fields of the event payload, and can therefore have any valid value.
- You may use only single notification rate, or avoid combination of events from different sources before you advance the StreamInsight time.
- Use various AdvanceTimeSettings available in StreamInsight. For example, you may place an upper bound for how far back from the previous CTI the start time of the future event can be. Using AdvanceTimePolicy, you can also choose whether incoming events with start time before the current application time are dropped, or simply adjusted to the current application time.

The way that you resolve the issue of adapting OPC times to StreamInsight depends completely on the needs of your application, and there is no single approach that fits all applications.

12.2.4 StreamInsight Option Examples

The OPC Data Client installation contains usage examples for the StreamInsight Option in various programming languages. The examples are being updated and enhanced more frequently than the base product, so if you like them, please check for newer builds of OPC Data Client from time to time.

12.2.4.1 List of StreamInsight Option Examples

The examples assume an existence of a local StreamInsight instance named Instance1.

The examples are currently provided mainly for Microsoft Visual Studio 2017. They can all be automatically converted to later versions of Microsoft Visual Studio.

The examples are targeting Microsoft .NET Framework 4.7. In Visual Studio, it is possible to re-target the projects to newer framework versions (e.g. 4.7.2) by changing the appropriate setting in the properties of the project.

Visual Studio solution with examples in C# is available from the Start menu or the Launcher application.

	Visual C#	Visual Basic	Visual F#	Visual C++
SimpleAEStreamInsightApplication (Section 13.1.4.1.6): This example shows how to create an OPC Alarms&Events event source, and query it for events with severity 20 or higher.	✔			
SimpleDAStreamInsightApplication (Section 13.1.4.1.7): This example shows how to create an OPC Data Access event source, and query it for events carrying even data value.	✔			
SimpleUAStreamInsightApplication (Section 13.1.4.1.10): This example shows how to create an OPC Unified Architecture data event source, and query it for events carrying even data value.	✔			

12.2.5 StreamInsight Option Application Deployment

This chapter describes how to deploy applications developed with use of StreamInsight Option for OPC Data Client.

For application deployment, you need to consider a combination of rules that exist for OPC Data Client, and for

StreamInsight.

To properly deploy the OPC Data Client part, please refer to the “Application Deployment” chapter in this document. All information given there (for .NET) also applies to applications created with the StreamInsight Option for OPC Data Client. In Addition, you need to deploy the assemblies that are specific to the StreamInsight Option for OPC Data Client, but you do it in the same way as with other assemblies. You need to deploy one additional assembly:

- **OpcLabs.OpcComplexEventProcessing.dll**

For deploying the StreamInsight server, and the StreamInsight entities into a StreamInsight server, please see the appropriate StreamInsight documentation.

12.2.6 Additional Resources for StreamInsight Option

If you are migrating from earlier version, please read the “What’s New” document (available from the Start menu or the Launcher application).

Study the Reference documentation (also available from Start menu).

Explore the examples and bonus tools and materials installed with the product.

Check the vendor’s Web page for updates, news, related products and other information.

13 Examples

OPC Data Client comes with large number of examples, in multiple programming languages and tools.

Some of these examples are **installed with the product (Section 13.1)**, and some are **included in the documentation (Section 13.2)**. There is also a huge overlap - many (but not all) examples appear both in the product, and are also listed in the documentation. Examples that were written for the documentation are typically somewhat shorter pieces of code, to illustrate a specific point, and are included in the product installation for convenience. Larger standalone examples ship primarily with the product, but pieces of them has been includer in the documentation for easier understanding.

Additional examples can be found in the Knowledge Base.

We are continuously adding new examples, and extending our coverage of programming languages tools. If you think that an example is missing, let us know and we will try to add it.

13.1 Examples Installed with the Product

This documentation section consists of several parts:

- **COM Examples (Section 13.1.1)**, and
- **.NET Framework Examples (Section 13.1.2)**, and
- **.NET Standard Examples (Section 13.1.3)**, and
- **Copies of Selected Installed Examples (Section 13.1.4)**.

Depending on whether you develop OPC applications for Microsoft .NET, or are using Microsoft COM, please refer to the corresponding part.

Examples are provided for OPC "Classic" (COM-based) specifications, OPC XML-DA, and OPC Unified Architecture (OPC UA).

The OPC Data Client installation contains usage examples in various programming languages. The examples are being updated and enhanced more frequently than the base product, so if you like them, please check for newer builds of OPC Data Client from time to time.

Note that the examples are different for OPC Data Client.NET and OPC Data Client-COM. The text is marked with corresponding .NET or COM icon further below.

> For those who want to use the command-line interface, a "Command Prompt From Here" links are installed into example folders that are related to command prompt, such as JScript Examples in WSH, PHP Console Examples, and VBScript Examples in WSH (WIndows Script Host). These links allow you to switch from the Windows Explorer to a CMD window with the current directory set to the chosen place, for quick experiments and exploration.

13.1.1 COM Examples

The examples are provided to cover wide range of tools and languages used in development of industrial applications.

For OPC Data Access, the examples show:

- reading and writing OPC items,
- working with multiple items at once,
- error handling,
- subscribing for item changes, and managing subscriptions,
- getting property values,
- browsing for OPC servers, branches and leaves, and properties,
- obtaining information about OPC servers,
- processing events,
- user interface for browsing,
- and more.

For OPC Alarms and Events, the examples show:

- subscribing to events,
- event filtering,
- error handling,
- subscribing for item changes, and managing subscriptions,
- browsing for OPC servers, areas and sources,
- querying for event categories, conditions and attributes,
- obtaining information about OPC servers,
- processing events,
- acknowledging conditions,
- refreshing event subscriptions,
- and more.

For OPC Unified Architecture, the examples show:

- reading and writing attributes of OPC nodes,
- working with multiple nodes/attributes at once,
- error handling,
- subscribing for monitored item changes, and managing subscriptions,
- discovering OPC servers,
- browsing for OPC nodes,
- processing events,
- user interface for browsing,
- and more.

Following categories of COM examples are available , and corresponding shortcuts exist in the Start menu or the Launcher application:

- **Free Pascal Examples (Lazarus) (Section 13.1.1.1)**
- **JScript Examples (IE, WSH) (Section 13.1.1.2)**
- **Object Pascal Examples (Delphi) (Section 13.1.1.3)**
- **Perl Examples (Section 13.1.1.4)**
- **PHP Examples (Section 13.1.1.5)**
- **PowerScript Examples (PowerBuilder) (Section 13.1.1.6)**
- **Python Examples (Section 13.1.1.7)**
- **REALbasic (Xojo) Examples (Section 13.1.1.8)**
- **VBA Examples in Excel (Section 13.1.1.9)**
- **VBScript Examples (ASP, IE, WSH) (Section 13.1.1.10)**
- **Visual Basic Examples (VB 6.0) (Section 13.1.1.11)**

- Visual C++ Examples
- **Visual FoxPro Examples (Section 13.1.1.13)**
- Xbase++ Examples

13.1.1.1 Free Pascal Examples (Lazarus)

The examples are currently provided for Lazarus 1.6 (FPC 3.0).

- UADocExamples: Contains all Free Pascal examples for OPC-UA that are given in the Reference documentation.

13.1.1.2 JScript Examples (IE, WSH)

JScript Examples in IE

In This Topic

The examples here run inside Internet Explorer, i.e. directly on client. No Web server is required. The EasyOPC-DA component must be installed on the client side, and the OPC server(s) must be accessible locally or via DCOM from the computer where EasyOPC-DA component resides.

- ReadAndDisplayValue_JScript.htm: Reads and displays an OPC item value.

JScript Examples in WSH

The examples here run in Windows Script Host (WSH), e.g. from Windows Command Prompt.

- **DocExamples** folder: Contains all JScript examples for OPC Classic that are given in the Reference documentation.
- **ReadAndDisplayValue.js**: Reads and displays an OPC item value.
- **UADocExamples** folder: Contains all JScript examples for OPC-UA that are given in the Reference documentation.

> For those who want to use the command-line interface, a "Command Prompt From Here" link is installed into the examples folder. This link allows you to switch from the Windows Explorer to a CMD window with the current directory set to the chosen place, for quick experiments and exploration.

13.1.1.3 Object Pascal Examples (Delphi)

The examples in Object Pascal are currently provided for Delphi 10.3. In case you are not able to directly load the projects into earlier versions of Delphi, the actual source code is likely to work just well with no or minor modifications, so you just need to re-create the "envelope" project files.

In earlier versions, we have tested the examples with Delphi XE7, and also briefly with Delphi XE8 and Delphi 10 Seattle. In OPC Data Client 2019.2, we have tested the examples with Delphi 10.3 Rio (10.3.2).

- **DocExamples**: Contains all Delphi examples for OPC Classic that are given in the Reference documentation.
- **EasyOpcUADemo**: Shows how to make a desktop windowed OPC UA client application, mimicking the example

that is available in C# and VB.NET. The application demos the most important product functions, including event-based subscriptions.
- **Imports**: Contains Delphi components created by importing OPC Data Client type libraries.
- **ReadAndDisplayValue**: The simplest form. Reads and displays an OPC item value.
- **UADocExamples**: Contains all Delphi examples for OPC-UA that are given in the Reference documentation.

13.1.1.4 Perl Examples

The Perl examples were tested in ActivePerl v5.14.2.

- **ReadAndDisplayValue.pl**: The simplest console application for OPC "Classic". Reads and displays an OPC item value.
- **SubscribeToValue.pl**: Subscribes to changes of an OPC Classic item, and displays the values.
- **UAReadAndDisplayValue.pl**: The simplest console application for OPC UA. Reads and displays an OPC item value.
- **UASubscribeToValue.pl**: Subscribes to changes of an OPC UA monitored item, and displays the values.

13.1.1.5 PHP Examples

Various PHP versions can be downloaded from http://php.net/downloads.php . PHP for Windows is available from https://windows.php.net/download/ .

In some earlier versions, the examples were tested in PHP v5.6, under command-line interpreter and Internet Information Server (IIS). In OPC Data Client 2019.2, the examples were tested in PHP 7.3.10 (VC15 x64 Thread Safe).

In This Topic

The examples use the COM extension for PHP. This extension can be enabled e.g. in following ways:

a) Add following to the php.ini file:

```
extension=ext/php_com_dotnet.dll
```

b) Use following in the php.ini file:

```
extension_dir =  "ext"
extension=php_com_dotnet.dll
```

c) From IIS, under Internet Information Services (IIS) Manager, by PHP Manager: Use

```
[PHP_COM_DOTNET]
extension=php_com_dotnet.dll
```

PHP Console Examples

In order to run the command-line examples, use

```
PHP.EXE –f "filename"
```

For easier troubleshooting of the command-line scripts, it is recommended that you enable the display_error option, e.g. in one of the following ways:

a) Use the following on the command line:

-d display_errors

b) Add following to the php.ini file:

display_errors = On

c) Copy the file php.ini-development over your php.ini file.

The provided examples are:

- **Console\DocExamples** folder: Contains all command-line PHP examples for OPC Classic that are given in the Reference documentation.
- **Console\UADocExamples** folder: Contains all command-line PHP examples for OPC-UA that are given in the Reference documentation.

> For those who want to use the command-line interface, a "Command Prompt From Here" link is installed into the examples folder. This link allows you to switch from the Windows Explorer to a CMD window with the current directory set to the chosen place, for quick experiments and exploration.

PHP Web Examples

In order to run the Web-based examples, you need to configure an IIS virtual directory that points to the physical directory where the examples are installed.

Examples in this folder are Web server scripts: the page, including OPC data, is generated on the server. The EasyOPC component must be installed on the server side, and the OPC server(s) must be accessible locally or remotely from the computer where EasyOPC component resides.

- **Web\ReadAndDisplayValue.php**: The simplest Web application. Reads and displays an OPC item value.
- **Web\ReadAndDisplayMultipleItems.php**. Reads four values at once and displays their contents. Aimed at showing how PHP can pass input and output arguments that are VARIANT arrays.
- **Web\UADocExamples** folder: Contains all Web server PHP examples for OPC-UA that are given in the Reference documentation.

13.1.1.6 PowerScript Examples (PowerBuilder)

The PowerScript examples were developer and tested in Appeon PowerBuilder 2017 R3.

The QuickOpcPowerBuilderExamples workspace contains following targets:

- **DocExamples**: Contains all PowerBuilder examples for OPC Classic that are given in the Reference documentation.
- **UADocExamples**: Contains all PowerBuilder examples for OPC-UA that are given in the Reference documentation.

13.1.1.7 Python Examples

With some earlier OPC Data Client versions, the examples were tested in Python 3.4.2, with pywin32-219 extension (Build 219). With OPC Data Client 2018.3 and 2019.2, the examples were tested in Python 3.6 (64-bit), and pywin32 (223) extension.

- **DocExamples** folder: Contains all Python examples for OPC Classic that are given in the Reference documentation.
- **UADocExamples** folder: Contains all Python examples for OPC-UA that are given in the Reference documentation.


In C:\Python*XX*\Scripts, do:
`pip install pypiwin32`

13.1.1.8 REALbasic (Xojo) Examples

Note: We do not regularly update these examples with new versions. You may have to modify the code to use newer versions.

The REALbasic (Xojo) examples were originally created in REAL Studio 2010r5, and converted to REAL Studio 2011r1.1, and later versions.

- **Xojo2014\ReadAndDisplayValue.xojo_binary_project**: The simplest form, for OPC "Classic". Reads and displays an OPC item value.
- **Xojo2014\UAReadAndDisplayValue...**: The simplest form, for OPC UA. Reads and displays an OPC item value.

13.1.1.9 VBA Examples in Excel

VBA stands for Visual Basic for Applications. The examples were developed and tested in various Excel versions between Excel 2007 and Excel 2016, but are saved in Excel 2003 format for compatibility.

- **ReadAndDisplayMultipleValues.xls**: Reads multiple OPC Classic item values, and stores them in cells of a worksheet.
- **ReadAndDisplayValue.xls**: The simplest Excel VBA example for OPC "Classic". Reads an OPC item value and stores it in a cell of a worksheet.
- **ReadAndWriteValue.xls**: Shows how to read or write an OPC Classic item value upon press of a button.
- **SubscribeToMultipleItems.xls**: Show how to subscribe to and unsubscribe from multiple OPC Classic items, and how to continuously update the worksheet cells with their values.
- **UAClientCertificateParameters.xls**: Sets the application name for the client certificate.
- **UADataDialog.xls**: Shows how a user can interactively select an OPC-UA endpoint and a data node.
- **UAReadAndDisplayMultiple.xls**: Reads multiple OPC UA nodes, and shows the results in cells of a worksheet.
- **UAReadAndDisplayValue.xls**: The simplest Excel VBA example for OPC UA. Reads an OPC UA value and stores it in a cell of a worksheet.
- **UAReadAndWriteValue.xls**: Shows how to read or write an OPC UA value upon press of a button.
- **UASubscribeToMultiple.xls**: Shows how to subscribe to and unsubscribe from multiple OPC UA monitored items, and how to continuously update the worksheet cells with their values.
- **UASubscribeToMultiple2.xls** and **UASubscribeToMultiple3.xls**: As UASubscribeToMultiple.xls, but allows sheet editing while being subscribed, using different approaches. The UASubscribeToMultiple3.xls example also shows hoe to specify an absolute deadband.


Much, if not all, of the functionality presented by these examples is achievable without any programming using the **Excel Option (Section 12.1)**. You should only consider using VBA if the features provided by the Excel Option are not sufficient for your application.

13.1.1.10 VBScript Examples (ASP, IE, WSH)

VBScript Examples in ASP

These VBScript examples run in Internet Information Server (IIS). In order to run the examples, you need to configure an IIS virtual directory that points to the physical directory where the examples are installed.

Examples in this folder are Web server scripts: the page, including OPC data, is generated on the server. The EasyOPC component must be installed on the server side, and the OPC server(s) must be accessible locally or remotely from the computer where EasyOPC component resides.

The server must be in the Web Server (IIS) role, and in it, make sure you have the "Web Server -> Application Development -> ASP" role service.

- **ReadAndDisplayValue_VBScript.asp**: The simplest Web application. Reads and displays an OPC item value.

VBScript Examples in IE

The examples here run inside Internet Explorer (IE), i.e. directly on client. No Web server is required. The EasyOPC component must be installed on the client side, and the OPC server(s) must be accessible locally or remotely from the computer where EasyOPC component resides.

- **ReadAndDisplayValue_VBScript.htm**: Reads and displays an OPC item value.

VBScript Examples in WSH

The examples here run in Windows Script Host (WSH), e.g. from Windows Command Prompt.

It is possible to use CScript (console output) or WScript (window output) to run the examples. Some examples give longer output, and CScript is better for them as you do not have to confirm each “line” of output by pressing a button as in WScript.

- **APISimplifier.wsc**, **ReadUsingSimplifier.vbs**: Shows how to create and use a component that provides domain-specific API to OPC Data Client.
- **ReadAndDisplayValue.vbs**: Reads and displays an OPC item value.
- **DocExamples** folder: Contains all VBScript examples for OPC Classic that are given in the Reference documentation.
- **UADocExamples** folder: Contains all VBScript examples for OPC-UA that are given in the Reference documentation.

> For those who want to use the command-line interface, a "Command Prompt From Here" link is installed into the examples folder. This link allows you to switch from the Windows Explorer to a CMD window with the current directory set to the chosen place, for quick experiments and exploration.

13.1.1.11 Visual Basic Examples (VB 6.0)

The examples are for Visual Basic 6.0.

- **ReadAndDisplayValue**: The simplest form. Reads and displays an OPC item value. This is what you get if you

follow the steps described in Quick Start for OPC Data Client-COM.
- **SubscribeMultipleItems**: Subscribes to multiple OPC-DA items, and displays the incoming changes in the list box.
- **UADocExamples**: Contains all VB 6.0 examples for OPC-UA that are given in the Reference documentation.
- **UAReadAndDisplayValue**: The simplest form for OPC-UA. Reads and displays an OPC-UA node value.
- **WriteMultipleItemValues**: Writes different values to multiple OPC Data Access items at once.

13.1.1.12 Visual C++ Examples

The majority of Visual C++ examples are currently provided for Microsoft Visual Studio 2017, with a possible conversion to Microsoft Visual Studio 2019.

We also provide an example in Microsoft Visual Studio 6.0 (Visual C++ 6.0), to show the feasibility, although we do not regularly update it with newer version of OPC Data Client.

In This Topic

Examples in Visual Studio 2017+

- **MFC\EasyOPCDADemo**: This is a clone of the Demo application that ships with the OPC Data Client-COM product.

- **Win32\DumpAddressSpace**: Dumps OPC server's address space recursively to the console.
- **Win32\RandomReads**: Reads random number of random items in a random fashion. This example can serve as kind of a stress test.
- **Win32\ReadMultipleItems**: Reads multiple items, and displays the results.
- **Win32\ReadMultipleItemsXml**: Reads multiple items from an OPC XML-DA server, and displays the results.
- **Win32\SubscribeMultipleItems**: Subscribes to changes of multiple items and display the value of the item with each change.
- **Win32\UADocExamples**: Contains all C++ examples for OPC-UA that are given in the Reference documentation.
- **Win32\WriteMultipleItemValues**: Writes values into multiple items at once.

All examples #include (through **stdafx.h**) a **QuickOpc.h** file in the **SDK\include** subdirectory. This file imports all OPC Data Client type libraries and declares wrappers for OPC Data Client interfaces and classes.

> We cannot assure that the examples target one of the Windows SDK versions installed in your development computer. When opening or building the examples, you may receive a warning or error to this effect. The examples do not rely on particulars of any Windows SDK version, and you should be able to easily retarget them to the Windows SDK version of your wish. In Visual Studio, if the retargeting is not offered to you or performed automatically, right-click on the solution node in the **Solution Explorer**, select the Retarget solution command, and proceed as needed.

Examples in Visual Studio 6.0

You will find some more information to this subject in the Knowledge Base article .

- **Win32\ReadMultipleItems**: Reads multiple items, and displays the results.

All examples #include (through **stdafx.h**) a **QuickOpc.h** file in the **SDK\include** subdirectory. This file imports all OPC Data Client type libraries and declares wrappers for OPC Data Client interfaces and classes.

13.1.1.13 Visual FoxPro Examples

Note: We do not regularly update these examples with new versions. You may have to modify the code to use newer version numbers.

The examples were tested in Visual FoxPro 9.0 with Service Pack 2.

- **ReadItemValue**: Reads a value of an OPC Classic item and displays it in a message box.
- **UAReadItem**: Reads a value of an OPC UA node and displays it in a message box.

13.1.1.14 Xbase++ Examples

Note: We do not regularly update these examples with new versions. You may have to modify the code to use newer version numbers.

The examples were tested with Xbase++ 1.90 SL1.

- **UAReadValue.prg**: Reads a values of an OPC UA node displays it.
- **WriteAndReadValue.prg**: Writes a value into an OPC Classic item, reads it back and displays it.

13.1.2 .NET Framework Examples

> ⚠ This article applies only if you are targeting .NET Framework runtime. If you are targeting .NET Standard, refer to **.NET Standard Examples (Section 13.1.3)** instead.

In This Topic

The examples are currently provided for Microsoft Visual Studio 2017. They can all be automatically converted to Microsoft Visual Studio 2019.

The examples are targeting Microsoft .NET Framework 4.7 (and will also run under newer versions). In Visual Studio, it is possible to re-target the projects to newer framework versions (e.g. 4.8) by changing the appropriate setting in the properties of the project.

Visual Studio solutions with examples in Visual Basic, C#, F# and C++ are available from the **Start** menu or the **Launcher** application.

NuGet Packages

Some examples (those from the DocExamples and UADocExamples projects, in C# and VB.NET) are also distributed (with source code) in form of NuGet packages, on www.nuget.org . The packages are:

- **OpcLabs.QuickOpc.Sample.CS**: Console-based QuickOPC examples in C# (source code).
- **OpcLabs.QuickOpc.Sample.VB**: Console-based QuickOPC examples in VB.NET (source code).

The strings in **bold** are the package IDs.

Using The Example Packages

The Example Package is not a runnable project on its own. It just contains pieces of source code that you can call or copy into your own project.

In order to run the code from console-based example packages, you need to call it from your project. The examples are organized by the OPC specification, and each example is a static method that you can directly invoke. For easier exploration, there are also methods that provide the user with a menu of methods to choose from.

- For OPC Alarms&Events examples, call method: DocExamples.AlarmsAndEvents.AEExamplesMenu.Main1
- For OPC Data Access, call method: DocExamples.DataAccess.DAExamplesMenu.Main1
- For OPC Unified Architecture examples, call method: UADocExamples.UAExamplesMenu.Main1
- For OPC XML-DA examples, call method: DocExamples.DataAccess.Xml.XmlExamplesMenu.Main1

Alternatively, you can call just a single selected example, e.g. for reading an OPC UA node, call the UADocExamples._EasyUAClient.Read.Main1 method.

Example Archives

Most examples for .NET Framework can also be downloaded directly from the Web, without having to use the product installer, in form of archive (.ZIP) files. They are:

- C++ Examples for .NET Framework (ZIP file)
- C# Examples for .NET Framework (ZIP file)
- PowerShell Examples for .NET Framework (ZIP file)
- VB.NET Examples for .NET Framework (ZIP file)

Note: F# examples are only available for .NET Standard - see **.NET Standard Examples (Section 13.1.3)**. You can, however, use the same F# source files in .NET Framework, and recreate the example project in .NET Framework, if you need.

Categories

Following categories of .NET examples are available:

- **Examples for OPC "Classic" (OPC-DA, OPC XML-DA and OPC-A&E) (Section 13.1.2.1)**
- **Examples for OPC Unified Architecture (OPC-UA) (Section 13.1.2.2)**
- **Integration Examples (Section 13.1.2.3)**
- **Reactive Programming Examples (Section 13.1.2.4)**
- **LINQPad Examples (Section 13.1.2.5)**

13.1.2.1 Examples for OPC "Classic" (OPC-DA, OPC XML-DA and OPC-A&E)

Console Examples

In This Topic

	Visual C#	Visual Basic	Visual F#	Visual C++
ConsoleApplication1 (Section 13.1.4.1.1): The simplest console application. Reads and displays an OPC item value in a console.			✔	✔
ConsoleLiveMapping (Section 13.1.4.1.2): Creates an object structure for a boiler, describes its mapping into OPC Data Access server using attributes, and then performs the live mapping. Boiler data is then read, written and/or subscribed to using plain .NET object access.	✔	✔		
DocExamples: A collection of OPC "Classic" and OPC XML-DA console-based examples that illustrate the use of individual objects in the product, and their members. These are the same examples that appear in reference and conceptual documentation, with an extra control routine that allows the use to choose an example to be performed.	✔	✔		
LogAsStringToSql (Section 13.1.4.1.3): Logs OPC Data Access item changes into an SQL database, using a subscription.Values of all data types are stored in a single NVARCHAR column.	✔	✔		
LogAsUnionToSql (Section 13.1.4.1.4): Logs OPC Data Access item changes into an SQL database, using a subscription. Values of all data types are stored in separate columns.	✔	✔		
LogToSqlEnhanced (Section 13.1.4.1.5): Logs OPC Data Access item changes into an SQL database, using a subscription. Item values and qualities are stored in their respective columns. Notifications with the same timestamp are merged into a single row.	✔	✔		
SubscribeFromXml (Section 13.1.4.1.11): Loads list of OPC items from an XML file and subscribes to them.	✔	✔		
WcfClient1 (Section 13.1.4.1.14): Using a Web service provided by the WcfService1 project (under Web folder), gets and displays a value of an OPC item.	✔	✔		
XmlEventLogger (Section 13.1.4.1.15): Logs OPC Alarms and Events notifications into an XML file.	✔	✔		
XmlLogger (Section 13.1.4.1.16): Logs OPC Data Access item changes into an XML file.	✔	✔		

Web Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
AutoRefreshWeb (Section 13.1.4.2.1): Web application with a screen that refreshes itself periodically.	✔	✔		
BrowseBranchesWeb (Section 13.1.4.2.2): Browses the branches in the OPC server (ASP.NET Web application).	✔	✔		
BrowseServersWeb (Section 13.1.4.2.3): Browses the available OPC servers (ASP.NET Web application).	✔	✔		
DataGridWebApplication (Section 13.1.4.2.4): Demonstrates how easily can WebControls.GridView be populated with data read from OPC Data Access server.	✔	✔		
WcfService1 (Section 13.1.4.2.6): A simple Web service using WCF technology. Provides a GetData method to read a value of an OPC item. Use WcfClient1 project (under Console folder) to test this Web service.	✔	✔		
WebApplication1 (Section 13.1.4.2.7): The simplest ASP.NET Web application for OPC "Classic". Reads and displays an OPC item value.	✔	✔		
WebService1 (Section 13.1.4.2.8): A simple Web service using ASMX technology. Provides "Hello World" method to read a value of an OPC item.	✔	✔		

Windows Forms Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
EasyOpcNetDemo (Section 13.1.4.3.1): This is a source of the Demo application for OPC "Classic" that ships with the OPC Data Client.NET product. The application shows most product functions, including the browsing forms, OPC property access, and event-based subscriptions.	✔	✔		
EasyOpcNetDemoXml (Section 13.1.4.3.2): This is a source of the Demo application for OPC "Classic" (OPC XML-enabled) that ships with the OPC Data Client.NET product. The application shows most product functions, including the browsing forms, OPC property access, and event-based subscriptions. The defaults are pre-filled for OPC XML-DA demo server, but the application is written in such a way that it can handle COM servers as well.	✔			
HmiScreen (Section 13.1.4.3.4): Windows Forms application that shows how to use implement an HMI screen by storing OPC Item IDs in the Tag property of screen controls, and animate the controls by subscribing to all items at once. Also shows a possibility how to write to an OPC item form the screen.	✔	✔		
ListView1: Shows how (Windows Forms) ListView items can be populated with OPC data, either using explicit Read, or with a subscription.		✔		
LiveBindingDemo: Shows live binding of OPC Data Access information (from simulation	✔	✔		

	Visual C#	Visual Basic	Visual F#	Visual C++
OPC server) to standard Windows Forms controls (Microsoft). All binding to OPC data is achieved with no manual coding, only by configuring the components in Visual Studio. 				
LiveBindingDemo2: Shows advanced live binding features. Among others, it demonstrates: · Binding kinds: binding to local vs. UTC timestamp. · Conversions: Use of LinearConverter, even bi-directionally. · Animations: Moving a control around the form, depending on an OPC tag value. · Cumulative: Adding incoming values to ListBox or ListView.	✔			
OvenControl (Section 13.1.4.3.5): Monitors sensors in an industrial oven, indicates level alarms by changing colors, allows the user to change a setpoint, and logs the values into a CSV file.	✔	✔		
QualityStrings (Section 13.1.4.3.6): Shows how numerical OPC quality codes are converted to displayable strings (Windows Forms application).	✔	✔		
SimpleLogToSql (Section 13.1.4.1.8): Logs OPC Data Access item changes into an SQL database, using a subscription.Values of all data types are stored in a single SQL_VARIANT column.	✔	✔		
SubscribeToMany (Section 13.1.4.3.7): Demonstrates and measures performance with large number of subscribed OPC items.	✔	✔		
ValueToMessageBox (Section 13.1.4.3.8): Very simple Windows Forms application that reads and displays an OPC item value in a message box after the user clicks on a button.	✔	✔		
WindowsFormsApplication1 (Section 13.1.4.3.9): The simplest Windows Forms application. Reads and displays an OPC item value on a form. This is what you get if you follow the steps described in Quick Start for OPC Data Client.NET.	✔	✔		

Windows Service Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
WindowsService1 (Section 13.1.4.4.1): A Windows Service that subscribes to items from the simulation server, and logs their changes into a file.	✔	✔		

WPF Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
OpcDAQualityDecoder (Section 13.1.4.5.1): A simple WPF application that decodes the cryptic OPC quality numbers into the separate fields and their symbolic representation.	✔			
WpfApplication1 (Section 13.1.4.5.3): The simplest WPF application for OPC "Classic". Reads and displays an OPC item value.	✔			

13.1.2.2 Examples for OPC Unified Architecture (OPC-UA)

Console Examples

In This Topic

	Visual C#	Visual Basic	Visual F#	Visual C++
UAConsoleApplication1: The simplest console application. Reads and displays an OPC-UA node value in a console.			✔	
UADocExamples: A collection of OPC-UA console-based examples that illustrate the use of individual objects in the product, and their members. These are the same examples that appear in reference and conceptual documentation, with an extra control routine	✔	✔	✔	

	Visual C#	Visual Basic	Visual F#	Visual C++
that allows the use to choose an example to be performed. Note: The amount of available example material may differ between the languages.				
UAConsoleLiveMapping (Section 13.1.4.1.12): Creates an object structure for a boiler, describes its mapping into OPC Unified Architecture server using attributes, and then performs the live mapping. Boiler data is then read, written and/or subscribed to using plain .NET object access.	✔	✔		
UASimpleLogToSql (Section 13.1.4.1.13): Logs OPC Unified Architecture data changes into an SQL database, using a subscription. Values of all data types are stored in a single SQL_VARIANT column.	✔			

Web Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
UAWebApplication1 (Section 13.1.4.2.5): The simplest ASP.NET Web application for OPC-UA. Reads and displays a value of a node in an OPC-UA server.	✔	✔		

Windows Forms Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
EasyOpcUADemo (Section 13.1.4.3.3): This is a source of the Demo application for OPC-UA that ships with the OPC Data Client-UA product. The application shows the most important product functions, including event-based subscriptions.	✔	✔		
UALiveBindingDemo: Shows live binding of OPC Unified Architecture information (from sample OPC-UA server) to standard Windows Forms controls (Microsoft). All binding to OPC-UA data is achieved with no manual coding, only by configuring the components in Visual Studio.	✔	✔		

	Visual C#	Visual Basic	Visual F#	Visual C++
				

Windows Service Examples

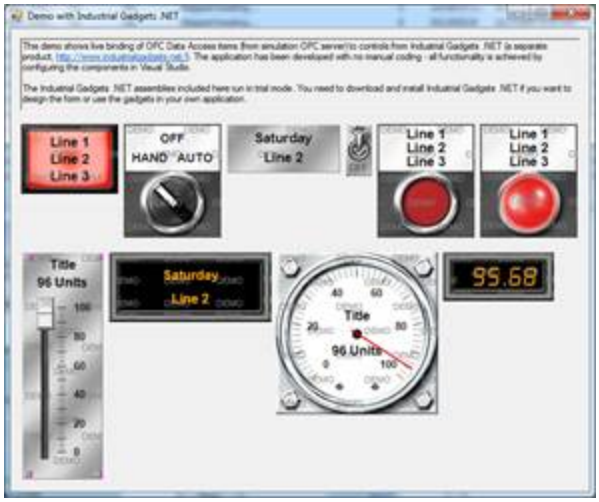
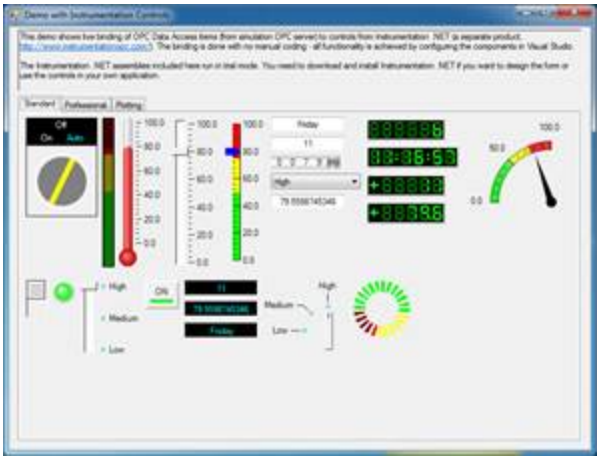
	Visual C#	Visual Basic	Visual F#	Visual C++

WPF Examples

	Visual C#	Visual Basic	Visual F#	Visual C++
UAWpfScreen (Section 13.1.4.5.2): Shows how to update WPF controls with dynamic OPC-UA data.	✔			
WpfLiveBindingDemo: Shows live binding of OPC Data Access information (from simulation OPC server) to standard Windows Presentation Foundation (WPF) controls (Microsoft). All binding to OPC data is achieved with no manual coding, only by configuring the components in Visual Studio.	✔			

13.1.2.3 Integration Examples

These examples show how OPC Data Client can be integrated with products from other parties.

	Visual C#	Visual Basic	Visual F#	Visual C++
IndustrialGadgetsDemo: Shows live binding of OPC Data Access items (from simulation OPC server) to controls from Industrial Gadgets .NET (a separate product, http://www.industrialgadgets.net/). The application has been developed with no manual coding - all functionality is achieved by configuring the components in Visual Studio. 	✔	✔		
InstrumentationControlsDemo: Shows live binding of OPC Data Access items (from simulation OPC server) to controls from Instrumentation .NET (a separate product, http://www.instrumentationopc.com/). The binding is done with no manual coding - all functionality is achieved by configuring the components in Visual Studio. 	✔	✔		
SymbolFactoryDemo: Shows live binding of OPC Data Access items (from simulation OPC server) to controls from Symbol Factory .NET (a separate product, http://www.symbolfactory.net/). All controls are bound to a single source item. The application has been developed with no manual coding - all functionality is achieved by configuring the components in Visual Studio.	✔	✔		

	Visual C#	Visual Basic	Visual F#	Visual C++
				

13.1.2.4 Reactive Programming Examples

Examples for reactive programming model need an installation of Microsoft Reactive Extensions for proper building. These extensions are now provided by Microsoft in the form of NuGet packages.

The examples only contain references to the additional packages needed, and not the packages themselves. With the help of automatic NuGet Package Restore feature (http://docs.nuget.org/docs/reference/package-restore), when enabled, the missing packages will automatically be downloaded and installed when you first build the project that references them.

In Visual Studio, this works “out of the box” (with default settings). If building the project in Visual Studio fails due to missing NuGet packages, you may have the automatic NuGet package restore turned off. Check your Visual Studio settings.

	Visual C#	Visual Basic	Visual F#	Visual C++
ReactiveDocExamples: A collection of console-based examples for reactive programming model that illustrate the use of individual objects in the product, and their members. These are the same examples that appear in reference documentation, with an extra control routine that allows the use to choose an example to be performed.	✔			

13.1.2.5 LINQPad Examples

We have created and tested LINQPad examples with LINQPad 5. They might work under LINQPad 4 as well.

You will find LINQPad examples under the CSharp, FSharp and VBNET directories in the examples, always under the LINQPad5 subdirectory. They are available in form of LINQPad sample library (.zip file). This allows you conveniently see the examples organized in a hierarchical structure, and select them quickly. In order to use the sample library from LINQPad, switch to the “Samples” tab in the lower left part of LINQPad, click "Download/import more samples...” in the tree view, and then click the “Browse...” link in the bottom left corner, and navigate to the .zip file from OPC Data Client.

In addition, the same LINQPad examples are included with the NuGet packages of OPC Data Client. This means that whenever you reference one of the OPC Data Client NuGet packages in LINQPad, LINQPad will automatically add the OPC

Data Client examples under its “Samples” tab.

13.1.3 .NET Standard Examples

The examples are currently created with Microsoft Visual Studio 2017, and can also be used with .NET Core CLI Tools, JetBrain Rider, and Visual Studio Code. They are likely to work with other development tools as well, but we have not tested it.

On Windows, if you have used the **Setup Program (Section 3.1.1)** to install OPC Data Client, Visual Studio solutions with examples in C# and F# are available from the **Start** menu or the **Launcher** application. On other platforms (when you are simply installing the OPC Data Client **NuGet Packages (Section 3.1.2)**), you are advised to download the example archives from the Web and unpack them on your computer. This is described further below.

In This Topic

Example Archives

Examples for .NET Standard can be downloaded directly from the Web, in form of archive (.TGZ or .ZIP) files. This is the primary distribution method, which can be used without running any installer, on all operating systems. The archive files available are:

- C# examples for .NET Core:
  - TGZ file: http://cdn.opclabs.com/files/downloads/QuickOpc/2020.2/Examples-NETCore-CSharp.tgz, and
  - ZIP file: http://cdn.opclabs.com/files/downloads/QuickOpc/2020.2/Examples-NETCore-CSharp.zip
- F# examples for .NET Core:
  - TGZ file: http://cdn.opclabs.com/files/downloads/QuickOpc/2020.2/Examples-NETCore-FSharp.tgz, and
  - ZIP file: http://cdn.opclabs.com/files/downloads/QuickOpc/2020.2/Examples-NETCore-FSharp.zip

⚠ The examples are delivered in source form only. If you want to run them, having installed the proper .NET runtime is not enough - you also need the corresponding SDK. For .NET Core: Linux instructions, Windows instructions - make sure you install ".NET Core SDK" and not just "Runtime".

Retrieving the Example Archives

In order to use the example archives, first download them to your system, and extract the files from the archive. With a GUI on Windows, you can download the files using a Web browser, and uncompress the .ZIP with built-in Windows Explorer (shell) commands. You may have similar features on your Linux, depending on what distribution you chose. If you don't, following commands can be used in the terminal to download and extract the archives (the URL and file names are just examples - replace them with the right ones if you need a different set of examples):

```
wget "http://cdn.opclabs.com/files/downloads/QuickOpc/2020.2/Examples-NETCore-CSharp.tgz"
tar -xvzf Examples-NETCore-CSharp.tgz
```

On systems where **wget** is not available, you can use `curl -O` instead.

Working with the Examples

You can now open the solutions or projects from your IDE, view and modify their code, or run them.

If you just want to run the examples from the command line without inspecting their source code, first change your current directory to the project type of your choice, e.g.:

cd Console

Then, build and launch the chosen project, e.g.:

dotnet run --project UADocExamples/MultiTargetUADocExamples.csproj

or

dotnet run --project DocExamples/MultiTargetDocExamples.csproj

⚠ All OPC Data Client project files for .NET Standard start with prefix **"MultiTarget"**. This is because there is also a second project over the same set of source files for .NET Framework, and a separate project file is needed for .NET Standard due to differences in the project format. Make sure you do not confuse the two projects. For example, in .NET Standard, use **"MultiTargetUADocExamples.csproj"**, and not **"UADocExamples.csproj"**.

Example NuGet Package

Some examples (those from the DocExamples and UADocExamples projects, in C#) are also distributed (with source code) in form of a NuGet package, on www.nuget.org . The package is:

- **OpcLabs.QuickOpc.Sample.CS**: Console-based QuickOPC examples in C# (source code).

The string in **bold** is the package ID.

The example NuGet package also contains code for OPC "Classic" specifications. This code is for .NET Framework only, and the only parts of it that will work under .NET Standard are those for the OPC XML-DA specification. You can dimply delete the folders that are not of interest to you from your project.

Using the Example NuGet Packages

The Example Package is not a runnable project on its own. It just contains pieces of source code that you can call or copy into your own project.

In order to run the code from console-based example packages, you need to call it from your project. The examples are organized by the OPC specification, and each example is a static method that you can directly invoke. For easier exploration, there are also methods that provide the user with a menu of methods to choose from.

- For OPC Unified Architecture examples, call method: UADocExamples.UAExamplesMenu.Main1
- For OPC XML-DA examples, call method: DocExamples.DataAccess.Xml.XmlExamplesMenu.Main1

Alternatively, you can call just a single selected example, e.g. for reading an OPC UA node, call the UADocExamples._EasyUAClient.Read.Main1 method.

Examples by Project Type

Console Application

	Visual C#	Visual F#
MultiTargetDocExamples: A collection of OPC XML-DA console-based examples that illustrate the use of individual objects in the product, and their members. These are the same examples that appear in reference and conceptual documentation, with an extra control routine that allows the use to choose an example to be performed. Note: The amount of available example material may differ between the languages.	✔	
MultiTargetUADocExamples: A collection of OPC-UA console-based examples that illustrate the use of individual objects in the product, and their members. These are the same examples that appear in reference and conceptual documentation, with an extra control routine that allows the use to choose an example to be performed. Note: The amount of available example material may differ between the languages.	✔	✔

13.1.4 Copies of Selected Installed Examples

This documentation section contains copies of selected examples that are installed with the product.

- Installed Examples - Console
  - **Installed Examples - Console - ConsoleApplication1 (Section 13.1.4.1.1)**
  - **Installed Examples - Console - ConsoleLiveMapping (Section 13.1.4.1.2)**
  - **Installed Examples - Console - LogAsStringToSql (Section 13.1.4.1.3)**
  - **Installed Examples - Console - LogAsUnionToSql (Section 13.1.4.1.4)**
  - **Installed Examples - Console - LogToSqlEnhanced (Section 13.1.4.1.5)**
  - **Installed Examples - Console - SimpleAEStreamInsightApplication (Section 13.1.4.1.6)**
  - **Installed Examples - Console - SimpleDAStreamInsightApplication (Section 13.1.4.1.7)**
  - **Installed Examples - Console - SimpleLogToSql (Section 13.1.4.1.8)**
  - **Installed Examples - Console - SimpleTrillApplication (Section 13.1.4.1.9)**
  - **Installed Examples - Console - SimpleUAStreamInsightApplication (Section 13.1.4.1.10)**
  - **Installed Examples - Console - SubscribeFromXml (Section 13.1.4.1.11)**
  - **Installed Examples - Console - UAConsoleLiveMapping (Section 13.1.4.1.12)**
  - **Installed Examples - Console - UASimpleLogToSql (Section 13.1.4.1.13)**
  - **Installed Examples - Console - WcfClient1 (Section 13.1.4.1.14)**
  - **Installed Examples - Console - XmlEventLogger (Section 13.1.4.1.15)**
  - **Installed Examples - Console - XmlLogger (Section 13.1.4.1.16)**
- Installed Examples - Web
  - **Installed Examples - Web - AutoRefreshWeb (Section 13.1.4.2.1)**
  - **Installed Examples - Web - BrowseBranchesWeb (Section 13.1.4.2.2)**
  - **Installed Examples - Web - BrowseServersWeb (Section 13.1.4.2.3)**
  - **Installed Examples - Web - DataGridWebApplication (Section 13.1.4.2.4)**
  - **Installed Examples - Web - UAWebApplication1 (Section 13.1.4.2.5)**
  - **Installed Examples - Web - WcfService1 (Section 13.1.4.2.6)**
  - **Installed Examples - Web - WebApplication1 (Section 13.1.4.2.7)**
  - **Installed Examples - Web - WebService1 (Section 13.1.4.2.8)**

- Installed Examples - WindowsForms
  - **Installed Examples - WindowsForms - EasyOpcNetDemo (Section 13.1.4.3.1)**
  - **Installed Examples - WindowsForms - EasyOpcNetDemoXml (Section 13.1.4.3.2)**
  - **Installed Examples - WindowsForms - EasyOpcUADemo (Section 13.1.4.3.3)**
  - **Installed Examples - WindowsForms - HmiScreen (Section 13.1.4.3.4)**
  - **Installed Examples - WindowsForms - OvenControl (Section 13.1.4.3.5)**
  - **Installed Examples - WindowsForms - QualityStrings (Section 13.1.4.3.6)**
  - **Installed Examples - WindowsForms - SubscribeToMany (Section 13.1.4.3.7)**
  - **Installed Examples - WindowsForms - ValueToMessageBox (Section 13.1.4.3.8)**
  - **Installed Examples - WindowsForms - WindowsFormsApplication1 (Section 13.1.4.3.9)**
- Installed Examples - WindowsService
  - **Installed Examples - WindowsService - WindowsService1 (Section 13.1.4.4.1)**
- Installed Examples - WPF
  - **Installed Examples - WPF - OpcDaQualityDecoder (Section 13.1.4.5.1)**
  - **Installed Examples - WPF - UAWpfScreen (Section 13.1.4.5.2)**
  - **Installed Examples - WPF - WpfApplication1 (Section 13.1.4.5.3)**

13.1.4.1 Installed Examples - Console

13.1.4.1.1 Installed Examples - Console - ConsoleApplication1

The simplest console application. Reads and displays an OPC item value in a console.

The main program:

C++/CLI

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
// ConsoleApplication1.cpp : main project file.

#include "stdafx.h"

using namespace OpcLabs::EasyOpc::DataAccess;
using namespace System;

int main(array<System::String ^> ^args)
{
    EasyDAClient^ client = gcnew EasyDAClient();
    Console::WriteLine("Reading item...");
    // ReadItemValue is an extension method on the IEasyDAClient interface; need to
call it explicitly in C++/CLR.
    Console::WriteLine(IEasyDAClientExtension::ReadItemValue(client, "",
"OPCLabs.KitServer.2", "Demo.Ramp"));
    Console::WriteLine("Press Enter to continue...");
    Console::ReadLine();
    return 0;
}
```

13.1.4.1.2 Installed Examples - Console - ConsoleLiveMapping

Creates an object structure for a boiler, describes its mapping into OPC Data Access server using attributes, and then performs the live mapping. Boiler data is then read, written and/or subscribed to using plain .NET object access.

The main program:

C#

```
// ConsoleLiveMapping: Creates an object structure for a boiler, describes its mapping
into OPC Data Access server using
// attributes, and then performs the live mapping. Boiler data is then read, written
and/or subscribed to using plain .NET
// object access.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;
using OpcLabs.EasyOpc.DataAccess.LiveMapping.Extensions;

namespace ConsoleLiveMapping
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine();
            Console.WriteLine("Mapping our data structures to OPC...");
            var mapper = new DAClientMapper();
            var boiler1 = new Boiler();
            mapper.Map(boiler1, new DAMappingContext
                {
                    ServerDescriptor = "OPCLabs.KitServer.2",   // local OPC server
                    // The NodeDescriptor below determines where in the OPC address
space we want to map our data to.
                    NodeDescriptor = new DANodeDescriptor { BrowsePath =
"/Boilers/Boiler #1"},
                    GroupParameters = 1000,  // requested update rate (for
subscriptions)
                });

            Console.WriteLine();
            Console.WriteLine("Reading all data of the boiler...");
            mapper.Read();
            Console.WriteLine("Drum level is: {0}",
boiler1.Drum.LevelIndicator.Output);

            Console.WriteLine();
            Console.WriteLine("Writing new setpoint value...");
            boiler1.LevelController.SetPoint = 50.0;
            Debug.Assert(boiler1.LevelController != null);
```

```
            mapper.WriteTarget(boiler1.LevelController, /*recurse:*/false);

            Console.WriteLine();
            Console.WriteLine("Subscribing to boiler data changes...");
            mapper.Subscribe(/*active:*/true);

            Thread.Sleep(30 * 1000);

            Console.WriteLine();
            Console.WriteLine("Unsubscribing from boiler data changes...");
            mapper.Subscribe(/*active:*/false);

            Console.WriteLine();
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
    }
}
```

VB.NET

```
' ConsoleLiveMapping: Creates an object structure for a boiler, describes its mapping
into OPC OPC Data Access server using
' attributes, and then performs the live mapping. Boiler data is then read, written
and/or subscribed to using plain .NET
' object access.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping.Extensions

' ReSharper disable CheckNamespace
Namespace ConsoleLiveMapping
    ' ReSharper restore CheckNamespace

    Friend Class Program
        Shared Sub Main()
            Console.WriteLine()
            Console.WriteLine("Mapping our data structures to OPC...")
            Dim mapper = New DAClientMapper()
            Dim boiler1 = New Boiler()
            mapper.Map(boiler1, New DAMappingContext With {.ServerDescriptor =
"OPCLabs.KitServer.2", .NodeDescriptor = New DANodeDescriptor With {.BrowsePath =
"/Boilers/Boiler #1"}, .GroupParameters = 1000}) ' requested update rate (for
subscriptions) -  local OPC server
            ' The NodeDescriptor below determines where in the OPC address space we
want to map our data to.

            Console.WriteLine()
            Console.WriteLine("Reading all data of the boiler...")
            mapper.Read()
            Console.WriteLine("Drum level is: {0}", boiler1.Drum.LevelIndicator.Output)

            Console.WriteLine()
            Console.WriteLine("Writing new setpoint value...")
            boiler1.LevelController.SetPoint = 50.0
```

```
            Debug.Assert(boiler1.LevelController IsNot Nothing)
            mapper.WriteTarget(boiler1.LevelController, False) 'recurse:False

            Console.WriteLine()
            Console.WriteLine("Subscribing to boiler data changes...")
            mapper.Subscribe(True) 'active:True

            Thread.Sleep(30 * 1000)

            Console.WriteLine()
            Console.WriteLine("Unsubscribing from boiler data changes...")
            mapper.Subscribe(False) 'active:True

            Console.WriteLine()
            Console.WriteLine("Press Enter to continue...")
            Console.ReadLine()
        End Sub
    End Class
End Namespace
```

The Boiler class:

C#

```
using System;
using OpcLabs.BaseLib.LiveMapping;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;

namespace ConsoleLiveMapping
{
    // The Boiler and its constituents are described in our application domain terms,
the way we want to work with them.
    // Attributes are used to describe the correspondence between our types and
members, and OPC nodes.

    // This is how the boiler looks in OPC address space:
    //  - Boiler #1
    //      - CC1001                    (CustomController)
    //          - ControlOut
    //          - Description
    //          - Input1
    //          - Input2
    //          - Input3
    //      - Drum1001                  (BoilerDrum)
    //          - LIX001                (LevelIndicator)
    //              - Output
    //      - FC1001                    (FlowController)
    //          - ControlOut
    //          - Measurement
    //          - SetPoint
    //      - LC1001                    (LevelController)
    //          - ControlOut
    //          - Measurement
    //          - SetPoint
```

```
    //      - Pipe1001                  (BoilerInputPipe)
    //          - FTX001                (FlowTransmitter)
    //              - Output
    //      - Pipe1002                  (BoilerOutputPipe)
    //          - FTX002                (FlowTransmitter)
    //              - Output

    [DAType]
    class Boiler
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [DANode(BrowsePath = "Pipe1001")]
        public BoilerInputPipe InputPipe = new BoilerInputPipe();

        [DANode(BrowsePath = "Drum1001")]
        public BoilerDrum Drum = new BoilerDrum();

        [DANode(BrowsePath = "Pipe1002")]
        public BoilerOutputPipe OutputPipe = new BoilerOutputPipe();

        [DANode(BrowsePath = "FC1001")]
        public FlowController FlowController = new FlowController();

        [DANode(BrowsePath = "LC1001")]
        public LevelController LevelController = new LevelController();

        [DANode(BrowsePath = "CC1001")]
        public CustomController CustomController = new CustomController();
    }

    [DAType]
    class BoilerInputPipe
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [DANode(BrowsePath = "FTX001")]
        public FlowTransmitter FlowTransmitter1 = new FlowTransmitter();

        [DANode(BrowsePath = "ValveX001")]
        public Valve Valve = new Valve();
    }

    [DAType]
    class BoilerDrum
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [DANode(BrowsePath = "LIX001")]
        public LevelIndicator LevelIndicator = new LevelIndicator();
    }

    [DAType]
    class BoilerOutputPipe
```

```
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [DANode(BrowsePath = "FTX002")]
        public FlowTransmitter FlowTransmitter2 = new FlowTransmitter();
    }

    [DAType]
    class FlowController : GenericController
    {
    }

    [DAType]
    class LevelController : GenericController
    {
    }

    [DAType]
    class CustomController
    {
        [DANode, DAItem]
        public double Input1 { get; set; }

        [DANode, DAItem]
        public double Input2 { get; set; }

        [DANode, DAItem]
        public double Input3 { get; set; }

        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double ControlOut { get; set; }

        [DANode, DAItem]
        public string Description { get; set; }
    }

    [DAType]
    class FlowTransmitter : GenericSensor
    {
    }

    [DAType]
    class Valve : GenericActuator
    {
    }

    [DAType]
    class LevelIndicator : GenericSensor
    {
    }

    [DAType]
    class GenericController
    {
        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
```

```
OPC writing
        public double Measurement { get; set; }

        [DANode, DAItem]
        public double SetPoint { get; set; }

        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double ControlOut { get; set; }
    }

    [DAType]
    class GenericSensor
    {
        // Meta-members are filled in by information collected during mapping, and
allow access to it later from your code.
        // Alternatively, you can derive your class from DAMappedNode, which will bring
in many meta-members automatically.
        [MetaMember("NodeDescriptor")]
        public DANodeDescriptor NodeDescriptor { get; set; }

        [DANode, DAItem(Operations = DAItemMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double Output
        {
            get { return _output; }
            set
            {
                _output = value;
                Console.WriteLine("Sensor \"{0}\" output is now {1}.", NodeDescriptor,
value);
            }
        }

        private double _output;
    }

    [DAType]
    class GenericActuator
    {
        [DANode, DAItem]
        public double Input { get; set; }
    }
}
```

VB.NET

```
Imports OpcLabs.BaseLib.LiveMapping
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping

' ReSharper disable CheckNamespace
Namespace ConsoleLiveMapping
    ' ReSharper restore CheckNamespace

    ' The Boiler and its constituents are described in our application domain terms,
the way we want to work with them.
```

```
    ' Attributes are used to describe the correspondence between our types and members,
and OPC nodes.

    ' This is how the boiler looks in OPC address space:
    '  - Boiler #1
    '      - CC1001                    (CustomController)
    '          - ControlOut
    '          - Description
    '          - Input1
    '          - Input2
    '          - Input3
    '      - Drum1001                  (BoilerDrum)
    '          - LIX001                (LevelIndicator)
    '              - Output
    '      - FC1001                    (FlowController)
    '          - ControlOut
    '          - Measurement
    '          - SetPoint
    '      - LC1001                    (LevelController)
    '          - ControlOut
    '          - Measurement
    '          - SetPoint
    '      - Pipe1001                  (BoilerInputPipe)
    '          - FTX001                (FlowTransmitter)
    '              - Output
    '      - Pipe1002                  (BoilerOutputPipe)
    '          - FTX002                (FlowTransmitter)
    '              - Output

    <DAType()> _
    Friend Class Boiler
        ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        <DANode(BrowsePath:="Pipe1001")> _
        Public InputPipe As New BoilerInputPipe()

        <DANode(BrowsePath:="Drum1001")> _
        Public Drum As New BoilerDrum()

        <DANode(BrowsePath:="Pipe1002")> _
        Public OutputPipe As New BoilerOutputPipe()

        <DANode(BrowsePath:="FC1001")> _
        Public FlowController As New FlowController()

        <DANode(BrowsePath:="LC1001")> _
        Public LevelController As New LevelController()

        <DANode(BrowsePath:="CC1001")> _
        Public CustomController As New CustomController()
    End Class

    <DAType()> _
    Friend Class BoilerInputPipe
        ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.
```

```
        <DANode(BrowsePath:="FTX001")> _
        Public FlowTransmitter1 As New FlowTransmitter()

        <DANode(BrowsePath:="ValveX001")> _
        Public Valve As New Valve()
    End Class

    <DAType()> _
    Friend Class BoilerDrum
        ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        <DANode(BrowsePath:="LIX001")> _
        Public LevelIndicator As New LevelIndicator()
    End Class

    <DAType()> _
    Friend Class BoilerOutputPipe
        ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        <DANode(BrowsePath:="FTX002")> _
        Public FlowTransmitter2 As New FlowTransmitter()
    End Class

    <DAType()> _
    Friend Class FlowController
        Inherits GenericController

    End Class

    <DAType()> _
    Friend Class LevelController
        Inherits GenericController

    End Class

    <DAType()> _
    Friend Class CustomController
        <DANode(), DAItem()> _
        Public Property Input1 As Double

        <DANode(), DAItem()> _
        Public Property Input2 As Double

        <DANode(), DAItem()> _
        Public Property Input3 As Double

        <DANode(), DAItem(Operations:=DAItemMappingOperations.ReadAndSubscribe)> _
        Public Property ControlOut As Double

        <DANode(), DAItem()> _
        Public Property Description As String
    End Class

    <DAType()> _
```

```
    Friend Class FlowTransmitter
        Inherits GenericSensor

    End Class

    <DAType()> _
    Friend Class Valve
        Inherits GenericActuator

    End Class

    <DAType()> _
    Friend Class LevelIndicator
        Inherits GenericSensor

    End Class

    <DAType()> _
    Friend Class GenericController
        <DANode(), DAItem(Operations:=DAItemMappingOperations.ReadAndSubscribe)> _
        Public Property Measurement As Double

        <DANode(), DAItem()> _
        Public Property SetPoint As Double

        <DANode(), DAItem(Operations:=DAItemMappingOperations.ReadAndSubscribe)> _
        Public Property ControlOut As Double
    End Class

    <DAType()> _
    Friend Class GenericSensor
        ' Meta-members are filled in by information collected during mapping, and allow
access to it later from your code.
        ' Alternatively, you can derive your class from DAMappedNode, which will bring
in many meta-members automatically.
        <MetaMember("NodeDescriptor")> _
        Public Property NodeDescriptor As DANodeDescriptor

        <DANode(), DAItem(Operations:=DAItemMappingOperations.ReadAndSubscribe)> _
        Public Property Output() As Double ' no OPC writing
            Get
                Return _output
            End Get
            Set(ByVal value As Double)
                _output = value
                Console.WriteLine("Sensor ""{0}"" output is now {1}.", NodeDescriptor,
value)
            End Set
        End Property

        Private _output As Double
    End Class

    <DAType()> _
    Friend Class GenericActuator
        <DANode(), DAItem()> _
        Public Property Input As Double
```

```
    End Class
End Namespace
```

13.1.4.1.3 Installed Examples - Console - LogAsStringToSql

Logs OPC Data Access item changes into an SQL database, using a subscription. Values of all data types are stored in a single NVARCHAR column.

The main program:

C#

```
// LogAsStringToSql: Logs OPC Data Access item changes into an SQL database, using a
subscription. Values of all data types are
// stored in a single NVARCHAR column.

// The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
// directory. The example assumes that the database is already created.

using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace LogAsStringToSql
{
    class Program
    {
        static void Main()
        {
            const string connectionString =
                "Data Source=(local);Initial Catalog=QuickOPCExamples;Integrated
Security=true";

            Console.WriteLine("Starting up...");
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Create all necessary ADO.NET objects.
                var adapter = new SqlDataAdapter("SELECT * FROM LogAsString",
connection);
                var dataSet = new DataSet();
                adapter.FillSchema(dataSet, SchemaType.Source, "LogAsString");
                adapter.InsertCommand = new
SqlCommandBuilder(adapter).GetInsertCommand();
                DataTable table = dataSet.Tables["LogAsString"];
                Debug.Assert(table != null);
```

```
                Console.WriteLine("Logging for 30 seconds...");
                // Subscribe to an OPC item, using an anonymous method to process the
notifications.
                int[] handles = EasyDAClient.SharedInstance.SubscribeMultipleItems(
                    new[]
                        {
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Incrementing (1 s)", 100, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Ramp (10 s)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BSTR", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BOOL", 1000, null)
                        },
                    (_, eventArgs) =>
                        {
                            Debug.Assert(eventArgs != null);

                            Console.Write(".");
                            // In this example, we only log valid data. Production
logger would also log errors.
                            if (eventArgs.Vtq != null)
                            {
                                // Fill a DataRow with the OPC data, and add it to a
DataTable.
                                Debug.Assert(table.Rows != null);
                                table.Rows.Clear();

                                DataRow row = table.NewRow();
                                row["ItemID"] =
eventArgs.Arguments.ItemDescriptor.ItemId;
                                row["Value"] = eventArgs.Vtq.Value; // The DataRow will
make the conversion to a string.
                                row["Timestamp"] = (eventArgs.Vtq.Timestamp <
(DateTime) SqlDateTime.MinValue)
                                                       ? (DateTime)SqlDateTime.MinValue
                                                       : eventArgs.Vtq.Timestamp;
                                row["Quality"] = (short)eventArgs.Vtq.Quality;

                                Debug.Assert(table.Rows != null);
                                table.Rows.Add(row);

                                // Update the underlying DataSet using an insert
command.
                                adapter.Update(dataSet, "LogAsString");
                            }
                        }
                    );
                System.Threading.Thread.Sleep(30*1000);
                Console.WriteLine();

                Console.WriteLine("Shutting down...");
                EasyDAClient.SharedInstance.UnsubscribeMultipleItems(handles);
            }

            Console.WriteLine("Finished.");
```

```
        }
    }
}
```

VB.NET

```
' LogAsStringToSql: Logs OPC Data Access item changes into an SQL database, using a
subscription. Values of all data types are
' stored in a single NVARCHAR column.

' The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
' directory. The example assumes that the database is already created.
Imports System.Data.SqlClient
Imports System.Data.SqlTypes
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Friend Class Program
    Shared WithEvents _client As New EasyDAClient
    Shared _adapter As SqlDataAdapter
    Shared _dataSet As DataSet
    Shared _table As DataTable

    Shared Sub Main()
        Const connectionString As String = "Data Source=(local);Initial
Catalog=QuickOPCExamples;Integrated Security=true"

        Console.WriteLine("Starting up...")
        Using connection = New SqlConnection(connectionString)
            connection.Open()

            ' Create all necessary ADO.NET objects.
            _adapter = New SqlDataAdapter("SELECT * FROM LogAsString", connection)
            _dataSet = New DataSet()
            _adapter.FillSchema(_dataSet, SchemaType.Source, "LogAsString")
            _adapter.InsertCommand = (New
SqlCommandBuilder(_adapter)).GetInsertCommand()
            _table = _dataSet.Tables("LogAsString")
            Debug.Assert(_table IsNot Nothing)

            Console.WriteLine("Logging for 30 seconds...")
            ' Subscribe to an OPC item, using an anonymous method to process the
notifications.
            Dim [handles]() As Integer = _client.SubscribeMultipleItems(New
DAItemGroupArguments() { _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Incrementing (1 s)", 100, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Ramp (10 s)", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BSTR", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BOOL", 1000, Nothing) _
                })
            Threading.Thread.Sleep(30 * 1000)
            Console.WriteLine()
```

```
            Console.WriteLine("Shutting down...")
            _client.UnsubscribeMultipleItems([handles])
        End Using

        Console.WriteLine("Finished.")
    End Sub

    Private Shared Sub ItemChanged(ByVal sender As Object, ByVal eventArgs As
EasyDAItemChangedEventArgs) Handles _client.ItemChanged
        Debug.Assert(eventArgs IsNot Nothing)

        Console.Write(".")
        ' In this example, we only log valid data. Production logger would also log
errors.
        If eventArgs.Vtq IsNot Nothing Then
            ' Fill a DataRow with the OPC data, and add it to a DataTable.
            Debug.Assert(_table.Rows IsNot Nothing)
            _table.Rows.Clear()

            Dim row As DataRow = _table.NewRow()
            row("ItemID") = eventArgs.Arguments.ItemDescriptor.ItemId
            row("Value") = eventArgs.Vtq.Value
            row("Timestamp") = If(eventArgs.Vtq.Timestamp <
CDate(SqlDateTime.MinValue), CDate(SqlDateTime.MinValue), eventArgs.Vtq.Timestamp)
            row("Quality") = CShort(Fix(eventArgs.Vtq.Quality))

            Debug.Assert(_table.Rows IsNot Nothing)
            _table.Rows.Add(row)

            ' Update the underlying DataSet using an insert command.
            _adapter.Update(_dataSet, "LogAsString")
        End If
    End Sub
End Class
```

13.1.4.1.4 Installed Examples - Console - LogAsUnionToSql

Logs OPC Data Access item changes into an SQL database, using a subscription. Values of all data types are stored in separate columns.

The main program:

C#

```
// LogAsUnionToSql: Logs OPC Data Access item changes into an SQL database, using a
subscription. Values of different data types
// are stored in separate columns.

// The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
// directory. The example assumes that the database is already created.
```

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace LogAsUnionToSql
{
    class Program
    {
        static void Main()
        {
            const string connectionString =
                "Data Source=(local);Initial Catalog=QuickOPCExamples;Integrated
Security=true";

            Console.WriteLine("Starting up...");
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Create all necessary ADO.NET objects.
                var adapter = new SqlDataAdapter("SELECT * FROM LogAsUnion",
connection);
                var dataSet = new DataSet();
                adapter.FillSchema(dataSet, SchemaType.Source, "LogAsUnion");
                adapter.InsertCommand = new
SqlCommandBuilder(adapter).GetInsertCommand();
                DataTable table = dataSet.Tables["LogAsUnion"];
                Debug.Assert(table != null);

                Console.WriteLine("Logging for 30 seconds...");
                // Subscribe to an OPC item, using an anonymous method to process the
notifications.
                int[] handles = EasyDAClient.SharedInstance.SubscribeMultipleItems(
                    new[]
                        {
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Incrementing (1 s)", 100, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Ramp (10 s)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BSTR", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BOOL", 1000, null)
                        },
                    (_, eventArgs) =>
                        {
                            Debug.Assert(eventArgs != null);

                            Console.Write(".");
                            // In this example, we only log valid data. Production
logger would also log errors.
                            if (eventArgs.Vtq != null)
                            {
```

```
                                // Fill a DataRow with the OPC data, and add it to a
DataTable.
                                Debug.Assert(table.Rows != null);
                                table.Rows.Clear();
                                DataRow row = table.NewRow();
                                row["ItemID"] =
eventArgs.Arguments.ItemDescriptor.ItemId;

                                if (eventArgs.Vtq.Value != null)
                                {
                                    Type type = eventArgs.Vtq.Value.GetType();
                                    // Store into a corresponding column.
                                    // The DataRow will make the conversion to a
string.
                                    if (type == typeof(Int16) || (type ==
typeof(Int32)) || type == typeof(Int64))
                                        row["IntegerValue"] = eventArgs.Vtq.Value;
                                    else if (type == typeof(Single) || type ==
typeof(Double))
                                        row["FloatValue"] = eventArgs.Vtq.Value;
                                    else if (type == typeof(string))
                                        row["StringValue"] = eventArgs.Vtq.Value;
                                    else if (type == typeof(Boolean))
                                        row["BooleanValue"] = eventArgs.Vtq.Value;
                                }

                                row["Timestamp"] = (eventArgs.Vtq.Timestamp <
(DateTime) SqlDateTime.MinValue)
                                                       ? (DateTime)SqlDateTime.MinValue
                                                       : eventArgs.Vtq.Timestamp;
                                row["Quality"] = (short)eventArgs.Vtq.Quality;

                                Debug.Assert(table.Rows != null);
                                table.Rows.Add(row);

                                // Update the underlying DataSet using an insert
command.
                                adapter.Update(dataSet, "LogAsUnion");
                            }
                        }
                    );
                System.Threading.Thread.Sleep(30*1000);
                Console.WriteLine();

                Console.WriteLine("Shutting down...");
                EasyDAClient.SharedInstance.UnsubscribeMultipleItems(handles);
            }

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
' LogAsUnionToSql: Logs OPC Data Access item changes into an SQL database, using a
subscription. Values of different data types
```

```
' are stored in separate columns.

' The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
' directory. The example assumes that the database is already created.
Imports System.Data.SqlClient
Imports System.Data.SqlTypes
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Friend Class Program
    Shared WithEvents _client As New EasyDAClient
    Shared _adapter As SqlDataAdapter
    Shared _dataSet As DataSet
    Shared _table As DataTable

    Shared Sub Main()
        Const connectionString As String = "Data Source=(local);Initial
Catalog=QuickOPCExamples;Integrated Security=true"

        Console.WriteLine("Starting up...")
        Using connection = New SqlConnection(connectionString)
            connection.Open()

            ' Create all necessary ADO.NET objects.
            _adapter = New SqlDataAdapter("SELECT * FROM LogAsUnion", connection)
            _dataSet = New DataSet()
            _adapter.FillSchema(_dataSet, SchemaType.Source, "LogAsUnion")
            _adapter.InsertCommand = (New
SqlCommandBuilder(_adapter)).GetInsertCommand()
            _table = _dataSet.Tables("LogAsUnion")
            Debug.Assert(_table IsNot Nothing)

            Console.WriteLine("Logging for 30 seconds...")
            ' Subscribe to an OPC item, using an anonymous method to process the
notifications.
            Dim [handles]() As Integer = _client.SubscribeMultipleItems(New
DAItemGroupArguments() { _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Incrementing (1 s)", 100, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Ramp (10 s)", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BSTR", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BOOL", 1000, Nothing) _
                })
            Threading.Thread.Sleep(30 * 1000)
            Console.WriteLine()

            Console.WriteLine("Shutting down...")
            _client.UnsubscribeMultipleItems([handles])
        End Using

        Console.WriteLine("Finished.")
    End Sub
```

```
    Private Shared Sub ItemChanged(ByVal sender As Object, ByVal eventArgs As
EasyDAItemChangedEventArgs) Handles _client.ItemChanged
        Debug.Assert(eventArgs IsNot Nothing)
        Console.Write(".")
        ' In this example, we only log valid data. Production logger would also log
errors.
        If eventArgs.Vtq IsNot Nothing Then
            ' Fill a DataRow with the OPC data, and add it to a DataTable.
            Debug.Assert(_table.Rows IsNot Nothing)
            _table.Rows.Clear()
            Dim row As DataRow = _table.NewRow()
            row("ItemID") = eventArgs.Arguments.ItemDescriptor.ItemId

            If eventArgs.Vtq.Value IsNot Nothing Then
                ' ReSharper disable VBPossibleMistakenCallToGetType.2
                Dim type As Type = eventArgs.Vtq.Value.GetType()
                ' ReSharper restore VBPossibleMistakenCallToGetType.2
                ' Store into a corresponding column.
                ' The DataRow will make the conversion to a string.
                If type Is GetType(Int16) OrElse (type Is GetType(Int32)) OrElse type
Is GetType(Int64) Then
                    row("IntegerValue") = eventArgs.Vtq.Value
                ElseIf type Is GetType(Single) OrElse type Is GetType(Double) Then
                    row("FloatValue") = eventArgs.Vtq.Value
                ElseIf type Is GetType(String) Then
                    row("StringValue") = eventArgs.Vtq.Value
                ElseIf type Is GetType(Boolean) Then
                    row("BooleanValue") = eventArgs.Vtq.Value
                End If
            End If

            row("Timestamp") = If(eventArgs.Vtq.Timestamp <
CDate(SqlDateTime.MinValue), CDate(SqlDateTime.MinValue), eventArgs.Vtq.Timestamp)
            row("Quality") = CShort(Fix(eventArgs.Vtq.Quality))

            Debug.Assert(_table.Rows IsNot Nothing)
            _table.Rows.Add(row)

            ' Update the underlying DataSet using an insert command.
            _adapter.Update(_dataSet, "LogAsUnion")
        End If
    End Sub
End Class
```

13.1.4.1.5 Installed Examples - Console - LogToSqlEnhanced

Logs OPC Data Access item changes into an SQL database, using a subscription. Item values and qualities are stored in their respective columns. Notifications with the same timestamp are merged into a single row.

The main program:

C#

```
// LogToSqlEnhanced: Logs OPC Data Access item changes into an SQL database, using a
subscription. Item values and qualities
// are stored in their respective columns. Notifications with the same timestamp are
merged into a single row.

// The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
// directory. The example assumes that the database is already created.

using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace LogToSqlEnhanced
{
    class Program
    {
        static void Main()
        {
            const string connectionString =
                "Data Source=(local);Initial Catalog=QuickOPCExamples;Integrated
Security=true";

            Console.WriteLine("Starting up...");
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Create all necessary ADO.NET objects.
                var adapter = new SqlDataAdapter("SELECT * FROM ColumnarLog",
connection);
                var dataSet = new DataSet();
                adapter.FillSchema(dataSet, SchemaType.Source, "ColumnarLog");
                adapter.InsertCommand = new
SqlCommandBuilder(adapter).GetInsertCommand();
                adapter.UpdateCommand = new
SqlCommandBuilder(adapter).GetUpdateCommand();
                DataTable table = dataSet.Tables["ColumnarLog"];

                Console.WriteLine("Logging for 30 seconds...");
                // Subscribe to an OPC item, using an anonymous method to process the
notifications.
                int[] handles = EasyDAClient.SharedInstance.SubscribeMultipleItems(
                    new[]
                        {
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 s)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (10 s)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (10 min)", 1000, null)
```

```
                    },
                (_, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);
                    Console.Write(".");
                    // In this example, we only log valid data. Production logger
would also log errors.
                    if (eventArgs.Vtq != null)
                    {
                        // Fill a DataRow with the OPC data, and add it to a
DataTable.
                        DateTime timestamp =
                            (eventArgs.Vtq.Timestamp < (DateTime)
SqlDateTime.MinValue)
                            ? (DateTime) SqlDateTime.MinValue
                            : eventArgs.Vtq.Timestamp;

                        DataRow row =
dataSet.Tables["ColumnarLog"].Rows.Find(timestamp);
                        bool adding = (row == null);
                        if (adding)
                        {
                            row = table.NewRow();
                            row["Timestamp"] = timestamp;
                        }

                        switch (eventArgs.Arguments.ItemDescriptor.ItemId)
                        {
                            case "Trends.Ramp (1 s)":
                                row["Ramp1Value"] = eventArgs.Vtq.Value ??
DBNull.Value;
                                row["Ramp1Quality"] = (short)eventArgs.Vtq.Quality;
                                break;
                            case "Trends.Ramp (10 s)":
                                row["Ramp2Value"] = eventArgs.Vtq.Value ??
DBNull.Value;
                                row["Ramp2Quality"] = (short)eventArgs.Vtq.Quality;
                                break;
                            case "Trends.Ramp (1 min)":
                                row["Ramp3Value"] = eventArgs.Vtq.Value ??
DBNull.Value;
                                row["Ramp3Quality"] = (short)eventArgs.Vtq.Quality;
                                break;
                            case "Trends.Ramp (10 min)":
                                row["Ramp4Value"] = eventArgs.Vtq.Value ??
DBNull.Value;
                                row["Ramp4Quality"] = (short)eventArgs.Vtq.Quality;
                                break;
                        }

                        if (adding)
                            table.Rows.Add(row);

                        // Update the underlying DataSet using an insert command.
                        adapter.Update(dataSet, "ColumnarLog");

                        // IMPROVE: For long-running logs, you may have to remove
```

```
old DataTable.Rows from memory.
                        }
                    }
                    );
                System.Threading.Thread.Sleep(30 * 1000);
                Console.WriteLine();

                Console.WriteLine("Shutting down...");
                EasyDAClient.SharedInstance.UnsubscribeMultipleItems(handles);
            }

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
' LogToSqlEnhanced: Logs OPC Data Access item changes into an SQL database, using a
subscription. Item values and qualities
' are stored in their respective columns. Notifications with the same timestamp are
merged into a single row.

' The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
' directory. The example assumes that the database is already created.
Imports System.Data.SqlClient
Imports System.Data.SqlTypes
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Friend Class Program
    Shared WithEvents _client As New EasyDAClient
    Shared _adapter As SqlDataAdapter
    Shared _dataSet As DataSet
    Shared _table As DataTable

    Shared Sub Main()
        Const connectionString As String = "Data Source=(local);Initial
Catalog=QuickOPCExamples;Integrated Security=true"

        Console.WriteLine("Starting up...")
        Using connection = New SqlConnection(connectionString)
            connection.Open()

            ' Create all necessary ADO.NET objects.
            _adapter = New SqlDataAdapter("SELECT * FROM ColumnarLog", connection)
            _dataSet = New DataSet()
            _adapter.FillSchema(_dataSet, SchemaType.Source, "ColumnarLog")
            _adapter.InsertCommand = (New
SqlCommandBuilder(_adapter)).GetInsertCommand()
            _adapter.UpdateCommand = (New
SqlCommandBuilder(_adapter)).GetUpdateCommand()
            _table = _dataSet.Tables("ColumnarLog")

            Console.WriteLine("Logging for 30 seconds...")
            ' Subscribe to an OPC item, using an anonymous method to process the
```

```
notifications.
            Dim [handles]() As Integer = _client.SubscribeMultipleItems(New
DAItemGroupArguments() { _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Ramp (1
s)", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Ramp
(10 s)", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Ramp (1
min)", 1000, Nothing), _
                    New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Ramp
(10 min)", 1000, Nothing) _
                })
            Threading.Thread.Sleep(30 * 1000)
            Console.WriteLine()

            Console.WriteLine("Shutting down...")
            _client.UnsubscribeMultipleItems([handles])
        End Using

        Console.WriteLine("Finished.")
    End Sub

    Private Shared Sub ItemChanged(ByVal sender As Object, ByVal eventArgs As
EasyDAItemChangedEventArgs) Handles _client.ItemChanged
        Console.Write(".")
        ' In this example, we only log valid data. Production logger would also log
errors.
        If eventArgs.Vtq IsNot Nothing Then
            ' Fill a DataRow with the OPC data, and add it to a DataTable.
            Dim timestamp As Date = If( _
                eventArgs.Vtq.Timestamp < CDate(SqlDateTime.MinValue), _
                CDate(SqlDateTime.MinValue), _
                eventArgs.Vtq.Timestamp)

            Dim row As DataRow = _dataSet.Tables("ColumnarLog").Rows.Find(timestamp)
            Dim adding As Boolean = (row Is Nothing)
            If adding Then
                row = _table.NewRow()
                row("Timestamp") = timestamp
            End If

            Select Case eventArgs.Arguments.ItemDescriptor.ItemId
                Case "Trends.Ramp (1 s)"
                    row("Ramp1Value") = If(eventArgs.Vtq.Value, DBNull.Value)
                    row("Ramp1Quality") = CShort(Fix(eventArgs.Vtq.Quality))
                Case "Trends.Ramp (10 s)"
                    row("Ramp2Value") = If(eventArgs.Vtq.Value, DBNull.Value)
                    row("Ramp2Quality") = CShort(Fix(eventArgs.Vtq.Quality))
                Case "Trends.Ramp (1 min)"
                    row("Ramp3Value") = If(eventArgs.Vtq.Value, DBNull.Value)
                    row("Ramp3Quality") = CShort(Fix(eventArgs.Vtq.Quality))
                Case "Trends.Ramp (10 min)"
                    row("Ramp4Value") = If(eventArgs.Vtq.Value, DBNull.Value)
                    row("Ramp4Quality") = CShort(Fix(eventArgs.Vtq.Quality))
            End Select

            If adding Then
```

```
                _table.Rows.Add(row)
            End If

            ' Update the underlying DataSet using an insert command.
            _adapter.Update(_dataSet, "ColumnarLog")

            ' IMPROVE: For long-running logs, you may have to remove old DataTable.Rows
from memory.
        End If
    End Sub
End Class
```

13.1.4.1.6 Installed Examples - Console - SimpleAEStreamInsightApplication

This example shows how to create an OPC Alarms&Events event source, and query it for events with severity 20 or higher.

The main program:

C#

```
// This example shows how to create an OPC Alarms&Events event source, and query it for
events with severity 20 or higher.

using System;
using System.Diagnostics;
using System.Reactive;
using System.ServiceModel;
using Microsoft.ComplexEventProcessing;
using Microsoft.ComplexEventProcessing.Linq;
using Microsoft.ComplexEventProcessing.ManagementService;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.ComplexEventProcessing;
using OpcLabs.EasyOpc.AlarmsAndEvents.Reactive;

namespace SimpleAEStreamInsightApplication
{
    class Program
    {
        static void Main()
        {
            // Create an embedded StreamInsight server
            //using (var server = Server.Create("Default"))
            using (var server = Server.Create("Instance1"))
            {
                Debug.Assert(server != null);

                // Create a local end point for the server embedded in this program
                var managementService = server.CreateManagementService();
                Debug.Assert(managementService != null);
                var host = new ServiceHost(managementService);
                host.AddServiceEndpoint(typeof(IManagementService), new
```

```
WSHttpBinding(SecurityMode.Message),
                    "http://localhost/MyStreamInsightServer");
                host.Open();

                /* The following entities will be defined and available in the server
for other clients:
                 * serverApp
                 * serverSource
                 * serverSink
                 * serverProcess
                 */

                // CREATE a StreamInsight APPLICATION in the server
                var myApp = server.CreateApplication("serverApp");

                // DEFINE a simple SOURCE (returns a point event every second)
                const string machineName = "";
                const string serverClass = "OPCLabs.KitEventServer.2";
                var observable = AENotificationObservable.Create(
                    machineName,
                    serverClass,
                    100,
                    AESubscriptionFilter.Empty,
                    AEAttributeSetDictionary.Empty);
                var mySource = myApp
                    .DefineObservable(() => observable)
                    .ToPointStreamable(
                        eventArgs => PointEvent.CreateInsert(
                            DateTimeOffset.Now,
                            (AENotificationPayload)eventArgs),
                        AdvanceTimeSettings.StrictlyIncreasingStartTime);

                // DEPLOY the source to the server for clients to use
                mySource.Deploy("serverSource");

                // Compose a QUERY over the source (return events with severity 20 or
higher)
                var myQuery = from e in mySource
                              where e.EventDataPayload.Severity >= 20
                              select e;

                // DEFINE a simple observer SINK (writes the value to the server
console)
                var mySink = myApp.DefineObserver(() =>
Observer.Create<AENotificationPayload>(
                    payload => Console.WriteLine("sink_Server..: {0}", payload)));

                // DEPLOY the sink to the server for clients to use
                mySink.Deploy("serverSink");

                // BIND the query to the sink and RUN it
                var binding = myQuery.Bind(mySink);
                Debug.Assert(binding != null);
                using (/*var proc = */binding.Run("serverProcess"))
                {
                    // Wait for the user stops the server
                    Console.WriteLine("-----------------------------------------------
```

```
----------------");
                    Console.WriteLine("MyStreamInsightServer is running, press Enter to
stop the server");
                    Console.WriteLine("-----------------------------------------------
----------------");
                    Console.WriteLine(" ");
                    Console.ReadLine();
                }
                host.Close();
            }
        }
    }
}
```

13.1.4.1.7 Installed Examples - Console - SimpleDAStreamInsightApplication

This example shows how to create an OPC Data Access event source, and query it for events carrying even data value.

The main program:

C#

```
// This example shows how to create an OPC Data Access event source, and query it for
events carrying even data value.

using System;
using System.Diagnostics;
using System.Reactive;
using System.ServiceModel;
using Microsoft.ComplexEventProcessing;
using Microsoft.ComplexEventProcessing.Linq;
using Microsoft.ComplexEventProcessing.ManagementService;
using OpcLabs.EasyOpc.DataAccess.ComplexEventProcessing;
using OpcLabs.EasyOpc.DataAccess.Reactive;

namespace SimpleDAStreamInsightApplication
{
    class Program
    {
        static void Main()
        {
            // Create an embedded StreamInsight server
            //using (var server = Server.Create("Default"))
            using (var server = Server.Create("Instance1"))
            {
                Debug.Assert(server != null);

                // Create a local end point for the server embedded in this program
                var managementService = server.CreateManagementService();
                Debug.Assert(managementService != null);
                var host = new ServiceHost(managementService);
```

```
                host.AddServiceEndpoint(typeof(IManagementService), new
WSHttpBinding(SecurityMode.Message),
                    "http://localhost/MyStreamInsightServer");
                host.Open();

                /* The following entities will be defined and available in the server
for other clients:
                 * serverApp
                 * serverSource
                 * serverSink
                 * serverProcess
                 */

                // CREATE a StreamInsight APPLICATION in the server
                var myApp = server.CreateApplication("serverApp");

                // DEFINE a simple SOURCE (returns a point event every second)
                const string machineName = "";
                const string serverClass = "OPCLabs.KitServer.2";
                const string itemId = "Simulation.Incrementing (1 s)";
                var observable = DAItemChangedObservable.Create<int>(machineName,
serverClass, itemId, 100);
                var mySource = myApp
                    .DefineObservable(() => observable)
                    .ToPointStreamable(
                        eventArgs =>
                            PointEvent.CreateInsert(DateTimeOffset.Now,
                            (DAItemChangedPayload<int>)eventArgs),
                        AdvanceTimeSettings.StrictlyIncreasingStartTime);

                // DEPLOY the source to the server for clients to use
                mySource.Deploy("serverSource");

                // Compose a QUERY over the source (return every event carrying even
data value)
                var myQuery = from e in mySource
                              where e.VtqPayload.Value % 2 == 0
                              select e;

                // DEFINE a simple observer SINK (writes the value to the server
console)
                var mySink = myApp.DefineObserver(() =>
Observer.Create<DAItemChangedPayload<int>>(
                    payload => Console.WriteLine("sink_Server..: {0}", payload)));

                // DEPLOY the sink to the server for clients to use
                mySink.Deploy("serverSink");

                // BIND the query to the sink and RUN it
                var binding = myQuery.Bind(mySink);
                Debug.Assert(binding != null);
                using (/*var proc = */binding.Run("serverProcess"))
                {
                    // Wait for the user stops the server
                    Console.WriteLine("------------------------------------------------
----------------");
                    Console.WriteLine("MyStreamInsightServer is running, press Enter to
```

```
stop the server");
                    Console.WriteLine("-----------------------------------------------
----------------");
                    Console.WriteLine(" ");
                    Console.ReadLine();
                }
                host.Close();
            }
        }
    }
}
```

13.1.4.1.8 Installed Examples - Console - SimpleLogToSql

Logs OPC Data Access item changes into an SQL database, using a subscription.Values of all data types are stored in a single SQL_VARIANT column.

The main program:

C#

```
// SimpleLogToSql: Logs OPC Data Access item changes into an SQL database, using a
subscription. Values of all data types are
// stored in a single SQL_VARIANT column.

// The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
// directory. The example assumes that the database is already created.

using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;

namespace SimpleLogToSql
{
    class Program
    {
        static void Main()
        {
            const string connectionString =
                "Data Source=(local);Initial Catalog=QuickOPCExamples;Integrated
Security=true";

            Console.WriteLine("Starting up...");
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Create all necessary ADO.NET objects.
                var adapter = new SqlDataAdapter("SELECT * FROM SimpleLog",
```

```
connection);
                var dataSet = new DataSet();
                adapter.FillSchema(dataSet, SchemaType.Source, "SimpleLog");
                adapter.InsertCommand = new
SqlCommandBuilder(adapter).GetInsertCommand();
                DataTable table = dataSet.Tables["SimpleLog"];
                Debug.Assert(table != null);

                Console.WriteLine("Logging for 30 seconds...");
                // Subscribe to an OPC item, using an anonymous method to process the
notifications.
                int handle = EasyDAClient.SharedInstance.SubscribeItem(
                    "",
                    "OPCLabs.KitServer.2",
                    "Simulation.Incrementing (1 s)",
                    100,
                    (_, eventArgs) =>
                        {
                            Debug.Assert(eventArgs != null);
                            Console.Write(".");

                            // In this example, we only log valid data. Production
logger would also log errors.
                            if (eventArgs.Vtq != null)
                            {
                                // Fill a DataRow with the OPC data, and add it to a
DataTable.
                                Debug.Assert(table.Rows != null);
                                table.Rows.Clear();
                                DataRow row = table.NewRow();
                                row["ItemID"] =
eventArgs.Arguments.ItemDescriptor.ItemId;
                                row["Value"] = eventArgs.Vtq.Value;
                                row["Timestamp"] = (eventArgs.Vtq.Timestamp <
(DateTime)SqlDateTime.MinValue)
                                                       ? (DateTime)SqlDateTime.MinValue
                                                       : eventArgs.Vtq.Timestamp;
                                row["Quality"] = (short)eventArgs.Vtq.Quality;

                                Debug.Assert(table.Rows != null);
                                table.Rows.Add(row);

                                // Update the underlying DataSet using an insert
command.
                                adapter.Update(dataSet, "SimpleLog");
                            }
                        }
                    );
                System.Threading.Thread.Sleep(30*1000);
                Console.WriteLine();

                Console.WriteLine("Shutting down...");
                EasyDAClient.SharedInstance.UnsubscribeItem(handle);
            }

            Console.WriteLine("Finished.");
        }
```

```
    }
}
```

VB.NET

```
' SimpleLogToSql: Logs OPC Data Access item changes into an SQL database, using a
subscription. Values of all data types are
' stored in a single SQL_VARIANT column.

' The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
' directory. The example assumes that the database is already created.
Imports System.Data.SqlClient
Imports System.Data.SqlTypes
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Friend Class Program
    Shared WithEvents _client As New EasyDAClient
    Shared _adapter As SqlDataAdapter
    Shared _dataSet As DataSet
    Shared _table As DataTable

    Shared Sub Main()
        Const connectionString As String = "Data Source=(local);Initial
Catalog=QuickOPCExamples;Integrated Security=true"

        Console.WriteLine("Starting up...")
        Using connection = New SqlConnection(connectionString)
            connection.Open()

            ' Create all necessary ADO.NET objects.
            _adapter = New SqlDataAdapter("SELECT * FROM SimpleLog", connection)
            _dataSet = New DataSet()
            _adapter.FillSchema(_dataSet, SchemaType.Source, "SimpleLog")
            _adapter.InsertCommand = (New
SqlCommandBuilder(_adapter)).GetInsertCommand()
            _table = _dataSet.Tables("SimpleLog")
            Debug.Assert(_table IsNot Nothing)

            Console.WriteLine("Logging for 30 seconds...")
            ' Subscribe to an OPC item, using an anonymous method to process the
notifications.
            Dim handle As Integer = _client.SubscribeItem( _
                "", _
                "OPCLabs.KitServer.2", _
                "Simulation.Incrementing (1 s)", _
                100)
            Threading.Thread.Sleep(30 * 1000)
            Console.WriteLine()

            Console.WriteLine("Shutting down...")
            _client.UnsubscribeItem(handle)
        End Using

        Console.WriteLine("Finished.")
    End Sub
```

```
    Private Shared Sub ItemChanged(ByVal sender As Object, ByVal eventArgs As
EasyDAItemChangedEventArgs) Handles _client.ItemChanged
        Debug.Assert(eventArgs IsNot Nothing)
        Console.Write(".")

        ' In this example, we only log valid data. Production logger would also log
errors.
        If eventArgs.Vtq IsNot Nothing Then
            ' Fill a DataRow with the OPC data, and add it to a DataTable.
            Debug.Assert(_table.Rows IsNot Nothing)
            _table.Rows.Clear()
            Dim row As DataRow = _table.NewRow()
            row("ItemID") = eventArgs.Arguments.ItemDescriptor.ItemId
            row("Value") = eventArgs.Vtq.Value
            row("Timestamp") = If(eventArgs.Vtq.Timestamp <
CDate(SqlDateTime.MinValue), CDate(SqlDateTime.MinValue), eventArgs.Vtq.Timestamp)
            row("Quality") = CShort(Fix(eventArgs.Vtq.Quality))

            Debug.Assert(_table.Rows IsNot Nothing)
            _table.Rows.Add(row)

            ' Update the underlying DataSet using an insert command.
            _adapter.Update(_dataSet, "SimpleLog")
        End If
    End Sub
End Class
```

13.1.4.1.9 Installed Examples - Console - SimpleTrillApplication

This example shows how to query real-time OPC UA data using Trill.

The main program:

C#

```
// Shows how to query real-time OPC UA data using Trill. The query looks for event
sequences where the previous value is
// less than 42, and the current value is greater than or equal to 42.

using System;
using System.Reactive.Linq;
using Microsoft.StreamProcessing;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Generic;
using OpcLabs.EasyOpc.UA.Reactive;

namespace SimpleTrillApplication
{
    public sealed class Program
    {
```

```
        public static void Main(string[] args)
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
                // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
                // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Console.WriteLine("Creating source observable...");
            // Create a data change observable for specified OPC UA node value.
            IObservable<UAAttributeData<byte>> sourceObservable =
UADataChangeNotificationObservable
                .Create<byte>(endpointDescriptor, "nsu=http://test.org/UA/Data/
;i=10846", 200)
                .Where(eventArgs => eventArgs.Succeeded)    // ignore events that carry
failures
                .Select(eventArgs => eventArgs.TypedAttributeData);

            Console.WriteLine("Creating input streamable...");
            // Load the observable as a stream in Trill, injecting a punctuation every
second. Because we use
            // FlushPolicy.FlushOnPunctuation, this will also flush the data every
second.
            IObservableIngressStreamable<UAAttributeData<byte>> inputStreamable =
                sourceObservable.Select(attributeData =>
                        StreamEvent.CreateStart(attributeData.SourceTimestamp.Ticks,
attributeData))
                    .ToStreamable(
                        DisorderPolicy.Drop(),
                        FlushPolicy.FlushOnPunctuation,

PeriodicPunctuationPolicy.Time((ulong)TimeSpan.FromSeconds(1).Ticks));

            // We will be building a query that takes a stream of UAAttributeData<byte>
events.
            Console.WriteLine("Creating the query...");
            // Query: look for pattern of [value < 42] --> [value >= 42]
            IStreamable<Empty, Tuple<byte, byte>> query = inputStreamable
                .AlterEventDuration(TimeSpan.FromSeconds(10).Ticks)
                .Detect(
                    default(Tuple<byte, byte>), // register to store the value
                    pattern => pattern
                        .SingleElement(e => e.TypedValue < 42, (ts, ev, reg) => new
Tuple<byte, byte>(ev.TypedValue, 0))
                        .SingleElement(e => e.TypedValue >= 42, (ts, ev, reg) => new
Tuple<byte, byte>(reg.Item1, ev.TypedValue)));

            Console.WriteLine("Running the query for 20 seconds...");
            query.ToStreamEventObservable()
                .Take(TimeSpan.FromSeconds(20))
                .ForEachAsync(streamEvent => Console.WriteLine(streamEvent)).Wait();

            Console.WriteLine("Finished.");
        }
```

```
        // Example output:
        //Creating source observable...
        //Creating input streamable...
        //Creating the query...
        //Running the query for 20 seconds...
        //[Punctuation: 637080168490000000]
        //[Interval: 637080168493266740 - 637080168591235679, (17, 76)]
        //[Punctuation: 637080168500000000]
        //[Interval: 637080168509585008 - 637080168607485434, (28, 197)]
        //[Punctuation: 637080168510000000]
        //[Punctuation: 637080168520000000]
        //[Punctuation: 637080168530000000]
        //[Punctuation: 637080168540000000]
        //[Interval: 637080168540136670 - 637080168638105248, (23, 108)]
        //[Punctuation: 637080168550000000]
        //[Interval: 637080168556412772 - 637080168654381015, (20, 157)]
        //[Punctuation: 637080168560000000]
        //[Interval: 637080168560631254 - 637080168658599730, (36, 53)]
        //[Punctuation: 637080168570000000]
        //[Interval: 637080168572974888 - 637080168670943594, (34, 113)]
        //[Punctuation: 637080168580000000]
        //[Interval: 637080168589224991 - 637080168687193550, (25, 176)]
        //[Punctuation: 637080168590000000]
        //[Punctuation: 637080168600000000]
        //[Interval: 637080168605474719 - 637080168703443726, (1, 161)]
        //[Punctuation: 637080168610000000]
        //[Interval: 637080168611568222 - 637080168709537000, (32, 222)]
        //[Punctuation: 637080168620000000]
        //[Punctuation: 637080168630000000]
        //[Punctuation: 637080168640000000]
        //[Interval: 637080168644224600 - 637080168742193693, (24, 87)]
        //[Punctuation: 637080168650000000]
        //[Punctuation: 637080168660000000]
        //[Interval: 637080168660475491 - 637080168758444908, (40, 230)]
        //[Punctuation: 637080168670000000]
        //Finished.
    }
}
```

13.1.4.1.10 Installed Examples - Console - SimpleUAStreamInsightApplication

This example shows how to create an OPC Unified Architecture data event source, and query it for events carrying even data value.

The main program:

C#

```
// This example shows how to create an OPC Unified Architecture data event source, and
query it for events carrying even data
```

```
// value.

using System;
using System.Diagnostics;
using System.Reactive;
using System.ServiceModel;
using Microsoft.ComplexEventProcessing;
using Microsoft.ComplexEventProcessing.Linq;
using Microsoft.ComplexEventProcessing.ManagementService;
using OpcLabs.EasyOpc.UA.ComplexEventProcessing;
using OpcLabs.EasyOpc.UA.Reactive;

namespace SimpleUAStreamInsightApplication
{
    class Program
    {
        static void Main()
        {
            // Create an embedded StreamInsight server
            //using (var server = Server.Create("Default"))
            using (var server = Server.Create("Instance1"))
            {
                Debug.Assert(server != null);

                // Create a local end point for the server embedded in this program
                var managementService = server.CreateManagementService();
                Debug.Assert(managementService != null);
                var host = new ServiceHost(managementService);
                host.AddServiceEndpoint(typeof(IManagementService), new
WSHttpBinding(SecurityMode.Message),
                    "http://localhost/MyStreamInsightServer");
                host.Open();

                /* The following entities will be defined and available in the server
for other clients:
                 * serverApp
                 * serverSource
                 * serverSink
                 * serverProcess
                 */

                // CREATE a StreamInsight APPLICATION in the server
                var myApp = server.CreateApplication("serverApp");

                // DEFINE a simple SOURCE (returns a point event every second)
                var observable = UADataChangeNotificationObservable.Create<int>(
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    "nsu=http://test.org/UA/Data/;i=11017", //
Data/Dynamic/UserScalar/Int32Value
                    100);
                var mySource = myApp
                    .DefineObservable(() => observable)
                    .ToPointStreamable(
                        eventArgs => PointEvent.CreateInsert(
                            DateTimeOffset.Now,
                            (UADataChangeNotificationPayload<int>)eventArgs),
                        AdvanceTimeSettings.StrictlyIncreasingStartTime);
```

```
                // DEPLOY the source to the server for clients to use
                mySource.Deploy("serverSource");

                // Compose a QUERY over the source (return every event carrying even
data value)
                var myQuery = from e in mySource
                              where e.AttributeDataPayload.Value % 2 == 0
                              select e;

                // DEFINE a simple observer SINK (writes the value to the server
console)
                var mySink = myApp.DefineObserver(() =>
Observer.Create<UADataChangeNotificationPayload<int>>(
                    payload => Console.WriteLine("sink_Server..: {0}", payload)));

                // DEPLOY the sink to the server for clients to use
                mySink.Deploy("serverSink");

                // BIND the query to the sink and RUN it
                var binding = myQuery.Bind(mySink);
                Debug.Assert(binding != null);
                using (/*var proc = */binding.Run("serverProcess"))
                {
                    // Wait for the user stops the server
                    Console.WriteLine("-----------------------------------------------
----------------");
                    Console.WriteLine("MyStreamInsightServer is running, press Enter to
stop the server");
                    Console.WriteLine("-----------------------------------------------
----------------");
                    Console.WriteLine(" ");
                    Console.ReadLine();
                }
                host.Close();
            }
        }
    }
}
```

13.1.4.1.11 Installed Examples - Console - SubscribeFromXml

Loads list of OPC items from an XML file and subscribes to them.

The main program:

C#

```
using System;
using System.Threading;
using System.Xml;
using System.Xml.Serialization;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
```

```
namespace SubscribeFromXml
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Loading items from XML file...");
            var xmlSerializer = new XmlSerializer(typeof(DAItemGroupArguments[]));
            var xmlReader = XmlReader.Create("OpcItems.xml", new XmlReaderSettings {IgnoreWhitespace
= true});
            var argArray = (DAItemGroupArguments[]) (xmlSerializer.Deserialize(xmlReader));

            if (argArray != null)
            {
                Console.WriteLine("Subscribing for 30 seconds...");
                EasyDAClient.SharedInstance.SubscribeMultipleItems(argArray,
                                                                   (_, eventArgs) =>
                                                                   Console.WriteLine("{0}: {1}",
                                                                       // ReSharper disable
PossibleNullReferenceException

eventArgs.Arguments.ItemDescriptor.ItemId,
                                                                                         eventArgs.Vtq));
                                                                       // ReSharper restore
PossibleNullReferenceException
                Thread.Sleep(30 * 1000);

                Console.WriteLine("Unsubscribing...");
                EasyDAClient.SharedInstance.UnsubscribeAllItems();
            }

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
Imports System.Threading
Imports System.Xml
Imports System.Xml.Serialization
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel


Friend Class Program
    Shared WithEvents _client As New EasyDAClient

    Shared Sub Main()
        Console.WriteLine("Loading items from XML file...")
        Dim xmlSerializer = New XmlSerializer(GetType(DAItemGroupArguments()))
        Dim xmlReader = Xml.XmlReader.Create("OpcItems.xml", New XmlReaderSettings With
{.IgnoreWhitespace = True})
        Dim argArray = CType(xmlSerializer.Deserialize(xmlReader), DAItemGroupArguments())

        If argArray IsNot Nothing Then
            Console.WriteLine("Subscribing for 30 seconds...")
            _client.SubscribeMultipleItems(argArray)
            Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            _client.UnsubscribeAllItems()
        End If

        Console.WriteLine("Finished.")
```

```
    End Sub

    Private Shared Sub ItemChanged(ByVal sender As Object, ByVal eventArgs As
EasyDAItemChangedEventArgs) Handles _client.ItemChanged
        Console.WriteLine("{0}: {1}", eventArgs.Arguments.ItemDescriptor.ItemId, eventArgs.Vtq)
    End Sub
End Class
```

The XML file with list of items:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfDAItemGroupArguments xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <DAItemGroupArguments>
    <State xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib.OperationModel">
      <TypeFullName
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib.Xml">System.Int32</TypeFullName>
      <Instance xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib.Xml">1</Instance>
    </State>
    <ServerDescriptor
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess.OperationModel">
      <UrlString
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib">opcda:OPCLabs.KitServer.2</UrlString>

    </ServerDescriptor>
    <ItemDescriptor
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess.OperationModel">
      <NodeId
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc">Simulation.Random</NodeId>
    </ItemDescriptor>
    <GroupParameters
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess.OperationModel">
      <RequestedUpdateRate
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess">1000</RequestedUpdateRate>

      <PercentDeadband
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess">0.0</PercentDeadband>
    </GroupParameters>
  </DAItemGroupArguments>
  <DAItemGroupArguments>
    <State xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib.OperationModel">
      <TypeFullName
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib.Xml">System.Int32</TypeFullName>
      <Instance xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib.Xml">2</Instance>
    </State>
    <ServerDescriptor
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess.OperationModel">
      <UrlString
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.BaseLib">opcda:OPCLabs.KitServer.2</UrlString>

    </ServerDescriptor>
    <ItemDescriptor
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess.OperationModel">
      <NodeId xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc">Simulation.Ramp (10
s)</NodeId>
    </ItemDescriptor>
    <GroupParameters
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess.OperationModel">
      <RequestedUpdateRate
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess">1000</RequestedUpdateRate>
```

```
      <PercentDeadband
xmlns="http://schemas.datacontract.org/2004/07/OpcLabs.EasyOpc.DataAccess">0.0</PercentDeadband>
    </GroupParameters>
  </DAItemGroupArguments>
</ArrayOfDAItemGroupArguments>
```

13.1.4.1.12 Installed Examples - Console - UAConsoleLiveMapping

Creates an object structure for a boiler, describes its mapping into OPC Unified Architecture server using attributes, and then performs the live mapping. Boiler data is then read, written and/or subscribed to using plain .NET object access..

The main program:

C#

```
// UAConsoleLiveMapping: Creates an object structure for a boiler, describes its
mapping into OPC Unified Architecture server
// using attributes, and then performs the live mapping. Boiler data is then read,
written and/or subscribed to using plain .NET
// object access.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.LiveMapping;
using OpcLabs.EasyOpc.UA.LiveMapping.Extensions;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UAConsoleLiveMapping
{
    class Program
    {
        static void Main()
        {
            // the OPC server
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Console.WriteLine();
            Console.WriteLine("Mapping our data structures to OPC...");
            var mapper = new UAClientMapper();
            var boiler1 = new Boiler();
            mapper.Map(boiler1, new UAMappingContext
            {
                EndpointDescriptor = endpointDescriptor,
                // The NodeDescriptor below determines where in the OPC address space
```

```
we want to map our data to.
                NodeDescriptor = new UANodeDescriptor
                    {
                        // '#' is a reserved character in a browse name, and must be
escaped by '&' in the path below.
                        BrowsePath = UABrowsePath.Parse("[ObjectsFolder]/Boilers/Boiler
&#1", "http://opcfoundation.org/UA/Boiler/")
                    },
                MonitoringParameters = 1000,  // requested sampling interval (for
subscriptions)
            });

            Console.WriteLine();
            Console.WriteLine("Starting the simulation of the boiler in the server,
using an OPC method call...");
            // Currently there is no live mapping for OPC methods, therefore we call
the OPC method in a traditional way.
            try
            {
                EasyUAClient.SharedInstance.CallMethod(
                    endpointDescriptor,
                    UABrowsePath.Parse("[ObjectsFolder]/Boilers/Boiler &#1/Simulation",
"http://opcfoundation.org/UA/Boiler/"),
                    UABrowsePath.Parse("
[nsu=http://opcfoundation.org/UA/Boiler/;i=1287].Start",
"http://opcfoundation.org/UA/"));
            }
            catch (UAException)
            {
                // Production code would test the current state of the simulation
first, and also handle the exception here.
            }

            Console.WriteLine();
            Console.WriteLine("Reading all data of the boiler...");
            mapper.Read();
            Console.WriteLine("Drum level is: {0}",
boiler1.Drum.LevelIndicator.Output);

            Console.WriteLine();
            Console.WriteLine("Writing new setpoint value...");
            boiler1.LevelController.SetPoint = 50.0;
            Debug.Assert(boiler1.LevelController != null);
            mapper.WriteTarget(boiler1.LevelController, /*recurse:*/false);

            Console.WriteLine();
            Console.WriteLine("Subscribing to boiler data changes...");
            mapper.Subscribe(/*active:*/true);

            Thread.Sleep(30 * 1000);

            Console.WriteLine();
            Console.WriteLine("Unsubscribing from boiler data changes...");
            mapper.Subscribe(/*active:*/false);

            Console.WriteLine();
            Console.WriteLine("Press Enter to continue...");
```

```
            Console.ReadLine();
        }
    }
}
```

VB.NET

```
' UAConsoleLiveMapping: Creates an object structure for a boiler, describes its mapping
into OPC Unified Architecture server
' using attributes, and then performs the live mapping. Boiler data is then read,
written and/or subscribed to using plain .NET
' object access.
Imports System.Threading
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.LiveMapping
Imports OpcLabs.EasyOpc.UA.LiveMapping.Extensions
Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.OperationModel

Friend Class Program
    Shared Sub Main()
        ' Define which server we will work with.
        Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
        ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

        Console.WriteLine()
        Console.WriteLine("Mapping our data structures to OPC...")
        Dim mapper = New UAClientMapper()
        Dim boiler1 = New Boiler()
        mapper.Map(boiler1, New UAMappingContext With { _
                        .EndpointDescriptor = endpointDescriptor, _
                        .NodeDescriptor = New UANodeDescriptor With { _
                        .BrowsePath = UABrowsePath.Parse("[ObjectsFolder]/Boilers/Boiler
#1", "http://opcfoundation.org/UA/Boiler/")}, _
                        .MonitoringParameters = 1000}) ' requested sampling interval (for
subscriptions) -  local OPC server
        ' The NodeDescriptor below determines where in the OPC address space we want to
map our data to.

        Console.WriteLine()
        Console.WriteLine("Starting the simulation of the boiler in the server, using
an OPC method call...")
        ' Currently there is no live mapping for OPC methods, therefore we call the OPC
method in a traditional way.
        Try
            EasyUAClient.SharedInstance.CallMethod( _
                endpointDescriptor, _
                UABrowsePath.Parse("[ObjectsFolder]/Boilers/Boiler #1/Simulation",
"http://opcfoundation.org/UA/Boiler/"), _
                UABrowsePath.Parse("
[nsu=http://opcfoundation.org/UA/Boiler/;i=1287].Start",
"http://opcfoundation.org/UA/"))
        Catch e1 As UAException
            ' Production code would test the current state of the simulation first, and
also handle the exception here.
```

```
        End Try

        Console.WriteLine()
        Console.WriteLine("Reading all data of the boiler...")
        mapper.Read()
        Console.WriteLine("Drum level is: {0}", boiler1.Drum.LevelIndicator.Output)

        Console.WriteLine()
        Console.WriteLine("Writing new setpoint value...")
        boiler1.LevelController.SetPoint = 50.0
        Debug.Assert(boiler1.LevelController IsNot Nothing)
        mapper.WriteTarget(boiler1.LevelController, False) 'recurse:

        Console.WriteLine()
        Console.WriteLine("Subscribing to boiler data changes...")
        mapper.Subscribe(True) 'active:

        Thread.Sleep(30 * 1000)

        Console.WriteLine()
        Console.WriteLine("Unsubscribing from boiler data changes...")
        mapper.Subscribe(False) 'active:

        Console.WriteLine()
        Console.WriteLine("Press Enter to continue...")
        Console.ReadLine()
    End Sub
End Class
```

The Boiler class:

C#

```
using System;
using OpcLabs.BaseLib.LiveMapping;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.LiveMapping;

namespace UAConsoleLiveMapping
{
    // The Boiler and its constituents are described in our application domain terms,
the way we want to work with them.
    // Attributes are used to describe the correspondence between our types and
members, and OPC nodes.

    // This is how the boiler looks in OPC address space:
    //  - Boiler #1
    //      - CC1001                    (CustomController)
    //          - ControlOut
    //          - Description
    //          - Input1
    //          - Input2
    //          - Input3
    //      - Drum1001                  (BoilerDrum)
    //          - LIX001                (LevelIndicator)
```

```
    //              - Output
    //      - FC1001                    (FlowController)
    //          - ControlOut
    //          - Measurement
    //          - SetPoint
    //      - LC1001                    (LevelController)
    //          - ControlOut
    //          - Measurement
    //          - SetPoint
    //      - Pipe1001                  (BoilerInputPipe)
    //          - FTX001                (FlowTransmitter)
    //              - Output
    //      - Pipe1002                  (BoilerOutputPipe)
    //          - FTX002                (FlowTransmitter)
    //              - Output

    [UANamespace("http://opcfoundation.org/UA/Boiler/")]
    [UAType]
    class Boiler
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [UANode(BrowsePath = "/PipeX001")]
        public BoilerInputPipe InputPipe = new BoilerInputPipe();

        [UANode(BrowsePath = "/DrumX001")]
        public BoilerDrum Drum = new BoilerDrum();

        [UANode(BrowsePath = "/PipeX002")]
        public BoilerOutputPipe OutputPipe = new BoilerOutputPipe();

        [UANode(BrowsePath = "/FCX001")]
        public FlowController FlowController = new FlowController();

        [UANode(BrowsePath = "/LCX001")]
        public LevelController LevelController = new LevelController();

        [UANode(BrowsePath = "/CCX001")]
        public CustomController CustomController = new CustomController();
    }

    [UAType]
    class BoilerInputPipe
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [UANode(BrowsePath = "/FTX001")]
        public FlowTransmitter FlowTransmitter1 = new FlowTransmitter();

        [UANode(BrowsePath = "/ValveX001")]
        public Valve Valve = new Valve();
    }

    [UAType]
    class BoilerDrum
```

```
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [UANode(BrowsePath = "/LIX001")]
        public LevelIndicator LevelIndicator = new LevelIndicator();
    }

    [UAType]
    class BoilerOutputPipe
    {
        // Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

        [UANode(BrowsePath = "/FTX002")]
        public FlowTransmitter FlowTransmitter2 = new FlowTransmitter();
    }

    [UAType]
    class FlowController : GenericController
    {
    }

    [UAType]
    class LevelController : GenericController
    {
    }

    [UAType]
    class CustomController
    {
        [UANode, UAData(Operations = UADataMappingOperations.Write)]    // not readable
        public double Input1 { get; set; }

        [UANode, UAData(Operations = UADataMappingOperations.Write)]    // not readable
        public double Input2 { get; set; }

        [UANode, UAData(Operations = UADataMappingOperations.Write)]    // not readable
        public double Input3 { get; set; }

        [UANode, UAData(Operations = UADataMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double ControlOut { get; set; }

        [UANode, UAData]
        public string Description { get; set; }
    }

    [UAType]
    class FlowTransmitter : GenericSensor
    {
    }

    [UAType]
    class Valve : GenericActuator
    {
    }
```

```
    [UAType]
    class LevelIndicator : GenericSensor
    {
    }

    [UAType]
    class GenericController
    {
        [UANode, UAData(Operations = UADataMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double Measurement { get; set; }

        [UANode, UAData]
        public double SetPoint { get; set; }

        [UANode, UAData(Operations = UADataMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double ControlOut { get; set; }
    }

    [UAType]
    class GenericSensor
    {
        // Meta-members are filled in by information collected during mapping, and
allow access to it later from your code.
        // Alternatively, you can derive your class from UAMappedNode, which will bring
in many meta-members automatically.
        [MetaMember("NodeDescriptor")]
        public UANodeDescriptor NodeDescriptor { get; set; }

        [UANode, UAData(Operations = UADataMappingOperations.ReadAndSubscribe)] // no
OPC writing
        public double Output
        {
            get { return _output; }
            set
            {
                _output = value;
                Console.WriteLine("Sensor \"{0}\" output is now {1}.", NodeDescriptor,
value);
            }
        }

        private double _output;
    }

    [UAType]
    class GenericActuator
    {
        [UANode, UAData(Operations = UADataMappingOperations.Write)]    // generic
actuator input is not readable
        public double Input { get; set; }
    }
}
```

VB.NET

```
Imports OpcLabs.BaseLib.LiveMapping
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.LiveMapping


' The Boiler and its constituents are described in our application domain terms, the
way we want to work with them.
' Attributes are used to describe the correspondence between our types and members, and
OPC nodes.

' This is how the boiler looks in OPC address space:
'  - Boiler #1
'      - CC1001                    (CustomController)
'          - ControlOut
'          - Description
'          - Input1
'          - Input2
'          - Input3
'      - Drum1001                  (BoilerDrum)
'          - LIX001                (LevelIndicator)
'              - Output
'      - FC1001                    (FlowController)
'          - ControlOut
'          - Measurement
'          - SetPoint
'      - LC1001                    (LevelController)
'          - ControlOut
'          - Measurement
'          - SetPoint
'      - Pipe1001                  (BoilerInputPipe)
'          - FTX001                (FlowTransmitter)
'              - Output
'      - Pipe1002                  (BoilerOutputPipe)
'          - FTX002                (FlowTransmitter)
'              - Output

<UANamespace("http://opcfoundation.org/UA/Boiler/"), UAType()> _
Friend Class Boiler
    ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

    <UANode(BrowsePath:="/PipeX001")> _
    Public InputPipe As New BoilerInputPipe()

    <UANode(BrowsePath:="/DrumX001")> _
    Public Drum As New BoilerDrum()

    <UANode(BrowsePath:="/PipeX002")> _
    Public OutputPipe As New BoilerOutputPipe()

    <UANode(BrowsePath:="/FCX001")> _
    Public FlowController As New FlowController()

    <UANode(BrowsePath:="/LCX001")> _
```

```
    Public LevelController As New LevelController()

    <UANode(BrowsePath:="/CCX001")> _
    Public CustomController As New CustomController()
End Class

<UAType()> _
Friend Class BoilerInputPipe
    ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

    <UANode(BrowsePath:="/FTX001")> _
    Public FlowTransmitter1 As New FlowTransmitter()

    <UANode(BrowsePath:="/ValveX001")> _
    Public Valve As New Valve()
End Class

<UAType()> _
Friend Class BoilerDrum
    ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

    <UANode(BrowsePath:="/LIX001")> _
    Public LevelIndicator As New LevelIndicator()
End Class

<UAType()> _
Friend Class BoilerOutputPipe
    ' Specifying BrowsePath-s here only because we have named the class members
differently from OPC node names.

    <UANode(BrowsePath:="/FTX002")> _
    Public FlowTransmitter2 As New FlowTransmitter()
End Class

<UAType()> _
Friend Class FlowController
    Inherits GenericController

End Class

<UAType()> _
Friend Class LevelController
    Inherits GenericController

End Class

<UAType()> _
Friend Class CustomController
    <UANode(), UAData(Operations:=UADataMappingOperations.Write)>
    Public Property Input1 As Double

    <UANode(), UAData(Operations:=UADataMappingOperations.Write)>
    Public Property Input2 As Double

    <UANode(), UAData(Operations:=UADataMappingOperations.Write)>
```

```
    Public Property Input3 As Double

    <UANode(), UAData(Operations:=UADataMappingOperations.ReadAndSubscribe)>
    Public Property ControlOut As Double

    <UANode(), UAData()>
    Public Property Description As String
End Class

<UAType()> _
Friend Class FlowTransmitter
    Inherits GenericSensor

End Class

<UAType()> _
Friend Class Valve
    Inherits GenericActuator

End Class

<UAType()> _
Friend Class LevelIndicator
    Inherits GenericSensor

End Class

<UAType()> _
Friend Class GenericController
    <UANode(), UAData(Operations := UADataMappingOperations.ReadAndSubscribe)>
    Public Property Measurement As Double

    <UANode(), UAData()>
    Public Property SetPoint As Double

    <UANode(), UAData(Operations:=UADataMappingOperations.ReadAndSubscribe)>
    Public Property ControlOut As Double
End Class

<UAType()> _
Friend Class GenericSensor
    ' Meta-members are filled in by information collected during mapping, and allow
access to it later from your code.
    ' Alternatively, you can derive your class from UAMappedNode, which will bring in
many meta-members automatically.
    <MetaMember("NodeDescriptor")>
    Public Property NodeDescriptor As UANodeDescriptor

    <UANode(), UAData(Operations:=UADataMappingOperations.ReadAndSubscribe)>
    Public Property Output() As Double ' no OPC writing
        Get
            Return _output
        End Get
        Set(ByVal value As Double)
            _output = value
            Console.WriteLine("Sensor ""{0}"" output is now {1}.", NodeDescriptor,
value)
```

```
        End Set
    End Property

    Private _output As Double
End Class

<UAType()> _
Friend Class GenericActuator
    <UANode(), UAData(Operations := UADataMappingOperations.Write)>
    Public Property Input As Double
End Class
```

13.1.4.1.13 Installed Examples - Console - UASimpleLogToSql

Logs OPC Unified Architecture data changes into an SQL database, using a subscription. Values of all data types are stored in a single SQL_VARIANT column.

The main program:

C#

```
// UASimpleLogToSql: Logs OPC Unified Architecture data changes into an SQL database,
using a subscription. Values of all
// data types are stored in a single SQL_VARIANT column.

// The database creation script is in the ExamplesNet\MSSQL\QuickOPCExamples.sql file
under the product installation
// directory. The example assumes that the database is already created.

using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.UA;

namespace UASimpleLogToSql
{
    class Program
    {
        static void Main()
        {
            const string connectionString =
                "Data Source=(local);Initial Catalog=QuickOPCExamples;Integrated
Security=true";

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            Console.WriteLine("Starting up...");
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Create all necessary ADO.NET objects.
                var adapter = new SqlDataAdapter("SELECT * FROM UASimpleLog",
connection);
                var dataSet = new DataSet();
                adapter.FillSchema(dataSet, SchemaType.Source, "UASimpleLog");
                adapter.InsertCommand = new
SqlCommandBuilder(adapter).GetInsertCommand();
                DataTable table = dataSet.Tables["UASimpleLog"];
                Debug.Assert(table != null);

                Console.WriteLine("Logging for 30 seconds...");
                // Subscribe to an OPC item, using an anonymous method to process the
notifications.
                int handle = EasyUAClient.SharedInstance.SubscribeDataChange(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10853",
                    100,
                    (_, eventArgs) =>
                    {
                        Debug.Assert(eventArgs != null);
                        Console.Write(".");

                        // In this example, we only log valid data. Production logger
would also log errors.
                        if (eventArgs.AttributeData != null)
                        {
                            // Fill a DataRow with the OPC data, and add it to a
DataTable.
                            Debug.Assert(table.Rows != null);
                            table.Rows.Clear();
                            DataRow row = table.NewRow();
                            row["NodeID"] = eventArgs.Arguments.NodeDescriptor.NodeId;
                            row["Value"] = eventArgs.AttributeData.Value;
                            row["SourceTimestamp"] =
(eventArgs.AttributeData.SourceTimestamp < (DateTime)SqlDateTime.MinValue)
                                                    ? (DateTime)SqlDateTime.MinValue
                                                    :
eventArgs.AttributeData.SourceTimestamp;
                            row["ServerTimestamp"] =
(eventArgs.AttributeData.ServerTimestamp < (DateTime)SqlDateTime.MinValue)
                                                    ? (DateTime)SqlDateTime.MinValue
                                                    :
eventArgs.AttributeData.ServerTimestamp;
                            row["StatusCode"] =
eventArgs.AttributeData.StatusCode.InternalValue;

                            Debug.Assert(table.Rows != null);
                            table.Rows.Add(row);

                            // Update the underlying DataSet using an insert command.
                            adapter.Update(dataSet, "UASimpleLog");
```

```
                        }
                    }
                    );
                Thread.Sleep(30 * 1000);
                Console.WriteLine();

                Console.WriteLine("Shutting down...");
                EasyUAClient.SharedInstance.UnsubscribeMonitoredItem(handle);
            }

            Console.WriteLine("Finished.");
        }
    }
}
```

13.1.4.1.14 Installed Examples - Console - WcfClient1

Using a Web service provided by the WcfService1 project (under Web folder), gets and displays a value of an OPC item.

The main program:

C#

```
// WcfClient1: Using a Web service provided by the WcfService1 project (under Web folder),
gets and displays a value of
// an OPC item.

using System;

namespace WcfClient1
{
    class Program
    {
        static void Main()
        {
            var client = new Service1Client();

            // Use the 'client' variable to call operations on the service.
            Console.WriteLine(client.GetData());
            Console.ReadLine();

            // Always close the client.
            client.Close();
        }
    }
}
```

VB.NET

```
' WcfClient1: Using a Web service provided by the WcfService1 project (under Web folder), gets
and displays a value of
' an OPC item.


Friend Class Program
```

```
    Shared Sub Main()
        Dim client = New Service1Client()

        ' Use the 'client' variable to call operations on the service.
        Console.WriteLine(client.GetData())
        Console.ReadLine()

        ' Always close the client.
        client.Close()
    End Sub
End Class
```

The Service1 class:

C#

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.5456
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------



[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
[System.ServiceModel.ServiceContractAttribute(ConfigurationName="IService1")]
public interface IService1
{


[System.ServiceModel.OperationContractAttribute(Action="http://tempuri.org/IService1/GetData",
ReplyAction="http://tempuri.org/IService1/GetDataResponse")]
    string GetData();
}

[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public interface IService1Channel : IService1, System.ServiceModel.IClientChannel
{
}

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public partial class Service1Client : System.ServiceModel.ClientBase<IService1>, IService1
{

    public Service1Client()
    {
    }

    public Service1Client(string endpointConfigurationName) :
            base(endpointConfigurationName)
    {
    }

    public Service1Client(string endpointConfigurationName, string remoteAddress) :
```

```
            base(endpointConfigurationName, remoteAddress)
    {
    }

    public Service1Client(string endpointConfigurationName,
System.ServiceModel.EndpointAddress remoteAddress) :
            base(endpointConfigurationName, remoteAddress)
    {
    }

    public Service1Client(System.ServiceModel.Channels.Binding binding,
System.ServiceModel.EndpointAddress remoteAddress) :
            base(binding, remoteAddress)
    {
    }

    public string GetData()
    {
        return base.Channel.GetData();
    }
}
```

VB.NET

```
'------------------------------------------------------------------------------
' <auto-generated>
'     This code was generated by a tool.
'     Runtime Version:2.0.50727.5456
'
'     Changes to this file may cause incorrect behavior and will be lost if
'     the code is regenerated.
' </auto-generated>
'------------------------------------------------------------------------------



<System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0"),
System.ServiceModel.ServiceContractAttribute(ConfigurationName:="IService1")> _
Public Interface IService1


<System.ServiceModel.OperationContractAttribute(Action:="http://tempuri.org/IService1/GetData",
ReplyAction:="http://tempuri.org/IService1/GetDataResponse")> _
    Function GetData() As String
End Interface

<System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")> _
Public Interface IService1Channel
    Inherits IService1, System.ServiceModel.IClientChannel

End Interface

<System.Diagnostics.DebuggerStepThroughAttribute(),
System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")> _
Partial Public Class Service1Client
    Inherits System.ServiceModel.ClientBase(Of IService1)
    Implements IService1

    Public Sub New()
    End Sub
```

```
    Public Sub New(ByVal endpointConfigurationName As String)
        MyBase.New(endpointConfigurationName)
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String, ByVal remoteAddress As String)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub

    Public Sub New(ByVal endpointConfigurationName As String, ByVal remoteAddress As
System.ServiceModel.EndpointAddress)
        MyBase.New(endpointConfigurationName, remoteAddress)
    End Sub

    Public Sub New(ByVal binding As System.ServiceModel.Channels.Binding, ByVal remoteAddress
As System.ServiceModel.EndpointAddress)
        MyBase.New(binding, remoteAddress)
    End Sub

    Public Function GetData() As String Implements IService1.GetData
        Return MyBase.Channel.GetData()
    End Function
End Class
```

13.1.4.1.15 Installed Examples - Console - XmlEventLogger

Logs OPC Alarms and Events notifications into an XML file.

The main program:

C#

```
using System;
using System.Diagnostics;
using System.Xml;
using System.Xml.Serialization;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace XmlEventLogger
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Starting up...");
            var xmlSerializer = new XmlSerializer(typeof(EasyAENotificationEventArgs));
            var xmlWriter = XmlWriter.Create("OpcEvents.xml", new XmlWriterSettings
            {
                Indent = true,
                CloseOutput = true
            });
            // The root element can have any name you need, but the name below also
allows reading the log back as .NET array
            xmlWriter.WriteStartElement("ArrayOfEasyAENotificationEventArgs");
```

```
            Console.WriteLine("Logging for 30 seconds...");
            int handle = EasyAEClient.SharedInstance.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 100,
                (_, eventArgs) =>
                    {
                        Debug.Assert(eventArgs != null);
                        xmlSerializer.Serialize(xmlWriter, eventArgs);
                    });
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Shutting down...");
            EasyAEClient.SharedInstance.UnsubscribeEvents(handle);
            xmlWriter.WriteEndElement();    // not really necessary - XmlWriter would
write the end tag for us anyway
            xmlWriter.Close();

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
Imports System.Xml
Imports System.Xml.Serialization
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel


Friend Class Program
    Shared WithEvents _client As New EasyAEClient
    Shared ReadOnly XmlSerializer As New
XmlSerializer(GetType(EasyAENotificationEventArgs))
    Shared _xmlWriter As XmlWriter

    Shared Sub Main()
        Console.WriteLine("Starting up...")
        _xmlWriter = XmlWriter.Create("OpcEvents.xml", New XmlWriterSettings With
{.Indent = True, .CloseOutput = True})
        ' The root element can have any name you need, but the name below also allows
reading the log back as .NET array
        _xmlWriter.WriteStartElement("ArrayOfEasyAENotificationEventArgs")

        Console.WriteLine("Logging for 30 seconds...")
        Dim handle As Integer = _client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
100)
        Threading.Thread.Sleep(30 * 1000)

        Console.WriteLine("Shutting down...")
        _client.UnsubscribeEvents(handle)
        _xmlWriter.WriteEndElement() ' not really necessary - XmlWriter would write the
end tag for us anyway
        _xmlWriter.Close()

        Console.WriteLine("Finished.")
    End Sub
```

```
    Private Shared Sub Notification(ByVal sender As Object, ByVal eventArgs As Object)
Handles _client.Notification
        Debug.Assert(eventArgs IsNot Nothing)
        XmlSerializer.Serialize(_xmlWriter, eventArgs)
    End Sub
End Class
```

13.1.4.1.16 Installed Examples - Console - XmlLogger

Logs OPC Data Access item changes into an XML file.

The main program:

C#

```
using System;
using System.Diagnostics;
using System.Xml;
using System.Xml.Serialization;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace XmlLogger
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Starting up...");
            var xmlSerializer = new XmlSerializer(typeof(EasyDAItemChangedEventArgs));
            var xmlWriter = XmlWriter.Create("OpcData.xml", new XmlWriterSettings
                                    {
                                        Indent = true,
                                        CloseOutput = true
                                    });
            // The root element can have any name you need, but the name below also
allows reading the log back as .NET array
            xmlWriter.WriteStartElement("ArrayOfEasyDAItemChangedEventArgs");

            Console.WriteLine("Logging for 30 seconds...");
            int handle = EasyDAClient.SharedInstance.SubscribeItem("",
"OPCLabs.KitServer.2",
                "Simulation.Incrementing (1 s)", 100,
               (_, eventArgs) =>
                   {
                       Debug.Assert(eventArgs != null);
                       xmlSerializer.Serialize(xmlWriter, eventArgs);
                   });
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Shutting down...");
            EasyDAClient.SharedInstance.UnsubscribeItem(handle);
```

```
            xmlWriter.WriteEndElement();    // not really necessary - XmlWriter would
write the end tag for us anyway
            xmlWriter.Close();

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
Imports System.Xml
Imports System.Xml.Serialization
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel


Friend Class Program
    Shared WithEvents _client As New EasyDAClient
    Shared ReadOnly XmlSerializer As New
XmlSerializer(GetType(EasyDAItemChangedEventArgs))
    Shared _xmlWriter As XmlWriter

    Shared Sub Main()
        Console.WriteLine("Starting up...")
        _xmlWriter = XmlWriter.Create("OpcData.xml", New XmlWriterSettings With
{.Indent = True, .CloseOutput = True})
        ' The root element can have any name you need, but the name below also allows
reading the log back as .NET array
        _xmlWriter.WriteStartElement("ArrayOfEasyDAItemChangedEventArgs")

        Console.WriteLine("Logging for 30 seconds...")
        Dim handle As Integer = _client.SubscribeItem( _
            "", _
            "OPCLabs.KitServer.2", _
            "Simulation.Incrementing (1 s)", _
            100)
        Threading.Thread.Sleep(30 * 1000)

        Console.WriteLine("Shutting down...")
        _client.UnsubscribeItem(handle)
        _xmlWriter.WriteEndElement() ' not really necessary - XmlWriter would write the
end tag for us anyway
        _xmlWriter.Close()

        Console.WriteLine("Finished.")
    End Sub

    Private Shared Sub ItemChanged(ByVal sender As Object, ByVal eventArgs As
EasyDAItemChangedEventArgs) Handles _client.ItemChanged
        Debug.Assert(eventArgs IsNot Nothing)
        XmlSerializer.Serialize(_xmlWriter, eventArgs)
    End Sub
End Class
```

12.1.4.2 Installed Examples - Web

13.1.4.2.1 Installed Examples - Web - AutoRefreshWeb

Web application with a screen that refreshes itself periodically.

The default page code-behind:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.

using JetBrains.Annotations;
using OpcLabs.EasyOpc.DataAccess;
using System;
using System.Web.UI.WebControls;
using OpcLabs.EasyOpc.OperationModel;

namespace AutoRefreshWeb
{
    // ReSharper disable InconsistentNaming
    public partial class _Default : System.Web.UI.Page
    // ReSharper restore InconsistentNaming
    {
        // ReSharper disable InconsistentNaming
        protected void Page_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Read(TextBox1, "Simulation.Ramp (10 s)");
            Read(TextBox2, "Simulation.Random");
            Read(TextBox3, "Simulation.Incrementing (1 s)");
        }

        protected void Read([NotNull] TextBox textBox, [NotNull] string itemId)
        {
            var client = new EasyDAClient();
            try
            {
                object value = client.ReadItemValue("", "OPCLabs.KitServer.2", itemId);
                textBox.Text = (value == null) ? "" : value.ToString();
            }
            catch (OpcException e)
            {
                textBox.Text = "***" + e.GetBaseException().Message;
            }
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.DataAccess
Imports System.Web.UI.WebControls
```

```
Imports OpcLabs.EasyOpc.OperationModel ' ReSharper disable InconsistentNaming
Partial Public Class _Default
    ' ReSharper restore InconsistentNaming
    Inherits UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
        Read(TextBox1, "Simulation.Ramp (10 s)")
        Read(TextBox2, "Simulation.Random")
        Read(TextBox3, "Simulation.Incrementing (1 s)")
    End Sub

    Protected Sub Read(ByVal textBox As TextBox, ByVal itemId As String)
        Dim client = New EasyDAClient()
        Try
            textBox.Text = client.ReadItemValue("", "OPCLabs.KitServer.2",
itemId).ToString()
        Catch e As OpcException
            textBox.Text = "***" & e.GetBaseException().Message
        End Try
    End Sub
End Class
```

13.1.4.2.2 Installed Examples - Web - BrowseBranchesWeb

Browses the branches in the OPC server (ASP.NET Web application).

The default page code-behind:

C#

```
using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace BrowseBranchesWeb
{
    // ReSharper disable InconsistentNaming
    public partial class _Default : System.Web.UI.Page
    // ReSharper restore InconsistentNaming
    {

        // ReSharper disable InconsistentNaming
        protected void Page_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            var client = new EasyDAClient();

            try
            {
                var elementDictionary = client.BrowseBranches("",
"OPCLabs.KitServer.2");
                Debug.Assert(elementDictionary.Keys != null);
```

```
                foreach (var key in elementDictionary.Keys)
                {
                    Debug.Assert(key != null);
                    var branchElement = elementDictionary[key];
                    Debug.Assert(branchElement != null);
                    ListBox1.Items.Add(branchElement.ItemId);
                }
                ListBox1.Visible = true;

            }
            catch (OpcException ex)
            {
                TextBox1.Text = ex.GetBaseException().Message;
                TextBox1.Visible = true;
            }

        }

    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);

        Load += Page_Load;
    }
    }
}
```

VB.NET

```
Imports OpcLabs.EasyOpc.DataAccess

Imports OpcLabs.EasyOpc.OperationModel ' ReSharper disable InconsistentNaming
Partial Public Class _Default
    ' ReSharper restore InconsistentNaming
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
        Dim client As New EasyDAClient

        Try
            Dim branchElements = client.BrowseBranches("", "OPCLabs.KitServer.2")

            For Each branchElement In branchElements
                ListBox1.Items.Add(branchElement.ItemId)
            Next
            ListBox1.Visible = True

        Catch ex As OpcException
            TextBox1.Text = ex.GetBaseException.Message
            TextBox1.Visible = True
        End Try

    End Sub
End Class
```

13.1.4.2.3 Installed Examples - Web - BrowseServersWeb

Browses the available OPC servers (ASP.NET Web application).

The default page code-behind:

C#

```
using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace BrowseServersWeb
{
    // ReSharper disable InconsistentNaming
    public partial class _Default : System.Web.UI.Page
    // ReSharper restore InconsistentNaming
    {

        // ReSharper disable InconsistentNaming
        protected void Page_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            var client = new EasyDAClient();

            try
            {
                ServerElementCollection serverElements = client.BrowseServers("");
                Debug.Assert(serverElements.Keys != null);

                // ReSharper disable once LoopCanBePartlyConvertedToQuery
                foreach (ServerElement serverElement in serverElements)
                {
                    Debug.Assert(serverElement != null);
                    ListBox1.Items.Add(serverElement.ServerClass);
                }
                ListBox1.Visible = true;

            }
            catch (OpcException ex)
            {
                TextBox1.Text = ex.GetBaseException().Message;
                TextBox1.Visible = true;
            }
        }


    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);

        Load += Page_Load;
    }
```

```
    }
}
```

VB.NET

```
Imports OpcLabs.EasyOpc.DataAccess

Imports OpcLabs.EasyOpc.OperationModel ' ReSharper disable InconsistentNaming
Partial Public Class _Default
    ' ReSharper restore InconsistentNaming
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
        Dim client As New EasyDAClient

        Try
            Dim serverElements = client.BrowseServers("")

            For Each serverElement In serverElements
                ListBox1.Items.Add(serverElement.ServerClass)
            Next
            ListBox1.Visible = True

        Catch ex As OpcException
            TextBox1.Text = ex.GetBaseException.Message
            TextBox1.Visible = True
        End Try
    End Sub

End Class
```

13.1.4.2.4 Installed Examples - Web - DataGridWebApplication

Demonstrates how easily can WebControls.GridView be populated with data read from OPC Data Access server.

The default page code-behind:

C#

```
// DataGridWebApplication: Demonstrates how easily can GridView be populated with data
read from OPC Data Access server.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.DataAccess;

namespace DataGridWebApplication
{
    // ReSharper disable InconsistentNaming
    public partial class _Default : System.Web.UI.Page
    // ReSharper restore InconsistentNaming
```

```
    {
        class Row
        {
            // ReSharper disable UnusedAutoPropertyAccessor.Local
            public string ItemId { get; set; }
            public string Value { get; set; }
            // ReSharper restore UnusedAutoPropertyAccessor.Local
        }

        protected void Page_Load(object sender, EventArgs e)
        {
            var client = new EasyDAClient();
            var itemDescriptors = new DAItemDescriptor[]
                {
                    "Simulation.Register_BOOL",
                    "Simulation.Register_I2",
                    "Demo.Ramp",
                    "Demo.Single"

                };

            ValueResult[] valueResults =
client.ReadMultipleItemValues("OPCLabs.KitServer.2", itemDescriptors);

            var data = new List<Row>();
            // ReSharper disable once LoopCanBeConvertedToQuery
            for (int i = 0; i < itemDescriptors.Length; i++)
                data.Add(new Row
                    {
                        // ReSharper disable once PossibleNullReferenceException
                        ItemId = itemDescriptors[i].ItemId,
                        // ReSharper disable PossibleNullReferenceException
                        Value = valueResults[i].Value.ToString()
                        // ReSharper restore PossibleNullReferenceException
                    });

            GridView1.DataSource = data;
            GridView1.DataBind();
        }
    }
}
```

VB.NET

```
' DataGridWebApplication: Demonstrates how easily can GridView be populated with data
read from OPC Data Access server.
Imports OpcLabs.EasyOpc.DataAccess

Imports OpcLabs.BaseLib.OperationModel ' ReSharper disable InconsistentNaming
Partial Public Class _Default
    Inherits UI.Page
    ' ReSharper restore InconsistentNaming
    Private Class Row
        Public Property ItemId As String

        Public Property Value As String
    End Class
```

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
        Dim client = New EasyDAClient()
        Dim itemDescriptors = New DAItemDescriptor() {"Simulation.Register_BOOL",
"Simulation.Register_I2", "Demo.Ramp", "Demo.Single"}

        Dim valueResults() As ValueResult =
client.ReadMultipleItemValues("OPCLabs.KitServer.2", itemDescriptors)

        Dim data = New List(Of Row)()
        For i As Integer = 0 To itemDescriptors.Length - 1
            data.Add(New Row With {.ItemId = itemDescriptors(i).ItemId, .Value =
valueResults(i).Value.ToString()})
        Next i

        GridView1.DataSource = data
        GridView1.DataBind()
    End Sub
End Class
```

13.1.4.2.5 Installed Examples - Web - UAWebApplication1

The simplest ASP.NET Web application for OPC-UA. Reads and displays a value of a node in an OPC-UA server.

The default page code-behind:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
using OpcLabs.EasyOpc.UA;
using System;

namespace WebApplication1
{
    // ReSharper disable InconsistentNaming
    public partial class _Default : System.Web.UI.Page
    // ReSharper restore InconsistentNaming
    {
        // ReSharper disable InconsistentNaming
        protected void Page_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            var client = new EasyUAClient();
            // ReSharper disable PossibleNullReferenceException
            TextBox1.Text = client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853").ToString();
            // ReSharper restore PossibleNullReferenceException
```

```
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.UA

' ReSharper disable InconsistentNaming
Partial Public Class _Default
    Inherits UI.Page
    ' ReSharper restore InconsistentNaming
    ' ReSharper disable InconsistentNaming
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)

        Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
        ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

        ' ReSharper restore InconsistentNaming
        Dim client = New EasyUAClient()
        ' ReSharper disable PossibleNullReferenceException
        TextBox1.Text = client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853").ToString()
        ' ReSharper restore PossibleNullReferenceException
    End Sub
End Class
```

13.1.4.2.6 Installed Examples - Web - WcfService1

A simple Web service using WCF technology. Provides a GetData method to read a value of an OPC item. Use WcfClient1 project (under Console folder) to test this Web service.

The service class:

C#

```
// WcfService1: A simple Web service using WCF technology. Provides a GetData method to
read a value of an OPC item.
// Use WcfClient1 project (under Console folder) to test this Web service.

using OpcLabs.EasyOpc.DataAccess;

namespace WcfService1
{
    // NOTE: If you change the class name "Service1" here, you must also update the
reference to "Service1" in Web.config and in the associated .svc file.
    public class Service1 : IService1
    {
```

```
        public string GetData()
        {
            var client = new EasyDAClient();
            object value = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Ramp");
            return (value == null) ? "" : value.ToString();
        }
    }
}
```

VB.NET

```
' WcfService1: A simple Web service using WCF technology. Provides a GetData method to
read a value of an OPC item.
' Use WcfClient1 project (under Console folder) to test this Web service.
Imports OpcLabs.EasyOpc.DataAccess

' NOTE: If you change the class name "Service1" here, you must also update the
reference to "Service1" in Web.config and in the associated .svc file.
Public Class Service1
    Implements IService1

    Public Function GetData() As String Implements IService1.GetData
        Dim client = New EasyDAClient()
        Dim value As Object = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Ramp")
        Return If(value Is Nothing, "", value.ToString())
    End Function
End Class
```

Service contract:

C#

```
using System.ServiceModel;

namespace WcfService1
{
    // NOTE: If you change the interface name "IService1" here, you must also update
the reference to "IService1" in Web.config.
    [ServiceContract]
    public interface IService1
    {

        [OperationContract]
        string GetData();
    }
}
```

VB.NET

```
Imports System.ServiceModel
```

```
' NOTE: If you change the interface name "IService1" here, you must also update the
reference to "IService1" in Web.config.
<ServiceContract()> _
Public Interface IService1

    <OperationContract()> _
    Function GetData() As String
End Interface
```

13.1.4.2.7 Installed Examples - Web - WebApplication1

The simplest ASP.NET Web application for OPC “Classic”. Reads and displays an OPC item value.

The default page code-behind:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
using OpcLabs.EasyOpc.DataAccess;
using System;

namespace WebApplication1
{
    // ReSharper disable InconsistentNaming
    public partial class _Default : System.Web.UI.Page
    // ReSharper restore InconsistentNaming
    {
        // ReSharper disable InconsistentNaming
        protected void Page_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            var client = new EasyDAClient();
            object value = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Single");
            TextBox1.Text = (value == null) ? "" : value.ToString();
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.DataAccess

' ReSharper disable InconsistentNaming
Partial Public Class _Default
    ' ReSharper restore InconsistentNaming
    Inherits Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
```

```
        Dim client As New EasyDAClient()
        TextBox1.Text = client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Single")
    End Sub

End Class
```

13.1.4.2.8 Installed Examples - Web - WebService1

A simple Web service using ASMX technology. Provides "Hello World" method to read a value of an OPC item.

Service code-behind:

C#

```
// WebService1: A simple Web service using ASMX technology. Provides "Hello World"
method to read a value of an OPC item.

using System.Web.Services;
using OpcLabs.EasyOpc.DataAccess;

namespace WebService1
{
    /// <summary>
    /// Summary description for Service1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX,
uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class Service1 : WebService
    {

        [WebMethod]
        public string HelloWorld()
        {
            var client = new EasyDAClient();
            object value = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Ramp");
            return (value == null) ? "" : value.ToString();
        }
    }
}
```

VB.NET

```
' WebService1: A simple Web service using ASMX technology. Provides "Hello World"
method to read a value of an OPC item.
Imports System.Web.Services
Imports OpcLabs.EasyOpc.DataAccess

''' <summary>
```

```
''' Summary description for Service1
''' </summary>
<WebService(Namespace:="http://tempuri.org/"),
WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1),
ComponentModel.ToolboxItem(False)> _
Public Class Service1
    Inherits WebService
    ' To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment
the following line.
    ' [System.Web.Script.Services.ScriptService]

    <WebMethod()> _
    Public Function HelloWorld() As String
        Dim client = New EasyDAClient()
        Dim value As Object = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Ramp")
        Return If(value Is Nothing, "", value.ToString())
    End Function
End Class
```

13.1.4.3 Installed Examples - WindowsForms

13.1.4.3.1 Installed Examples - WindowsForms - EasyOpcNetDemo

This is a source of the Demo application for OPC "Classic" that ships with the OPC Data Client.NET product. The application shows most product functions, including the browsing forms, OPC property access, and event-based subscriptions.

The main form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.

using System.Diagnostics;
using EasyOpcNetDemo.Properties;
using JetBrains.Annotations;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using System;
using System.Globalization;
using System.Reflection;
using System.Windows.Forms;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
using OpcLabs.EasyOpc.OperationModel;

[assembly:CLSCompliant(true)]
namespace EasyOpcNetDemo
```

```
{
    public partial class MainForm : Form
    {
        // ReSharper disable once NotNullMemberIsNotInitialized
        public MainForm()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void browseServersButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);

            opcServerDialog1.Location = machineNameTextBox.Text;
            if (opcServerDialog1.ShowDialog(this) == DialogResult.OK)
            {
                Debug.Assert(opcServerDialog1.ServerElement != null);
                serverClassTextBox.Text = opcServerDialog1.ServerElement.ServerClass;
            }
        }

        // ReSharper disable InconsistentNaming
        private void browseItemsButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);

            daItemDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text;
            daItemDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text;
            if (daItemDialog1.ShowDialog() == DialogResult.OK)
            {
                Debug.Assert(daItemDialog1.NodeElement != null);
                itemIdTextBox.Text = daItemDialog1.NodeElement.ItemId;
            }
        }

        // ReSharper disable InconsistentNaming
        private void readItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            DAVtq vtq = null;
            Exception exception = null;
            try
            {
                vtq = easyDAClient1.ReadItem(
                    machineNameTextBox.Text,
                    serverClassTextBox.Text,
                    itemIdTextBox.Text);
            }
            catch (OpcException ex)
```

```
            {
                exception = ex;
            }
            DisplayVtq(vtq);
            DisplayException(exception);
        }

        private void DisplayVtq(DAVtq vtq)
        {
            if (vtq == null)
            {
                valueTextBox.Text = "";
                timestampTextBox.Text = "";
                qualityTextBox.Text = "";
            }
            else
            {
                valueTextBox.Text = vtq.DisplayValue();
                timestampTextBox.Text =
vtq.Timestamp.ToString(CultureInfo.CurrentCulture);
                qualityTextBox.Text = vtq.Quality.ToString();
            }
        }

        private void DisplayException(Exception exception)
        {
            exceptionTextBox.Text = exception == null ? "" :
exception.GetBaseException().Message;
        }

        private bool _isItemSubscribedValue/* = false*/;
        private int _itemHandle/* = 0*/;

        public bool IsItemSubscribed
        {
            get { return _isItemSubscribedValue; }
            set
            {
                _isItemSubscribedValue = value;
                subscribeItemButton.Enabled = !_isItemSubscribedValue;
                changeItemSubscriptionButton.Enabled = _isItemSubscribedValue;
                unsubscribeItemButton.Enabled = _isItemSubscribedValue;
            }
        }

        // ReSharper disable InconsistentNaming
        private void subscribeItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            const VarTypes dataType = VarTypes.Empty;
            // ReSharper disable SuggestUseVarKeywordEvident
            int requestedUpdateRate = (int)requestedUpdateRateNumericUpDown.Value;
            float percentDeadband = (float)percentDeadbandNumericUpDown.Value;
```

```
            // ReSharper restore SuggestUseVarKeywordEvident
            _itemHandle = easyDAClient1.SubscribeItem(
                machineNameTextBox.Text,
                serverClassTextBox.Text,
                itemIdTextBox.Text,
                dataType,
                requestedUpdateRate,
                percentDeadband);
            IsItemSubscribed = true;
        }

        // ReSharper disable InconsistentNaming
        private void changeItemSubscriptionButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            var groupParameters = new DAGroupParameters(
                (int)requestedUpdateRateNumericUpDown.Value,
                (float)percentDeadbandNumericUpDown.Value);
            easyDAClient1.ChangeItemSubscription(_itemHandle, groupParameters);
        }

        // ReSharper disable InconsistentNaming
        private void unsubscribeItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            easyDAClient1.UnsubscribeItem(_itemHandle);
            _itemHandle = 0;
            IsItemSubscribed = false;
        }

        // ReSharper disable InconsistentNaming
        private void easyDAClient1_ItemChanged([NotNull] object sender, [NotNull]
EasyDAItemChangedEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            DisplayVtq(e.Vtq);
            DisplayException(e.Exception);
        }

        // ReSharper disable InconsistentNaming
        private void browseMachinesButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            if (computerBrowserDialog1.ShowDialog() == DialogResult.OK)
               machineNameTextBox.Text = computerBrowserDialog1.SelectedName;
        }

        // ReSharper disable InconsistentNaming
        private void aboutButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            MessageBox.Show(this, Assembly.GetExecutingAssembly().FullName,
                Resources.MainForm_aboutButton_Click_Assembly_Name,
MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }
```

```
        // ReSharper disable InconsistentNaming
        private void browsePropertiesButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            daPropertyDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text;
            daPropertyDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text;
            daPropertyDialog1.NodeDescriptor = itemIdTextBox.Text;
            if (daPropertyDialog1.ShowDialog() == DialogResult.OK)
            {
                Debug.Assert(daPropertyDialog1.PropertyElement != null);
                propertyIdMaskedTextBox.Text =
daPropertyDialog1.PropertyElement.PropertyId.ToString();
            }
        }

        // ReSharper disable InconsistentNaming
        private void getPropertyValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            int propertyId = Convert.ToInt32(propertyIdMaskedTextBox.Text,
CultureInfo.CurrentCulture);
            object value = null;
            Exception exception = null;
            try
            {
                value = easyDAClient1.GetPropertyValue(
                    machineNameTextBox.Text,
                    serverClassTextBox.Text,
                    itemIdTextBox.Text,
                    propertyId);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            propertyValueTextBox.Text = (value == null ? "(null)" :
                String.Format(CultureInfo.CurrentCulture, "{0}", value));
            DisplayException(exception);
        }

        // ReSharper disable InconsistentNaming
        private void writeItemValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            object value = valueToWriteTextBox.Text;
```

```
            Exception exception = null;
            try
            {
                easyDAClient1.WriteItemValue(
                    machineNameTextBox.Text,
                    serverClassTextBox.Text,
                    itemIdTextBox.Text,
                    value);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            DisplayException(exception);
        }

        // ReSharper disable InconsistentNaming
        private void closeButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Close();
        }

        // ReSharper disable InconsistentNaming
        private void MainForm_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {

        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports System.Globalization
Imports OpcLabs.EasyOpc.OperationModel
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

<Assembly:CLSCompliant(True)>
' ReSharper disable CheckNamespace
Namespace EasyOpcNetDemo
    ' ReSharper restore CheckNamespace

    Partial Public Class MainForm
        Inherits Form
        Public Sub New()
            InitializeComponent()
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browseServersButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browseServersButton.Click
            ' ReSharper restore InconsistentNaming
```

```
            opcServerDialog1.Location = machineNameTextBox.Text
            If opcServerDialog1.ShowDialog(Me) = DialogResult.OK Then
                serverClassTextBox.Text = opcServerDialog1.ServerElement.ServerClass
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browseItemsButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browseItemsButton.Click
            ' ReSharper restore InconsistentNaming

            opcDAItemDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text
            opcDAItemDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text
            If opcDAItemDialog1.ShowDialog() = DialogResult.OK Then
                itemIdTextBox.Text = opcDAItemDialog1.NodeElement.ItemId
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub readItemButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles readItemButton.Click
            ' ReSharper restore InconsistentNaming

            Dim vtq As DAVtq = Nothing
            Dim exception As Exception = Nothing
            Try
                vtq = easyDAClient1.ReadItem(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text)
            Catch ex As OpcException
                exception = ex
            End Try
            DisplayVtq(vtq)
            DisplayException(exception)
        End Sub

        Private Sub DisplayVtq(ByVal vtq As DAVtq)
            If vtq Is Nothing Then
                valueTextBox.Text = ""
                timestampTextBox.Text = ""
                qualityTextBox.Text = ""
            Else
                valueTextBox.Text = vtq.DisplayValue()
                timestampTextBox.Text = vtq.Timestamp.ToString()
                qualityTextBox.Text = vtq.Quality.ToString()
            End If
        End Sub

        Private Sub DisplayException(ByVal exception As Exception)
            If CObj(exception) Is CObj(Nothing) Then
                exceptionTextBox.Text = ""
            Else
                exceptionTextBox.Text = exception.GetBaseException().Message
            End If
        End Sub

        Private _isItemSubscribedValue As Boolean ' = false
```

```
        Private _itemHandle As Integer ' = 0

        Public Property IsItemSubscribed() As Boolean
            Get
                Return _isItemSubscribedValue
            End Get
            Set(ByVal value As Boolean)
                _isItemSubscribedValue = value
                subscribeItemButton.Enabled = Not _isItemSubscribedValue
                changeItemSubscriptionButton.Enabled = _isItemSubscribedValue
                unsubscribeItemButton.Enabled = _isItemSubscribedValue
            End Set
        End Property

        ' ReSharper disable InconsistentNaming
        Private Sub subscribeItemButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles subscribeItemButton.Click
            ' ReSharper restore InconsistentNaming

            Const dataType As Short = VarTypes.Empty
            Dim requestedUpdateRate As Integer =
CInt(Fix(requestedUpdateRateNumericUpDown.Value))
            Dim percentDeadband As Single = CSng(percentDeadbandNumericUpDown.Value)
            _itemHandle = easyDAClient1.SubscribeItem(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text, dataType, requestedUpdateRate,
percentDeadband)
            IsItemSubscribed = True
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub changeItemSubscriptionButton_Click(ByVal sender As Object, ByVal e
As EventArgs) Handles changeItemSubscriptionButton.Click
            ' ReSharper restore InconsistentNaming

            Dim groupParameters = New
DAGroupParameters(CInt(Fix(requestedUpdateRateNumericUpDown.Value)),
CSng(percentDeadbandNumericUpDown.Value))
            easyDAClient1.ChangeItemSubscription(_itemHandle, groupParameters)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub unsubscribeItemButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles unsubscribeItemButton.Click
            ' ReSharper restore InconsistentNaming

            easyDAClient1.UnsubscribeItem(_itemHandle)
            _itemHandle = 0
            IsItemSubscribed = False
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub easyDAClient1_ItemChanged(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs) Handles easyDAClient1.ItemChanged
            ' ReSharper restore InconsistentNaming

            DisplayVtq(e.Vtq)
            DisplayException(e.Exception)
```

```
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browseMachinesButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browseMachinesButton.Click
            ' ReSharper restore InconsistentNaming

            If computerBrowserDialog1.ShowDialog() = DialogResult.OK Then
                machineNameTextBox.Text = computerBrowserDialog1.SelectedName
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub aboutButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles aboutButton.Click
            ' ReSharper restore InconsistentNaming

            MessageBox.Show(Me, Reflection.Assembly.GetExecutingAssembly().FullName,
"Assembly Name", MessageBoxButtons.OK, MessageBoxIcon.Information)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browsePropertiesButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browsePropertiesButton.Click
            ' ReSharper restore InconsistentNaming

            opcDAPropertyDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text
            opcDAPropertyDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text
            opcDAPropertyDialog1.NodeDescriptor = itemIdTextBox.Text
            If opcDAPropertyDialog1.ShowDialog() = DialogResult.OK Then
                propertyIdMaskedTextBox.Text =
opcDAPropertyDialog1.PropertyElement.PropertyId.ToString()
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub getPropertyValueButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles getPropertyValueButton.Click
            ' ReSharper restore InconsistentNaming

            Dim propertyId As Integer = Convert.ToInt32(propertyIdMaskedTextBox.Text,
CultureInfo.CurrentCulture)
            Dim value As Object = Nothing
            Dim exception As Exception = Nothing
            Try
                value = easyDAClient1.GetPropertyValue(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text, propertyId)
            Catch ex As OpcException
                exception = ex
            End Try
            propertyValueTextBox.Text = (If(value Is Nothing, "(null)",
String.Format(CultureInfo.CurrentCulture, "{0}", value)))
            DisplayException(exception)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub writeItemValueButton_Click(ByVal sender As Object, ByVal e As
```

```
EventArgs) Handles writeItemValueButton.Click
            ' ReSharper restore InconsistentNaming

            Dim value As Object = valueToWriteTextBox.Text
            Dim exception As Exception = Nothing
            Try
                easyDAClient1.WriteItemValue(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text, value)
            Catch ex As OpcException
                exception = ex
            End Try
            DisplayException(exception)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub closeButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles closeButton.Click
            ' ReSharper restore InconsistentNaming

            Close()
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub MainForm_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
            ' ReSharper restore InconsistentNaming

        End Sub
    End Class
End Namespace
```

13.1.4.3.2 Installed Examples - WindowsForms - EasyOpcNetDemoXml

This is a source of the Demo application for OPC "Classic" (OPC XML-enabled) that ships with the OPC Data Client.NET product. The application shows most product functions, including the browsing forms, OPC property access, and event-based subscriptions. The defaults are pre-filled for OPC XML-DA demo server, but the application is written in such a way that it can handle COM servers as well.

The main form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.

using System.Diagnostics;
using EasyOpcNetDemoXml.Properties;
using JetBrains.Annotations;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
```

```
using System;
using System.Globalization;
using System.Reflection;
using System.Windows.Forms;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
using OpcLabs.EasyOpc.OperationModel;

[assembly:CLSCompliant(true)]
namespace EasyOpcNetDemoXml
{
    public partial class MainForm : Form
    {
        // ReSharper disable once NotNullMemberIsNotInitialized
        public MainForm()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void browseItemsButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);
            Debug.Assert(daItemDialog1.NodeDescriptor != null);

            daItemDialog1.ServerDescriptor.UrlString = serverUrlTextBox.Text;
            daItemDialog1.NodeDescriptor.ItemId = itemIdTextBox.Text;
            daItemDialog1.NodeDescriptor.NodePath = nodePathTextBox.Text;
            if (daItemDialog1.ShowDialog() == DialogResult.OK)
            {
                Debug.Assert(daItemDialog1.NodeDescriptor != null);
                itemIdTextBox.Text = daItemDialog1.NodeDescriptor.ItemId;
                nodePathTextBox.Text = daItemDialog1.NodeDescriptor.NodePath;
            }
        }

        // ReSharper disable InconsistentNaming
        private void readItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            DAVtq vtq = null;
            Exception exception = null;
            try
            {
                vtq = easyDAClient1.ReadItem(
                    serverUrlTextBox.Text,
                    new DAItemDescriptor(itemIdTextBox.Text) { NodePath =
nodePathTextBox.Text });
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            DisplayVtq(vtq);
```

```
            DisplayException(exception);
        }

        private void DisplayVtq(DAVtq vtq)
        {
            if (vtq == null)
            {
                valueTextBox.Text = "";
                timestampTextBox.Text = "";
                qualityTextBox.Text = "";
            }
            else
            {
                valueTextBox.Text = vtq.DisplayValue();
                timestampTextBox.Text =
vtq.Timestamp.ToString(CultureInfo.CurrentCulture);
                qualityTextBox.Text = vtq.Quality.ToString();
            }
        }

        private void DisplayException(Exception exception)
        {
            exceptionTextBox.Text = exception == null ? "" :
exception.GetBaseException().Message;
        }

        private bool _isItemSubscribedValue/* = false*/;
        private int _itemHandle/* = 0*/;

        public bool IsItemSubscribed
        {
            get { return _isItemSubscribedValue; }
            set
            {
                _isItemSubscribedValue = value;
                subscribeItemButton.Enabled = !_isItemSubscribedValue;
                changeItemSubscriptionButton.Enabled = _isItemSubscribedValue;
                unsubscribeItemButton.Enabled = _isItemSubscribedValue;
            }
        }

        // ReSharper disable InconsistentNaming
        private void subscribeItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            const VarTypes dataType = VarTypes.Empty;
            // ReSharper disable SuggestUseVarKeywordEvident
            int requestedUpdateRate = (int)requestedUpdateRateNumericUpDown.Value;
            float percentDeadband = (float)percentDeadbandNumericUpDown.Value;
            // ReSharper restore SuggestUseVarKeywordEvident
            _itemHandle = easyDAClient1.SubscribeItem(
                serverUrlTextBox.Text,
                new DAItemDescriptor(itemIdTextBox.Text, dataType) { NodePath =
nodePathTextBox.Text },
```

```
                requestedUpdateRate,
                percentDeadband);
            IsItemSubscribed = true;
        }

        // ReSharper disable InconsistentNaming
        private void changeItemSubscriptionButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            var groupParameters = new DAGroupParameters(
                (int)requestedUpdateRateNumericUpDown.Value,
                (float)percentDeadbandNumericUpDown.Value);
            easyDAClient1.ChangeItemSubscription(_itemHandle, groupParameters);
        }

        // ReSharper disable InconsistentNaming
        private void unsubscribeItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            easyDAClient1.UnsubscribeItem(_itemHandle);
            _itemHandle = 0;
            IsItemSubscribed = false;
        }

        // ReSharper disable InconsistentNaming
        private void easyDAClient1_ItemChanged([NotNull] object sender, [NotNull]
EasyDAItemChangedEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            DisplayVtq(e.Vtq);
            DisplayException(e.Exception);
        }

        // ReSharper disable InconsistentNaming
        private void aboutButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            MessageBox.Show(this, Assembly.GetExecutingAssembly().FullName,
                Resources.MainForm_aboutButton_Click_Assembly_Name,
MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }

        // ReSharper disable InconsistentNaming
        private void browsePropertiesButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            daPropertyDialog1.ServerDescriptor.UrlString = serverUrlTextBox.Text;
            daPropertyDialog1.NodeDescriptor.ItemId = itemIdTextBox.Text;
            daPropertyDialog1.NodeDescriptor.NodePath = nodePathTextBox.Text;
            if (daPropertyDialog1.ShowDialog() == DialogResult.OK)
            {
                Debug.Assert(daPropertyDialog1.PropertyElement != null);
                propertyTextBox.Text = daPropertyDialog1.PropertyDescriptor.ToString();
```

```
            }
        }

        // ReSharper disable InconsistentNaming
        private void getPropertyValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);
            Debug.Assert(propertyTextBox.Text != null);

            object value = null;
            Exception exception = null;
            try
            {
                value = easyDAClient1.GetPropertyValue(
                    serverUrlTextBox.Text,
                    new DANodeDescriptor(itemIdTextBox.Text) { NodePath =
nodePathTextBox.Text },
                    propertyTextBox.Text);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            propertyValueTextBox.Text = (value == null ? "(null)" :
                String.Format(CultureInfo.CurrentCulture, "{0}", value));
            DisplayException(exception);
        }

        // ReSharper disable InconsistentNaming
        private void writeItemValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            object value = valueToWriteTextBox.Text;
            Exception exception = null;
            try
            {
                easyDAClient1.WriteItemValue(
                    serverUrlTextBox.Text,
                    new DAItemDescriptor(itemIdTextBox.Text) { NodePath =
nodePathTextBox.Text },
                    value);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            DisplayException(exception);
        }

        // ReSharper disable InconsistentNaming
        private void closeButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
```

```
        {
            Close();
        }

        // ReSharper disable InconsistentNaming
        private void MainForm_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {

        }

        // ReSharper disable InconsistentNaming
        private void browseComputersAndServersButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(serverUrlTextBox.Text != null);

            opcComputerAndServerDialog1.ServerDescriptor.UrlString =
serverUrlTextBox.Text;
            if (opcComputerAndServerDialog1.ShowDialog(this) == DialogResult.OK)
                serverUrlTextBox.Text =
opcComputerAndServerDialog1.ServerDescriptor.UrlString;
        }
    }
}
```

13.1.4.3.3 Installed Examples - WindowsForms - EasyOpcUADemo

This is a source of the Demo application for OPC-UA that ships with the OPC Data Client-UA product. The application shows the most important product functions, including event-based subscriptions.

The main form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
using System;
using System.Globalization;
using System.Reflection;
using System.Windows.Forms;
using EasyOpcUADemo.Properties;
using JetBrains.Annotations;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

[assembly:CLSCompliant(true)]
namespace EasyOpcUADemo
{
    public partial class MainForm : Form
    {
```

```
        // ReSharper disable once NotNullMemberIsNotInitialized
        public MainForm()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void readButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            UAAttributeData attributeData = null;
            Exception exception = null;
            try
            {
                // ReSharper disable AssignNullToNotNullAttribute
                attributeData = easyUAClient1.Read(serverUriTextBox.Text,
nodeIdTextBox.Text);
                // ReSharper restore AssignNullToNotNullAttribute
            }
            catch (UAException ex)
            {
                exception = ex;
            }
            DisplayAttributeData(attributeData);
            DisplayException(exception);
        }

        private void DisplayAttributeData(UAAttributeData attributeData)
        {
            if (attributeData == null)
            {
                valueTextBox.Text = "";
                statusTextBox.Text = "";
                sourceTimestampTextBox.Text = "";
                serverTimestampTextBox.Text = "";
            }
            else
            {
                valueTextBox.Text = attributeData.DisplayValue();
                statusTextBox.Text = attributeData.StatusCode.ToString();
                sourceTimestampTextBox.Text =
attributeData.SourceTimestamp.ToString(CultureInfo.CurrentCulture);
                serverTimestampTextBox.Text =
attributeData.ServerTimestamp.ToString(CultureInfo.CurrentCulture);
            }
        }

        private void DisplayException(Exception exception)
        {
            exceptionTextBox.Text = exception == null ? "" :
exception.GetBaseException().Message;
        }

        private bool _isSubscribed/* = false*/;
        private int _handle/* = 0*/;

        public bool IsSubscribed
```

```
        {
            get { return _isSubscribed; }
            set
            {
                _isSubscribed = value;
                subscribeMonitoredItemButton.Enabled = !_isSubscribed;
                changeMonitoredItemSubscriptionButton.Enabled = _isSubscribed;
                unsubscribeMonitoredItemButton.Enabled = _isSubscribed;
            }
        }

        // ReSharper disable InconsistentNaming
        private void subscribeButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            // ReSharper disable AssignNullToNotNullAttribute
            _handle = easyUAClient1.SubscribeDataChange(serverUriTextBox.Text,
nodeIdTextBox.Text,

(int)samplingIntervalNumericUpDown.Value);
            // ReSharper restore AssignNullToNotNullAttribute
            IsSubscribed = true;
        }

        // ReSharper disable InconsistentNaming
        private void changeSubscriptionButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            easyUAClient1.ChangeMonitoredItemSubscription(_handle,
(int)samplingIntervalNumericUpDown.Value);
        }

        // ReSharper disable InconsistentNaming
        private void unsubscribeButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Unsubscribe();
        }

        private void Unsubscribe()
        {
            easyUAClient1.UnsubscribeMonitoredItem(_handle);
            _handle = 0;
            IsSubscribed = false;
        }

        // ReSharper disable InconsistentNaming
        private void easyUAClient1_DataChangeNotification([NotNull] object sender,
[NotNull] EasyUADataChangeNotificationEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            DisplayAttributeData((e.Exception == null) ? e.AttributeData : null);
            DisplayException(e.Exception);
        }

        // ReSharper disable InconsistentNaming
        private void aboutButton_Click(object sender, EventArgs e)
```

```
        // ReSharper restore InconsistentNaming
        {
            MessageBox.Show(this, Assembly.GetExecutingAssembly().FullName,
Resources.MainForm_AssemblyName,
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        // ReSharper disable InconsistentNaming
        private void closeButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            if (IsSubscribed)
                Unsubscribe();
            Close();
        }

        // ReSharper disable InconsistentNaming
        private void MainForm_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {

        }

        // ReSharper disable InconsistentNaming
        private void discoverServersButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            // ReSharper disable AssignNullToNotNullAttribute
            uaHostAndEndpointDialog1.EndpointDescriptor = serverUriTextBox.Text;
            // ReSharper restore AssignNullToNotNullAttribute
            if (uaHostAndEndpointDialog1.ShowDialog() == DialogResult.OK)
                serverUriTextBox.Text =
uaHostAndEndpointDialog1.EndpointDescriptor.UrlString;
        }

        // ReSharper disable InconsistentNaming
        private void browseDataButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            uaDataDialog1.EndpointDescriptor = serverUriTextBox.Text;
            if (uaDataDialog1.ShowDialog() == DialogResult.OK)
                // ReSharper disable PossibleNullReferenceException
                nodeIdTextBox.Text = uaDataDialog1.NodeDescriptor.NodeId;
                // ReSharper restore PossibleNullReferenceException
        }

        // ReSharper disable InconsistentNaming
        private void writeValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Exception exception = null;
            try
            {
                // ReSharper disable AssignNullToNotNullAttribute
                easyUAClient1.WriteValue(serverUriTextBox.Text, nodeIdTextBox.Text,
valueToWriteTextBox.Text);
                // ReSharper restore AssignNullToNotNullAttribute
```

```
            }
            catch (UAException ex)
            {
                exception = ex;
            }
            DisplayException(exception);
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

<Assembly:CLSCompliant(True)>

Partial Public Class MainForm
    Inherits Form
    Public Sub New()
        InitializeComponent()
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub readButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
readButton.Click
        ' ReSharper restore InconsistentNaming

        Dim attributeData As UAAttributeData = Nothing
        Dim exception As Exception = Nothing
        Try
            attributeData = easyUAClient1.Read(serverUriTextBox.Text,
nodeIdTextBox.Text)
        Catch ex As UAException
            exception = ex
        End Try
        DisplayAttributeData(attributeData)
        DisplayException(exception)
    End Sub

    Private Sub DisplayAttributeData(ByVal attributeData As UAAttributeData)
        If attributeData Is Nothing Then
            valueTextBox.Text = ""
            statusTextBox.Text = ""
            sourceTimestampTextBox.Text = ""
            serverTimestampTextBox.Text = ""
        Else
            valueTextBox.Text = attributeData.DisplayValue()
            statusTextBox.Text = attributeData.StatusCode.ToString()
            sourceTimestampTextBox.Text = attributeData.SourceTimestamp.ToString()
            serverTimestampTextBox.Text = attributeData.ServerTimestamp.ToString()
        End If
    End Sub

    Private Sub DisplayException(ByVal exception As Exception)
```

```
        exceptionTextBox.Text = If(exception Is Nothing, "",
exception.GetBaseException().Message)
    End Sub

    Private _isSubscribed As Boolean ' = false
    Private _handle As Integer ' = 0

    Public Property IsSubscribed() As Boolean
        Get
            Return _isSubscribed
        End Get
        Set(ByVal value As Boolean)
            _isSubscribed = value
            subscribeMonitoredItemButton.Enabled = Not _isSubscribed
            changeMonitoredItemSubscriptionButton.Enabled = _isSubscribed
            unsubscribeMonitoredItemButton.Enabled = _isSubscribed
        End Set
    End Property

    ' ReSharper disable InconsistentNaming
    Private Sub subscribeButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles subscribeMonitoredItemButton.Click
        ' ReSharper restore InconsistentNaming

        _handle = easyUAClient1.SubscribeDataChange(serverUriTextBox.Text,
nodeIdTextBox.Text, CInt(Fix(samplingIntervalNumericUpDown.Value)))
        IsSubscribed = True
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub changeSubscriptionButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles changeMonitoredItemSubscriptionButton.Click
        ' ReSharper restore InconsistentNaming
        easyUAClient1.ChangeMonitoredItemSubscription(_handle,
CInt(Fix(samplingIntervalNumericUpDown.Value)))
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub unsubscribeButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles unsubscribeMonitoredItemButton.Click
        ' ReSharper restore InconsistentNaming
        Unsubscribe()
    End Sub

    Private Sub Unsubscribe()
        easyUAClient1.UnsubscribeMonitoredItem(_handle)
        _handle = 0
        IsSubscribed = False
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub easyUAClient1_DataChangeNotification(ByVal sender As Object, ByVal e As
EasyUADataChangeNotificationEventArgs) Handles easyUAClient1.DataChangeNotification
        ' ReSharper restore InconsistentNaming
        DisplayAttributeData(If(e.Exception Is Nothing, e.AttributeData, Nothing))
        DisplayException(e.Exception)
    End Sub
```

```
    ' ReSharper disable InconsistentNaming
    Private Sub aboutButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
aboutButton.Click
        ' ReSharper restore InconsistentNaming
        MessageBox.Show(Me, Reflection.Assembly.GetExecutingAssembly().FullName,
My.Resources.MainForm_AssemblyName, MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub closeButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
closeButton.Click
        ' ReSharper restore InconsistentNaming
        If IsSubscribed Then
            Unsubscribe()
        End If
        Close()
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub MainForm_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
        ' ReSharper restore InconsistentNaming

    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub writeValueButton_Click(ByVal sender As System.Object, ByVal e As
EventArgs) Handles writeValueButton.Click
        ' ReSharper restore InconsistentNaming

        Dim exception As Exception = Nothing
        Try
            easyUAClient1.WriteValue(serverUriTextBox.Text, nodeIdTextBox.Text,
valueToWriteTextBox.Text)
        Catch ex As UAException
            exception = ex
        End Try
        DisplayException(exception)
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub discoverServersButton_Click(ByVal sender As System.Object, ByVal e As
EventArgs) Handles discoverServersButton.Click
        ' ReSharper restore InconsistentNaming

        uaHostAndEndpointDialog1.EndpointDescriptor = serverUriTextBox.Text
        If uaHostAndEndpointDialog1.ShowDialog() = DialogResult.OK Then
            serverUriTextBox.Text =
uaHostAndEndpointDialog1.EndpointDescriptor.UrlString
        End If
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub browseDataButton_Click(ByVal sender As System.Object, ByVal e As
EventArgs) Handles browseDataButton.Click
        ' ReSharper restore InconsistentNaming
```

```
        uaDataDialog1.EndpointDescriptor = serverUriTextBox.Text
        If uaDataDialog1.ShowDialog() = DialogResult.OK Then
            nodeIdTextBox.Text = uaDataDialog1.NodeDescriptor.NodeId
        End If
    End Sub
End Class
```

13.1.4.3.4 Installed Examples - WindowsForms - HmiScreen

Windows Forms application that shows how to use implement an HMI screen by storing OPC Item IDs in the Tag property of screen controls, and animate the controls by subscribing to all items at once. Also shows a possibility how to write to an OPC item form the screen.

The form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.

using System.Diagnostics;
using EasyOpcNetDemo.Properties;
using JetBrains.Annotations;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using System;
using System.Globalization;
using System.Reflection;
using System.Windows.Forms;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
using OpcLabs.EasyOpc.OperationModel;

[assembly:CLSCompliant(true)]
namespace EasyOpcNetDemo
{
    public partial class MainForm : Form
    {
        // ReSharper disable once NotNullMemberIsNotInitialized
        public MainForm()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void browseServersButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);

            opcServerDialog1.Location = machineNameTextBox.Text;
            if (opcServerDialog1.ShowDialog(this) == DialogResult.OK)
            {
```

```
            Debug.Assert(opcServerDialog1.ServerElement != null);
            serverClassTextBox.Text = opcServerDialog1.ServerElement.ServerClass;
        }
    }

    // ReSharper disable InconsistentNaming
    private void browseItemsButton_Click(object sender, EventArgs e)
    // ReSharper restore InconsistentNaming
    {
        Debug.Assert(machineNameTextBox.Text != null);
        Debug.Assert(serverClassTextBox.Text != null);

        daItemDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text;
        daItemDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text;
        if (daItemDialog1.ShowDialog() == DialogResult.OK)
        {
            Debug.Assert(daItemDialog1.NodeElement != null);
            itemIdTextBox.Text = daItemDialog1.NodeElement.ItemId;
        }
    }

    // ReSharper disable InconsistentNaming
    private void readItemButton_Click(object sender, EventArgs e)
    // ReSharper restore InconsistentNaming
    {
        Debug.Assert(machineNameTextBox.Text != null);
        Debug.Assert(serverClassTextBox.Text != null);
        Debug.Assert(itemIdTextBox.Text != null);

        DAVtq vtq = null;
        Exception exception = null;
        try
        {
            vtq = easyDAClient1.ReadItem(
                machineNameTextBox.Text,
                serverClassTextBox.Text,
                itemIdTextBox.Text);
        }
        catch (OpcException ex)
        {
            exception = ex;
        }
        DisplayVtq(vtq);
        DisplayException(exception);
    }

    private void DisplayVtq(DAVtq vtq)
    {
        if (vtq == null)
        {
            valueTextBox.Text = "";
            timestampTextBox.Text = "";
            qualityTextBox.Text = "";
        }
        else
        {
            valueTextBox.Text = vtq.DisplayValue();
```

```
                timestampTextBox.Text =
vtq.Timestamp.ToString(CultureInfo.CurrentCulture);
                qualityTextBox.Text = vtq.Quality.ToString();
            }
        }

        private void DisplayException(Exception exception)
        {
            exceptionTextBox.Text = exception == null ? "" :
exception.GetBaseException().Message;
        }

        private bool _isItemSubscribedValue/* = false*/;
        private int _itemHandle/* = 0*/;

        public bool IsItemSubscribed
        {
            get { return _isItemSubscribedValue; }
            set
            {
                _isItemSubscribedValue = value;
                subscribeItemButton.Enabled = !_isItemSubscribedValue;
                changeItemSubscriptionButton.Enabled = _isItemSubscribedValue;
                unsubscribeItemButton.Enabled = _isItemSubscribedValue;
            }
        }

        // ReSharper disable InconsistentNaming
        private void subscribeItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            const VarTypes dataType = VarTypes.Empty;
            // ReSharper disable SuggestUseVarKeywordEvident
            int requestedUpdateRate = (int)requestedUpdateRateNumericUpDown.Value;
            float percentDeadband = (float)percentDeadbandNumericUpDown.Value;
            // ReSharper restore SuggestUseVarKeywordEvident
            _itemHandle = easyDAClient1.SubscribeItem(
                machineNameTextBox.Text,
                serverClassTextBox.Text,
                itemIdTextBox.Text,
                dataType,
                requestedUpdateRate,
                percentDeadband);
            IsItemSubscribed = true;
        }

        // ReSharper disable InconsistentNaming
        private void changeItemSubscriptionButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            var groupParameters = new DAGroupParameters(
                (int)requestedUpdateRateNumericUpDown.Value,
                (float)percentDeadbandNumericUpDown.Value);
```

```
            easyDAClient1.ChangeItemSubscription(_itemHandle, groupParameters);
        }

        // ReSharper disable InconsistentNaming
        private void unsubscribeItemButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            easyDAClient1.UnsubscribeItem(_itemHandle);
            _itemHandle = 0;
            IsItemSubscribed = false;
        }

        // ReSharper disable InconsistentNaming
        private void easyDAClient1_ItemChanged([NotNull] object sender, [NotNull]
EasyDAItemChangedEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            DisplayVtq(e.Vtq);
            DisplayException(e.Exception);
        }

        // ReSharper disable InconsistentNaming
        private void browseMachinesButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            if (computerBrowserDialog1.ShowDialog() == DialogResult.OK)
               machineNameTextBox.Text = computerBrowserDialog1.SelectedName;
        }

        // ReSharper disable InconsistentNaming
        private void aboutButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            MessageBox.Show(this, Assembly.GetExecutingAssembly().FullName,
                Resources.MainForm_aboutButton_Click_Assembly_Name,
MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }

        // ReSharper disable InconsistentNaming
        private void browsePropertiesButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            daPropertyDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text;
            daPropertyDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text;
            daPropertyDialog1.NodeDescriptor = itemIdTextBox.Text;
            if (daPropertyDialog1.ShowDialog() == DialogResult.OK)
            {
                Debug.Assert(daPropertyDialog1.PropertyElement != null);
                propertyIdMaskedTextBox.Text =
daPropertyDialog1.PropertyElement.PropertyId.ToString();
            }
        }
```

```
        // ReSharper disable InconsistentNaming
        private void getPropertyValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            int propertyId = Convert.ToInt32(propertyIdMaskedTextBox.Text,
CultureInfo.CurrentCulture);
            object value = null;
            Exception exception = null;
            try
            {
                value = easyDAClient1.GetPropertyValue(
                    machineNameTextBox.Text,
                    serverClassTextBox.Text,
                    itemIdTextBox.Text,
                    propertyId);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            propertyValueTextBox.Text = (value == null ? "(null)" :
                String.Format(CultureInfo.CurrentCulture, "{0}", value));
            DisplayException(exception);
        }

        // ReSharper disable InconsistentNaming
        private void writeItemValueButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Debug.Assert(machineNameTextBox.Text != null);
            Debug.Assert(serverClassTextBox.Text != null);
            Debug.Assert(itemIdTextBox.Text != null);

            object value = valueToWriteTextBox.Text;
            Exception exception = null;
            try
            {
                easyDAClient1.WriteItemValue(
                    machineNameTextBox.Text,
                    serverClassTextBox.Text,
                    itemIdTextBox.Text,
                    value);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            DisplayException(exception);
        }

        // ReSharper disable InconsistentNaming
        private void closeButton_Click(object sender, EventArgs e)
```

```
        // ReSharper restore InconsistentNaming
        {
            Close();
        }

        // ReSharper disable InconsistentNaming
        private void MainForm_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {

        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports System.Globalization
Imports OpcLabs.EasyOpc.OperationModel
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

<Assembly:CLSCompliant(True)>
' ReSharper disable CheckNamespace
Namespace EasyOpcNetDemo
    ' ReSharper restore CheckNamespace

    Partial Public Class MainForm
        Inherits Form
        Public Sub New()
            InitializeComponent()
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browseServersButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browseServersButton.Click
            ' ReSharper restore InconsistentNaming

            opcServerDialog1.Location = machineNameTextBox.Text
            If opcServerDialog1.ShowDialog(Me) = DialogResult.OK Then
                serverClassTextBox.Text = opcServerDialog1.ServerElement.ServerClass
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browseItemsButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browseItemsButton.Click
            ' ReSharper restore InconsistentNaming

            opcDAItemDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text
            opcDAItemDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text
            If opcDAItemDialog1.ShowDialog() = DialogResult.OK Then
                itemIdTextBox.Text = opcDAItemDialog1.NodeElement.ItemId
            End If
        End Sub
```

```
        ' ReSharper disable InconsistentNaming
        Private Sub readItemButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles readItemButton.Click
            ' ReSharper restore InconsistentNaming

            Dim vtq As DAVtq = Nothing
            Dim exception As Exception = Nothing
            Try
                vtq = easyDAClient1.ReadItem(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text)
            Catch ex As OpcException
                exception = ex
            End Try
            DisplayVtq(vtq)
            DisplayException(exception)
        End Sub

        Private Sub DisplayVtq(ByVal vtq As DAVtq)
            If vtq Is Nothing Then
                valueTextBox.Text = ""
                timestampTextBox.Text = ""
                qualityTextBox.Text = ""
            Else
                valueTextBox.Text = vtq.DisplayValue()
                timestampTextBox.Text = vtq.Timestamp.ToString()
                qualityTextBox.Text = vtq.Quality.ToString()
            End If
        End Sub

        Private Sub DisplayException(ByVal exception As Exception)
            If CObj(exception) Is CObj(Nothing) Then
                exceptionTextBox.Text = ""
            Else
                exceptionTextBox.Text = exception.GetBaseException().Message
            End If
        End Sub

        Private _isItemSubscribedValue As Boolean ' = false
        Private _itemHandle As Integer ' = 0

        Public Property IsItemSubscribed() As Boolean
            Get
                Return _isItemSubscribedValue
            End Get
            Set(ByVal value As Boolean)
                _isItemSubscribedValue = value
                subscribeItemButton.Enabled = Not _isItemSubscribedValue
                changeItemSubscriptionButton.Enabled = _isItemSubscribedValue
                unsubscribeItemButton.Enabled = _isItemSubscribedValue
            End Set
        End Property

        ' ReSharper disable InconsistentNaming
        Private Sub subscribeItemButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles subscribeItemButton.Click
            ' ReSharper restore InconsistentNaming
```

```
            Const dataType As Short = VarTypes.Empty
            Dim requestedUpdateRate As Integer =
CInt(Fix(requestedUpdateRateNumericUpDown.Value))
            Dim percentDeadband As Single = CSng(percentDeadbandNumericUpDown.Value)
            _itemHandle = easyDAClient1.SubscribeItem(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text, dataType, requestedUpdateRate,
percentDeadband)
            IsItemSubscribed = True
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub changeItemSubscriptionButton_Click(ByVal sender As Object, ByVal e
As EventArgs) Handles changeItemSubscriptionButton.Click
            ' ReSharper restore InconsistentNaming

            Dim groupParameters = New
DAGroupParameters(CInt(Fix(requestedUpdateRateNumericUpDown.Value)),
CSng(percentDeadbandNumericUpDown.Value))
            easyDAClient1.ChangeItemSubscription(_itemHandle, groupParameters)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub unsubscribeItemButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles unsubscribeItemButton.Click
            ' ReSharper restore InconsistentNaming

            easyDAClient1.UnsubscribeItem(_itemHandle)
            _itemHandle = 0
            IsItemSubscribed = False
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub easyDAClient1_ItemChanged(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs) Handles easyDAClient1.ItemChanged
            ' ReSharper restore InconsistentNaming

            DisplayVtq(e.Vtq)
            DisplayException(e.Exception)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browseMachinesButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browseMachinesButton.Click
            ' ReSharper restore InconsistentNaming

            If computerBrowserDialog1.ShowDialog() = DialogResult.OK Then
                machineNameTextBox.Text = computerBrowserDialog1.SelectedName
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub aboutButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles aboutButton.Click
            ' ReSharper restore InconsistentNaming

            MessageBox.Show(Me, Reflection.Assembly.GetExecutingAssembly().FullName,
```

```
"Assembly Name", MessageBoxButtons.OK, MessageBoxIcon.Information)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub browsePropertiesButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles browsePropertiesButton.Click
            ' ReSharper restore InconsistentNaming

            opcDAPropertyDialog1.ServerDescriptor.MachineName = machineNameTextBox.Text
            opcDAPropertyDialog1.ServerDescriptor.ServerClass = serverClassTextBox.Text
            opcDAPropertyDialog1.NodeDescriptor = itemIdTextBox.Text
            If opcDAPropertyDialog1.ShowDialog() = DialogResult.OK Then
                propertyIdMaskedTextBox.Text =
opcDAPropertyDialog1.PropertyElement.PropertyId.ToString()
            End If
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub getPropertyValueButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles getPropertyValueButton.Click
            ' ReSharper restore InconsistentNaming

            Dim propertyId As Integer = Convert.ToInt32(propertyIdMaskedTextBox.Text,
CultureInfo.CurrentCulture)
            Dim value As Object = Nothing
            Dim exception As Exception = Nothing
            Try
                value = easyDAClient1.GetPropertyValue(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text, propertyId)
            Catch ex As OpcException
                exception = ex
            End Try
            propertyValueTextBox.Text = (If(value Is Nothing, "(null)",
String.Format(CultureInfo.CurrentCulture, "{0}", value)))
            DisplayException(exception)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub writeItemValueButton_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles writeItemValueButton.Click
            ' ReSharper restore InconsistentNaming

            Dim value As Object = valueToWriteTextBox.Text
            Dim exception As Exception = Nothing
            Try
                easyDAClient1.WriteItemValue(machineNameTextBox.Text,
serverClassTextBox.Text, itemIdTextBox.Text, value)
            Catch ex As OpcException
                exception = ex
            End Try
            DisplayException(exception)
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub closeButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles closeButton.Click
            ' ReSharper restore InconsistentNaming
```

```
            Close()
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub MainForm_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
            ' ReSharper restore InconsistentNaming

        End Sub
    End Class
End Namespace
```

13.1.4.3.5 Installed Examples - WindowsForms - OvenControl

Monitors sensors in an industrial oven, indicates level alarms by changing colors, allows the user to change a setpoint, and logs the values into a CSV file.

The form:

C#

```
using System;
using System.Diagnostics;
using System.Drawing;
using System.Globalization;
using System.Windows.Forms;
using JetBrains.Annotations;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
using OpcLabs.EasyOpc.OperationModel;

namespace OvenControl
{
    public partial class Form1 : Form
    {
        private const string MachineName = "";
        private const string ServerClass = "SWToolbox.TOPServer.V5";    // or
"Kepware.KEPServerEX.V5"

        private DAVtq _fanPower;
        private DAVtq _heaterPower;
        private DAVtq _ovenTemperature;
        private DAVtq _heaterTemperature;
        private DAVtq _fanSpeed;
        private DAVtq _temperatureSetpoint;


        // ReSharper disable once NotNullMemberIsNotInitialized
        public Form1()
        {
            InitializeComponent();
```

```
}

private Color _defaultBackColor = Color.White;


// ReSharper disable InconsistentNaming
private void Form1_Load(object sender, EventArgs e)
// ReSharper restore InconsistentNaming
{
    SetDefaults();
    _defaultBackColor = txbOvenTemperature.BackColor;
}

private void SetDefaults()
{
    nudUpdateRate.Value = 10;
}

// ReSharper disable InconsistentNaming
private void btnStart_Click(object sender, EventArgs e)
// ReSharper restore InconsistentNaming
{
    btnStart.Enabled = false;
    timer1.Interval = Decimal.ToInt32(nudUpdateRate.Value * 1000);
    timer1.Start();
    UpdateValuesAndLog();
    btnStop.Enabled = true;
}

// ReSharper disable InconsistentNaming
private void btnStop_Click(object sender, EventArgs e)
// ReSharper restore InconsistentNaming
{
    btnStop.Enabled = false;
    timer1.Stop();
    btnStart.Enabled = true;
}

// ReSharper disable InconsistentNaming
private void timer1_Tick(object sender, EventArgs e)
// ReSharper restore InconsistentNaming
{
    UpdateValuesAndLog();
}

private static Int32? ConvertValueToInt32(DAVtq vtq)
{
    if (vtq == null) return null;
    if (!vtq.HasValue) return null;
    return vtq.Value as Int32?;
}

private static string GetTextVtq(DAVtq vtq)
{
    if ((vtq == null) || (!vtq.HasValue))
    {
        return "???";
```

```
        }
        return vtq.DisplayValue();
    }

    private void DisplayVtq(
        DAVtq vtq,
        // ReSharper disable once SuggestBaseTypeForParameter
        [NotNull] TextBox txb,
        int colorTestId)
    {
        if ((vtq == null) || (!vtq.HasValue))
        {
            // ReSharper disable LocalizableElement
            txb.Text = "???";
            // ReSharper restore LocalizableElement
            txb.BackColor = Color.Magenta;
        }
        else
        {
            txb.Text = vtq.DisplayValue();

            Int32? value;
            Color backColor = _defaultBackColor;
            switch (colorTestId)
            {
                case 0:
                    txb.BackColor = _defaultBackColor;
                    break;

                case 1: // oven temperature
                    value = ConvertValueToInt32(vtq);
                    if (value.HasValue)
                    {
                        Int32? range = ConvertValueToInt32(_temperatureSetpoint);
                        if (range.HasValue)
                        {
                            if (value <= (range - 5)) backColor = Color.Blue;
                            else
                                if (value >= (range + 5)) backColor = Color.Red;
                        }
                    }
                    txb.BackColor = backColor;
                    break;

                case 2: // oven temperature
                    value = ConvertValueToInt32(vtq);
                    // ReSharper disable once UseNullPropagation
                    if (value.HasValue)
                    {
                        if (value < 200) backColor = Color.Red;
                    }
                    txb.BackColor = backColor;
                    break;
            }
        }
    }
```

```
        private DAVtq ReadVtq([NotNull] string itemId)
        {
            DAVtq vtq;
            Exception exception = null;
            try
            {
                vtq = easyDAClient1.ReadItem(MachineName, ServerClass, itemId);
            }
            catch (OpcException ex)
            {
                exception = ex;
                vtq = null;
            }
            DisplayException(exception);
            return vtq;
        }

        [CanBeNull]
        private DAVtqResult[] ReadMultipleVtq([NotNull] string[] itemIds)
        {
            DAVtqResult[] results;
            Exception exception = null;
            var itemDescriptors = new DAItemDescriptor[itemIds.Length];
            for (int i = 0; i < itemIds.Length; i++)
            {
                Debug.Assert(itemIds[i] != null);
                itemDescriptors[i] = new DAItemDescriptor(itemIds[i]);
            }
            try
            {
                results = easyDAClient1.ReadMultipleItems(new
ServerDescriptor(MachineName, ServerClass),
                    itemDescriptors);
            }
            catch (OpcException ex)
            {
                exception = ex;
                results = null;
            }
            DisplayException(exception);
            return results;
        }

        private string _lastExceptionMessage = "";
        private void DisplayException(Exception exception)
        {
            //txbExceptions.Text = exception == null ? "" :
exception.GetBaseException().Message;
            if (exception != null)
            {
                string newMessage = String.Format("{0} {1}", DateTime.Now,
exception.GetBaseException().Message);
                if (_lastExceptionMessage != newMessage)
                {
                    _lastExceptionMessage = newMessage;
                    txbExceptions.AppendText(newMessage + Environment.NewLine);
                }
```

```
            }
        }

        private void ReadMultiple()
        {
            DAVtqResult[] results = ReadMultipleVtq(new[]
               {
                   "Channel1.Device1.FanPower", "Channel1.Device1.FanSpeed",
"Channel1.Device1.HeaterPower",
                   "Channel1.Device1.HeaterTemp", "Channel1.Device1.TempSetPoint",
"Channel1.Device1.OvenTemp"
               });

            if (results == null)
                return;

            Debug.Assert(results[0] != null);
            Debug.Assert(results[1] != null);
            Debug.Assert(results[2] != null);
            Debug.Assert(results[3] != null);
            Debug.Assert(results[4] != null);
            Debug.Assert(results[5] != null);

            if (results[0].Exception == null) _fanPower = results[0].Vtq;
            else
            {
                _fanPower = null;
                DisplayException(results[0].Exception);
            }
            if (results[1].Exception == null) _fanSpeed = results[1].Vtq;
            else
            {
                _fanSpeed = null;
                DisplayException(results[1].Exception);
            }
            if (results[2].Exception == null) _heaterPower = results[2].Vtq;
            else
            {
                _heaterPower = null;
                DisplayException(results[2].Exception);
            }
            if (results[3].Exception == null) _heaterTemperature = results[3].Vtq;
            else
            {
                _heaterTemperature = null;
                DisplayException(results[3].Exception);
            }
            if (results[4].Exception == null) _temperatureSetpoint = results[4].Vtq;
            else
            {
                _temperatureSetpoint = null;
                DisplayException(results[4].Exception);
            }
            if (results[5].Exception == null) _ovenTemperature = results[5].Vtq;
            else
            {
                _ovenTemperature = null;
```

```
                DisplayException(results[5].Exception);
            }
        }


        private void UpdateValuesAndLog()
        {
            // read multiple item
            ReadMultiple();

            DisplayVtq(_fanPower, txbFanPower, 0);
            DisplayVtq(_fanSpeed, txbFanSpeed, 0);
            DisplayVtq(_heaterPower, txbHeaterPower, 0);
            DisplayVtq(_heaterTemperature, txbHeaterTemperature, 2);
            DisplayVtq(_temperatureSetpoint, txbTemperatureSetpoint, 0);
            DisplayVtq(_ovenTemperature, txbOvenTemperature, 1);

            WriteToLog();
        }

        // ReSharper disable InconsistentNaming
        private void btnSetTemperatureSetpoint_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            object value = txbNewTemperatureSetpoint.Text;
            Exception exception = null;
            try
            {
                easyDAClient1.WriteItemValue(
                    MachineName,
                    ServerClass,
                    "Channel1.Device1.TempSetPoint",
                    value);
            }
            catch (OpcException ex)
            {
                exception = ex;
            }
            DisplayException(exception);

            _temperatureSetpoint = ReadVtq("Channel1.Device1.TempSetPoint");
            DisplayVtq(_temperatureSetpoint, txbTemperatureSetpoint, 0);
            DisplayVtq(_ovenTemperature, txbOvenTemperature, 1);

        }

        private void WriteToLog()
        {
            try
            {
                var sw = new System.IO.StreamWriter(Application.StartupPath +
"\\OvenControl.csv", true);
                sw.WriteLine(
                    DateTime.Now.ToString(CultureInfo.InvariantCulture) + "," +
                    GetTextVtq(_fanPower) + "," +
                    GetTextVtq(_heaterPower) + "," +
                    GetTextVtq(_ovenTemperature) + "," +
```

```
                    GetTextVtq(_heaterTemperature) + "," +
                    GetTextVtq(_fanSpeed) + "," +
                    GetTextVtq(_temperatureSetpoint));

                sw.Close();
            }
            catch (Exception e)
            {
                DisplayException(e);
            }
        }

        // ReSharper disable InconsistentNaming
        private void btnClose_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            Close();
        }
    }
}
```

VB.NET

```
Imports System.Globalization
Imports JetBrains.Annotations
Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel
Imports OpcLabs.EasyOpc.DataAccess.OperationModel


Partial Public Class Form1
    Inherits Form

    Private Const MachineName As String = ""
    Private Const ServerClass As String = "SWToolbox.TOPServer.V5" ' or
"Kepware.KEPServerEX.V5"

    Private _fanPower As DAVtq
    Private _heaterPower As DAVtq
    Private _ovenTemperature As DAVtq
    Private _heaterTemperature As DAVtq
    Private _fanSpeed As DAVtq
    Private _temperatureSetpoint As DAVtq


    Public Sub New()
        InitializeComponent()
    End Sub

    Private _defaultBackColor As Color = Color.White


    ' ReSharper disable InconsistentNaming
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
        ' ReSharper restore InconsistentNaming
```

```
        SetDefaults()
        _defaultBackColor = txbOvenTemperature.BackColor
    End Sub

    Private Sub SetDefaults()
        nudUpdateRate.Value = 10
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnStart.Click
        ' ReSharper restore InconsistentNaming
        btnStart.Enabled = False
        timer1.Interval = Decimal.ToInt32(nudUpdateRate.Value * 1000)
        timer1.Start()
        UpdateValuesAndLog()
        btnStop.Enabled = True
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub btnStop_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnStop.Click
        ' ReSharper restore InconsistentNaming
        btnStop.Enabled = False
        timer1.Stop()
        btnStart.Enabled = True
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub timer1_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles
timer1.Tick
        ' ReSharper restore InconsistentNaming
        UpdateValuesAndLog()
    End Sub

    Private Function ConvertValueToInt32(ByVal vtq As DAVtq) As Int32?
        If vtq Is Nothing Then
            Return Nothing
        End If
        If Not vtq.HasValue() Then
            Return Nothing
        End If
        Return CType(vtq.Value, Int32?)
    End Function

    Private Function GetTextVtq(ByVal vtq As DAVtq) As String
        If (vtq Is Nothing) OrElse ((Not vtq.HasValue())) Then
            Return "???"
        End If
        Return vtq.DisplayValue()
    End Function

    Private Sub DisplayVtq(ByVal vtq As DAVtq, <NotNull()> ByVal txb As TextBox, ByVal
colorTestId As Integer)
        If (vtq Is Nothing) OrElse ((Not vtq.HasValue())) Then
            ' ReSharper disable LocalizableElement
            txb.Text = "???"
```

```
            ' ReSharper restore LocalizableElement
            txb.BackColor = Color.Magenta
        Else
            txb.Text = vtq.DisplayValue()

            Dim value? As Int32
            Dim aBackColor As Color = _defaultBackColor
            Select Case colorTestId
                Case 0
                    txb.BackColor = _defaultBackColor

                Case 1 ' oven temperature
                    value = ConvertValueToInt32(vtq)
                    If value.HasValue Then
                        Dim range? As Int32 = ConvertValueToInt32(_temperatureSetpoint)
                        If range.HasValue Then
                            If value <= (range - 5) Then
                                aBackColor = Color.Blue
                            Else
                                If value >= (range + 5) Then
                                    aBackColor = Color.Red
                                End If
                            End If
                        End If
                    End If
                    txb.BackColor = aBackColor

                Case 2 ' oven temperature
                    value = ConvertValueToInt32(vtq)
                    If value.HasValue Then
                        If value < 200 Then
                            aBackColor = Color.Red
                        End If
                    End If
                    txb.BackColor = aBackColor
            End Select
        End If
    End Sub

    Private Function ReadVtq(<NotNull()> ByVal itemId As String) As DAVtq
        Dim vtq As DAVtq
        Dim exception As Exception = Nothing
        Try
            vtq = easyDAClient1.ReadItem(MachineName, ServerClass, itemId)
        Catch ex As OpcException
            exception = ex
            vtq = Nothing
        End Try
        DisplayException(exception)
        Return vtq
    End Function

    <CanBeNull()> _
    Private Function ReadMultipleVtq(<NotNull()> ByVal itemIds() As String) As
DAVtqResult()
        Dim results() As DAVtqResult
        Dim exception As Exception = Nothing
```

```
        Dim itemDescriptors = New DAItemDescriptor(itemIds.Length - 1) {}
        For i As Integer = 0 To itemIds.Length - 1
            Debug.Assert(itemIds(i) IsNot Nothing)
            itemDescriptors(i) = New DAItemDescriptor(itemIds(i))
        Next i
        Try
            results = easyDAClient1.ReadMultipleItems(New ServerDescriptor(MachineName,
ServerClass), itemDescriptors)
        Catch ex As OpcException
            exception = ex
            results = Nothing
        End Try
        DisplayException(exception)
        Return results
    End Function

    Private _lastExceptionMessage As String = ""
    Private Sub DisplayException(ByVal exception As Exception)
        'txbExceptions.Text = exception == null ? "" :
exception.GetBaseException().Message;
        If exception IsNot Nothing Then
            Dim newMessage As String = String.Format("{0} {1}", Date.Now,
exception.GetBaseException().Message)
            If _lastExceptionMessage <> newMessage Then
                _lastExceptionMessage = newMessage
                txbExceptions.AppendText(newMessage & Environment.NewLine)
            End If
        End If
    End Sub

    Private Sub ReadMultiple()
        Dim results() As DAVtqResult = ReadMultipleVtq(New String()
{"Channel1.Device1.FanPower", "Channel1.Device1.FanSpeed",
"Channel1.Device1.HeaterPower", "Channel1.Device1.HeaterTemp",
"Channel1.Device1.TempSetPoint", "Channel1.Device1.OvenTemp"})

        If results Is Nothing Then
            Return
        End If

        Debug.Assert(results(0) IsNot Nothing)
        Debug.Assert(results(1) IsNot Nothing)
        Debug.Assert(results(2) IsNot Nothing)
        Debug.Assert(results(3) IsNot Nothing)
        Debug.Assert(results(4) IsNot Nothing)
        Debug.Assert(results(5) IsNot Nothing)

        If results(0).Exception Is Nothing Then
            _fanPower = results(0).Vtq
        Else
            _fanPower = Nothing
            DisplayException(results(0).Exception)
        End If
        If results(1).Exception Is Nothing Then
            _fanSpeed = results(1).Vtq
        Else
            _fanSpeed = Nothing
```

```
            DisplayException(results(1).Exception)
        End If
        If results(2).Exception Is Nothing Then
            _heaterPower = results(2).Vtq
        Else
            _heaterPower = Nothing
            DisplayException(results(2).Exception)
        End If
        If results(3).Exception Is Nothing Then
            _heaterTemperature = results(3).Vtq
        Else
            _heaterTemperature = Nothing
            DisplayException(results(3).Exception)
        End If
        If results(4).Exception Is Nothing Then
            _temperatureSetpoint = results(4).Vtq
        Else
            _temperatureSetpoint = Nothing
            DisplayException(results(4).Exception)
        End If
        If results(5).Exception Is Nothing Then
            _ovenTemperature = results(5).Vtq
        Else
            _ovenTemperature = Nothing
            DisplayException(results(5).Exception)
        End If
    End Sub


    Private Sub UpdateValuesAndLog()
        ' read multiple item
        ReadMultiple()

        DisplayVtq(_fanPower, txbFanPower, 0)
        DisplayVtq(_fanSpeed, txbFanSpeed, 0)
        DisplayVtq(_heaterPower, txbHeaterPower, 0)
        DisplayVtq(_heaterTemperature, txbHeaterTemperature, 2)
        DisplayVtq(_temperatureSetpoint, txbTemperatureSetpoint, 0)
        DisplayVtq(_ovenTemperature, txbOvenTemperature, 1)

        WriteToLog()
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub btnSetTemperatureSetpoint_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles btnSetTemperatureSetpoint.Click
        ' ReSharper restore InconsistentNaming
        Dim value As Object = txbNewTemperatureSetpoint.Text
        Dim exception As Exception = Nothing
        Try
            easyDAClient1.WriteItemValue(MachineName, ServerClass,
"Channel1.Device1.TempSetPoint", value)
        Catch ex As OpcException
            exception = ex
        End Try
        DisplayException(exception)
```

```
        _temperatureSetpoint = ReadVtq("Channel1.Device1.TempSetPoint")
        DisplayVtq(_temperatureSetpoint, txbTemperatureSetpoint, 0)
        DisplayVtq(_ovenTemperature, txbOvenTemperature, 1)

    End Sub

    Private Sub WriteToLog()
        Try
            Dim sw = New IO.StreamWriter(Application.StartupPath & "\OvenControl.csv",
True)
            sw.WriteLine(Date.Now.ToString(CultureInfo.InvariantCulture) & "," &
GetTextVtq(_fanPower) & "," & GetTextVtq(_heaterPower) & "," &
GetTextVtq(_ovenTemperature) & "," & GetTextVtq(_heaterTemperature) & "," &
GetTextVtq(_fanSpeed) & "," & GetTextVtq(_temperatureSetpoint))

            sw.Close()
        Catch e As Exception
            DisplayException(e)
        End Try
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub btnClose_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnClose.Click
        ' ReSharper restore InconsistentNaming
        Close()
    End Sub
End Class
```

13.1.4.3.6 Installed Examples - WindowsForms - QualityStrings

Shows how numerical OPC quality codes are converted to displayable strings (Windows Forms application).

The form:

C#

```
using OpcLabs.EasyOpc.DataAccess;

namespace QualityStrings
{
    public partial class Form1
    {

        // ReSharper disable once NotNullMemberIsNotInitialized
        internal Form1()
        {
            InitializeComponent();
        }
        // ReSharper disable InconsistentNaming
        private void Form1_Load(object sender, System.EventArgs e)
```

```
        // ReSharper restore InconsistentNaming
        {
            //define the quality codes that we want to run thru our function as a test
            var qualityTests = new [] {0, 10, 50, 64, 100, 128, 150, 192, 210};

            //now test each of the quality codes defined above
            foreach (int qualityTest in qualityTests)
            {
                ListBox1.Items.Add(new DAQuality(qualityTest));
            }
        }

    private static Form1 _defaultInstance;
    public static Form1 DefaultInstance
    {
        get
        {
            // ReSharper disable ConvertIfStatementToNullCoalescingExpression
            if (_defaultInstance == null)
            // ReSharper restore ConvertIfStatementToNullCoalescingExpression
                _defaultInstance = new Form1();

            return _defaultInstance;
        }
    }
    }

}
```

VB.NET

```
Imports OpcLabs.EasyOpc.DataAccess

Public Class Form1

    ' ReSharper disable InconsistentNaming
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As EventArgs) Handles
MyBase.Load
        ' ReSharper restore InconsistentNaming

        'define the quality codes that we want to run thru our function as a test
        Dim qualityTests() As Integer = New Integer() {0, 10, 50, 64, 100, 128, 150,
192, 210}

        'now test each of the quality codes defined above
        For Each qualityTest As Integer In qualityTests
            ListBox1.Items.Add(New DAQuality(qualityTest))
        Next
    End Sub
End Class
```

13.1.4.3.7 Installed Examples - WindowsForms - SubscribeToMany

Demonstrates and measures performance with large number of subscribed OPC items.

The form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.

using System.Diagnostics;
using System.Globalization;
using JetBrains.Annotations;
using OpcLabs.EasyOpc.DataAccess;
using System;
using System.Windows.Forms;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace SubscribeToMany
{
    public partial class Form1 : Form
    {
        // ReSharper disable once NotNullMemberIsNotInitialized
        public Form1()
        {
            InitializeComponent();
        }

        int _changeCount;
        int _startTickCount;

        double GetElapsedTime()
        {
            return (Environment.TickCount - _startTickCount) / 1000.0;
        }

        // ReSharper disable InconsistentNaming
        private void startButton_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            // ReSharper disable SuggestUseVarKeywordEvident
            int numberOfItems = (int)numberOfItemsNumericUpDown.Value;
            // ReSharper restore SuggestUseVarKeywordEvident

            startButton.Enabled = false;
            numberOfItemsNumericUpDown.Enabled = false;

            var argumentArray = new DAItemGroupArguments[numberOfItems];
            for (int i = 0; i < numberOfItems; i++)
            {
                var listViewItem = new ListViewItem();
                valuesListView.Items.Add(listViewItem);
                int copy = (i / 100) + 1;
```

```
                int phase = (i%100) + 1;
                string itemId =
String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", copy, phase);
                argumentArray[i] = new DAItemGroupArguments("", "OPCLabs.KitServer.2",
itemId, 50, listViewItem);
            }

            _changeCount = 0;
            _startTickCount = Environment.TickCount;
            timer1.Start();

            easyDAClient1.SubscribeMultipleItems(argumentArray);

        }

        // ReSharper disable InconsistentNaming
        private void easyDAClient1_ItemChanged([NotNull] object sender, [NotNull]
EasyDAItemChangedEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            _changeCount++;

            var listViewItem = (ListViewItem)e.Arguments.State;
            Debug.Assert(listViewItem != null);

            string text;
            if (e.Exception == null)
            {
                Debug.Assert(e.Vtq != null);
                text = e.Vtq.DisplayValue();
            }
            else
                text = "*Error*";
            listViewItem.Text = text;
        }

        // ReSharper disable InconsistentNaming
        private void timer1_Tick(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            double elapsedTime = GetElapsedTime();
            // ReSharper disable CompareOfFloatsByEqualityOperator
            if (elapsedTime == 0.0d)
            // ReSharper restore CompareOfFloatsByEqualityOperator
                return;

            int changeCount = _changeCount;
            double changesPerSecond = changeCount / elapsedTime;

            changeCountTextBox.Text = changeCount.ToString(CultureInfo.CurrentCulture);
            elapsedTimeTextBox.Text = String.Format("{0:0.0}", elapsedTime);
            changesPerSecondTextBox.Text = String.Format("{0:0.0}", changesPerSecond);
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.DataAccess
'using System.Linq;

Imports OpcLabs.EasyOpc.DataAccess.OperationModel ' ReSharper disable CheckNamespace
Namespace SubscribeToMany
    ' ReSharper restore CheckNamespace

    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
        End Sub

        Private _changeCount As Integer
        Private _startTickCount As Integer

        Private Function GetElapsedTime() As Double
            Return (Environment.TickCount - _startTickCount) / 1000.0
        End Function

        ' ReSharper disable InconsistentNaming
        Private Sub startButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles startButton.Click
            ' ReSharper restore InconsistentNaming

            Dim numberOfItems As Integer = CInt(Fix(numberOfItemsNumericUpDown.Value))

            startButton.Enabled = False
            numberOfItemsNumericUpDown.Enabled = False

            Dim argumentArray = New DAItemGroupArguments(numberOfItems - 1) {}
            For i As Integer = 0 To numberOfItems - 1
                Dim listViewItem = New ListViewItem()
                valuesListView.Items.Add(listViewItem)
                Dim copy As Integer = (i \ 100) + 1
                Dim phase As Integer = i Mod 100
                Dim itemId As String =
String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", copy, phase)
                argumentArray(i) = New DAItemGroupArguments("", "OPCLabs.KitServer.2",
itemId, 50, listViewItem)
            Next i

            _changeCount = 0
            _startTickCount = Environment.TickCount
            timer1.Start()

            easyDAClient1.SubscribeMultipleItems(argumentArray)

        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub easyDAClient1_ItemChanged(ByVal sender As Object, ByVal e As
```

```
EasyDAItemChangedEventArgs) Handles easyDAClient1.ItemChanged
            ' ReSharper restore InconsistentNaming

            _changeCount += 1

            Dim listViewItem = CType(e.Arguments.State, ListViewItem)
            Dim itemText As String
            If e.Exception IsNot Nothing Then
                itemText = "*Error*"
            Else
                itemText = e.Vtq.DisplayValue()
            End If
            listViewItem.Text = itemText
        End Sub

        ' ReSharper disable InconsistentNaming
        Private Sub timer1_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles
timer1.Tick
            ' ReSharper restore InconsistentNaming

            Dim elapsedTime As Double = GetElapsedTime()
            ' ReSharper disable CompareOfFloatsByEqualityOperator
            If elapsedTime = 0.0 Then
                ' ReSharper restore CompareOfFloatsByEqualityOperator
                Return
            End If

            Dim changeCount As Integer = _changeCount
            Dim changesPerSecond As Double = changeCount / elapsedTime

            changeCountTextBox.Text = changeCount.ToString()
            elapsedTimeTextBox.Text = String.Format("{0:0.0}", elapsedTime)
            changesPerSecondTextBox.Text = String.Format("{0:0.0}", changesPerSecond)
        End Sub
    End Class
End Namespace
```

13.1.4.3.8 Installed Examples - WindowsForms - ValueToMessageBox

Demonstrates and measures performance with large number of subscribed OPC items.

The form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
using System;
using System.Windows.Forms;
using OpcLabs.EasyOpc.DataAccess;
```

```
namespace ValueToMessageBox
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void button1_Click(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            // Create EasyOPC-DA component
            var client = new EasyDAClient();

            // Read item value and display it in a message box
            object value = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Single");
            if (value != null)
                MessageBox.Show(value.ToString());
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.DataAccess

Public Class Form1

    ' ReSharper disable InconsistentNaming
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As EventArgs)
Handles Button1.Click
        ' ReSharper restore InconsistentNaming

        ' Create EasyOPC-DA component
        Dim easyDaClient As New OpcLabs.EasyOpc.DataAccess.EasyDAClient

        ' Read item value and display it in a message box
        MessageBox.Show(easyDaClient.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Single"))
    End Sub
End Class
```

13.1.4.3.9 Installed Examples - WindowsForms - WindowsFormsApplication1

The simplest Windows Forms application. Reads and displays an OPC item value on a form. This is what you get if you follow the steps described in Quick Start for OPC Data Client.NET.

The form:

C#

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
using System;
using System.Windows.Forms;
using OpcLabs.EasyOpc.DataAccess;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        // ReSharper disable once NotNullMemberIsNotInitialized
        public Form1()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void Form1_Load(object sender, EventArgs e)
        // ReSharper restore InconsistentNaming
        {
            object value = easyDAClient1.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Single");
            textBox1.Text = (value == null) ? "" : value.ToString();
        }
    }
}
```

VB.NET

```
' $Header: $
' Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
Imports OpcLabs.EasyOpc.DataAccess

Public Class Form1

    ' ReSharper disable InconsistentNaming
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As EventArgs) Handles
MyBase.Load
        ' ReSharper restore InconsistentNaming

        TextBox1.Text = EasyDAClient1.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Single")
    End Sub
End Class
```

13.1.4.4 Installed Examples - WindowsService

13.1.4.4.1 Installed Examples - WindowsService - WindowsService1

A Windows Service that subscribes to items from the simulation server, and logs their changes into a file.

The service:

C#

```
// WindowsService1: A Windows Service that subscribes to items from the simulation
server, and logs their changes into
// a file.

// Install the service by running:
//      C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe /i
WindowsService1.exe
// If you get "Access denied" error when starting the service, change its configuration
to run under Local System account.

using System;
using System.Diagnostics;
using System.IO;
using System.ServiceProcess;
using JetBrains.Annotations;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace WindowsService1
{
    public partial class Service1 : ServiceBase
    {
        const string FilePath = "C:\\Service1.txt";

        // ReSharper disable once NotNullMemberIsNotInitialized
        public Service1()
        {
            InitializeComponent();
        }

        protected override void OnStart(string[] args)
        {
            File.Create(FilePath).Close();

            easyDAClient1.SubscribeMultipleItems(
                new[]
                    {
                        new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Incrementing (1 s)", 100, null),
                        new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Ramp (10 s)", 1000, null),
                        new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BSTR", 1000, null),
                        new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BOOL", 1000, null)
                    });
```

```
        }

        protected override void OnStop()
        {
            easyDAClient1.UnsubscribeAllItems();
        }

        // ReSharper disable InconsistentNaming
        private void easyDAClient1_ItemChanged([NotNull] object sender, [NotNull]
EasyDAItemChangedEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            string line;
            if (e.Exception == null)
            {
                Trace.Assert(e.Vtq != null);
                line = string.Format("{0}: {1}", e.Arguments.ItemDescriptor.ItemId,
e.Vtq.DisplayValue());
            }
            else
                line = String.Format("{0}: ** {1} **",
e.Arguments.ItemDescriptor.ItemId, e.Exception.GetBaseException());

            using (var textWriter = File.AppendText(FilePath))
                textWriter.WriteLine(line);
        }
    }
}
```

VB.NET

```
' WindowsService1: A Windows Service that subscribes to items from the simulation
server, and logs their changes into
' a file.

' Install the service by running:
'      C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe /i
WindowsService1.exe
' If you get "Access denied" error when starting the service, change its configuration
to run under Local System account.
Imports System.IO
Imports System.ServiceProcess
Imports JetBrains.Annotations
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Partial Public Class Service1
    Inherits ServiceBase

    Private Const FilePath As String = "C:\Service1.txt"

    Public Sub New()
        InitializeComponent()
    End Sub

    Protected Overrides Sub OnStart(ByVal args() As String)
        File.Create(FilePath).Close()
```

```
        easyDAClient1.SubscribeMultipleItems(New DAItemGroupArguments() { _
            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Incrementing (1 s)", 100, Nothing), _
            New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Simulation.Ramp (10
s)", 1000, Nothing), _
            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BSTR", 1000, Nothing), _
            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_BOOL", 1000, Nothing) _
        })
    End Sub

    Protected Overrides Sub OnStop()
        easyDAClient1.UnsubscribeAllItems()
    End Sub

    ' ReSharper disable InconsistentNaming
    Private Sub easyDAClient1_ItemChanged(<NotNull()> ByVal sender As Object,
<NotNull()> ByVal e As EasyDAItemChangedEventArgs) Handles easyDAClient1.ItemChanged
        ' ReSharper restore InconsistentNaming
        Dim line As String
        If e.Exception Is Nothing Then
            Trace.Assert(e.Vtq IsNot Nothing)
            line = String.Format("{0}: {1}", e.Arguments.ItemDescriptor.ItemId,
e.Vtq.DisplayValue())
        Else
            line = String.Format("{0}: ** {1} **", e.Arguments.ItemDescriptor.ItemId,
e.Exception.GetBaseException())
        End If

        Using textWriter = File.AppendText(FilePath)
            textWriter.WriteLine(line)
        End Using
    End Sub
End Class
```

13.1.4.5 Installed Examples - WPF

13.1.4.5.1 Installed Examples - WPF - OpcDaQualityDecoder

A simple WPF application that decodes the cryptic OPC quality numbers into the separate fields and their symbolic representation.

The service:

C#

```
using OpcLabs.EasyOpc.DataAccess;
using System;
using System.Windows;
```

```
namespace OpcDAQualityDecoder
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1
    {
        public Window1()
        {
            InitializeComponent();
        }

        // ReSharper disable InconsistentNaming
        private void DecodeButton_Click(object sender, RoutedEventArgs e)
        // ReSharper restore InconsistentNaming
        {
            int value = Convert.ToUInt16(valueTextBox.Text);
            var quality = new DAQuality(value);

            qualityChoiceStringTextBox.Text = quality.QualityChoiceBitField.ToString();
            statusStringTextBox.Text = quality.StatusBitField.ToString();
            limitStringTextBox.Text = quality.LimitBitField.ToString();
        }
    }
}
```

13.1.4.5.2 Installed Examples - WPF - UAWpfScreen

Shows how to update WPF controls with dynamic OPC-UA data.

The service:

C#

```
using System.Windows;
using System.Windows.Controls;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UAWpfScreen
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow/* : Window*/
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void ClientOnDataChangeNotification(object sender,
```

```
EasyUADataChangeNotificationEventArgs eventArgs)
        {
            // We have passed in a reference to the target TextBox as the State
property in EasyUAMonitoredItemArguments.
            var textBox = (TextBox) eventArgs.Arguments.State;

            if (eventArgs.Succeeded)
                textBox.Text = eventArgs.AttributeData.DisplayValue();
            else
                textBox.Text = "*** Error: " + eventArgs.Exception;
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            // Define the target controls, the OPC data we want to monitor, and how
fast.
            var arguments = new []
            {
                new EasyUAMonitoredItemArguments(TextBox1, "http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10845", 1000),
                new EasyUAMonitoredItemArguments(TextBox2, "http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10853", 1000),
                new EasyUAMonitoredItemArguments(TextBox3, "http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10855", 1000)
            };

            // Hook the event handler, and subscribe to OPC data
            _client.DataChangeNotification += ClientOnDataChangeNotification;
            _handles = _client.SubscribeMultipleMonitoredItems(arguments);
        }

        private void Window_Unloaded(object sender, RoutedEventArgs e)
        {
            // Unsubscribe from OPC data, and unhook the event handler
            _client.UnsubscribeMultipleMonitoredItems(_handles);
            _client.DataChangeNotification -= ClientOnDataChangeNotification;
        }

        private readonly EasyUAClient _client = new EasyUAClient();

        private int[] _handles;
    }
}
```

13.1.4.5.3 Installed Examples - WPF - WpfApplication1

The simplest WPF application for OPC “Classic”. Reads and displays an OPC item value.

The service:

C#

```
using System.Windows;
```

```
using OpcLabs.EasyOpc.DataAccess;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow/* : Window*/
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            var client = new EasyDAClient();
            var value = client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Single");
            TextBox1.Text = (value== null) ? "" : value.ToString();
        }
    }
}
```

13.2 Examples in the Documentation (List)

This documentation section contains a list of all examples included directly in the documentation.

> Only examples that are able to "stand alone", i.e. have a function of their own that can be individually described and demonstrated, are listed. Small code snippets that only form a part of a larger article are not considered for this index.

- Examples - Licensing
  - **Examples - Licensing - Obtain serial number (Section 13.2.1.1)**
  - **Examples - Licensing - Register managed resource (Section 13.2.1.2)**
- Examples - Live Mapping
  - **Examples - Live Mapping - Define OPC Data Access type-less mapping and read (Section 13.2.2.1)**
  - **Examples - Live Mapping - Define OPC Unified Architecture type-less mapping and read (Section 13.2.2.2)**
  - **Examples - Live Mapping - Subscribe and subscribe with OPC Data Access type-less mapping (Section 13.2.2.3)**
  - **Examples - Live Mapping - Various data kinds with OPC Data Access type-less mapping (Section 13.2.2.4)**
- Examples - OPC Alarms&Events
  - **Examples - OPC Alarms&Events - Acknowledge a condition (Section 13.2.3.1)**
  - **Examples - OPC Alarms&Events - Browse for areas (Section 13.2.3.2)**
  - **Examples - OPC Alarms&Events - Browse for servers (Section 13.2.3.3)**
  - **Examples - OPC Alarms&Events - Browse for sources (Section 13.2.3.4)**
  - **Examples - OPC Alarms&Events - Callback using a lambda (Section 13.2.3.5)**

13.2.1 Examples - Licensing

13.2.1.1 Examples - Licensing - Obtain serial number

The example below shows how to obtain the serial number of the active license, and determine whether it is a demo or trial license.

C#

```
// Shows how to obtain the serial number of the active license, and determine whether
it is a stock demo or trial license.

using System;
using OpcLabs.EasyOpc.UA;

namespace UADocExamples.Licensing
{
    class LicenseInfo
    {
        public static void SerialNumber()
        {
```

```
            // Instantiate the client object.
            var client = new EasyUAClient();

            // Obtain the serial number from the license info.
            long serialNumber = (uint)client.LicenseInfo["Multipurpose.SerialNumber"];

            // Display the serial number.
            Console.WriteLine("SerialNumber: {0}", serialNumber);

            // Determine whether we are running as demo or trial.
            if ((1111110000 <= serialNumber) && (serialNumber <= 1111119999))
                Console.WriteLine("This is a stock demo or trial license.");
            else
                Console.WriteLine("This is not a stock demo or trial license.");
        }
    }
}
```

Object Pascal

```
// Shows how to obtain the serial number of the active license, and determine whether
it is a stock demo or trial license.

class procedure LicenseInfo.SerialNumber;
var
  Client: _EasyUAClient;
  SerialNumber: Int64;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain the serial number from the license info.
  SerialNumber := Int64(client.LicenseInfo['Multipurpose.SerialNumber']);

  // Display the serial number.
  WriteLn('SerialNumber: ', SerialNumber);

  // Determine whether we are running as demo or trial.
  if ((1111110000 <= SerialNumber) and (SerialNumber <= 1111119999)) then
    WriteLn('This is a stock demo or trial license.')
  else
    WriteLn('This is not a stock demo or trial license.');
end;
```

PHP

```
// This example shows how to obtain nodes under a given node of the OPC-UA address
space.
// Shows how to obtain the serial number of the active license, and determine whether
it is a stock demo or trial license.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain the serial number from the license info.
$SerialNumber = $Client->LicenseInfo["Multipurpose.SerialNumber"];
```

```
// Display the serial number.
printf("SerialNumber: %s\n", $SerialNumber);

// Determine whether we are running as demo or trial.
if ((1111110000 <= $SerialNumber) and ($SerialNumber <= 1111119999))
    printf("This is a stock demo or trial license.\n");
  else
    printf("This is not a stock demo or trial license.\n");
```

VB.NET

```
' Shows how to obtain the serial number of the active license, and determine whether it
is a stock demo or trial license.

Imports OpcLabs.EasyOpc.UA

Namespace UADocExamples.Licensing
    Friend Class LicenseInfo
        Public Shared Sub SerialNumber()

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            ' Obtain the serial number from the license info.
            Dim serialNumber As Long =
CUInt(client.LicenseInfo("Multipurpose.SerialNumber"))

            ' Display the serial number.
            Console.WriteLine("SerialNumber: {0}", serialNumber)

            ' Determine whether we are running as demo or trial.
            If (1111110000 <= serialNumber) And (serialNumber <= 1111119999) Then
                Console.WriteLine("This is a stock demo or trial license.")
            Else
                Console.WriteLine("This is not a stock demo or trial license.")
            End If
        End Sub
    End Class
End Namespace
```

13.2.1.2 Examples - Licensing - Register managed resource

The example below shows how to register a license located in an embedded managed resource.

C#

```
// Shows how to register a license located in an embedded managed resource.

using System;
using System.Reflection;
using OpcLabs.BaseLib.ComponentModel;
using OpcLabs.EasyOpc.UA;
```

```
namespace UADocExamples.Licensing
{
    class _LicensingManagement
    {
        public static void RegisterManagedResource()
        {
            // Register a license that is we embed as a managed resource in this
program.

            // The first two arguments should always be "QuickOPC" and "Multipurpose".
            // The third argument determines the assembly where the license resides.
            // The fourth argument is the namespace-qualified name of the managed
resource.
            LicensingManagement.Instance.RegisterManagedResource("QuickOPC",
"Multipurpose",
                Assembly.GetExecutingAssembly(),
                "UADocExamples.Licensing.Key-DemoOrTrial-WebForm-1999003494-
20180611.bin");

            // Instantiate the client object, obtain the serial number from the license
info, and display the serial number.
            var client = new EasyUAClient();
            long serialNumber = (uint)client.LicenseInfo["Multipurpose.SerialNumber"];
            Console.WriteLine("SerialNumber: {0}", serialNumber);

            // The license we ship for this purpose is a trial license with low runtime
limit, so it won't be of much use.
            // But you get the point...
        }
    }
}
```

VB.NET

```
' Shows how to register a license located in an embedded managed resource.

Imports System.Reflection
Imports OpcLabs.BaseLib.ComponentModel
Imports OpcLabs.EasyOpc.UA

Namespace UADocExamples.Licensing
    Friend Class _LicensingManagement
        Public Shared Sub RegisterManagedResource()

            ' Register a license that is we embed as a managed resource in this
program.

            ' The first two arguments should always be "QuickOPC" and "Multipurpose".
            ' The third argument determines the assembly where the license resides.
            ' The fourth argument is the namespace-qualified name of the managed
resource.
            LicensingManagement.Instance.RegisterManagedResource("QuickOPC",
"Multipurpose",
                    Assembly.GetExecutingAssembly(),
                    "UADocExamples.Licensing.Key-DemoOrTrial-WebForm-1999003494-
20180611.bin")
```

```
            ' Instantiate the client object, obtain the serial number from the license
info, and display the serial number.
            Dim client = New EasyUAClient()
            Dim serialNumber As Long =
CUInt(client.LicenseInfo("Multipurpose.SerialNumber"))
            Console.WriteLine("SerialNumber: {0}", serialNumber)

            ' The license we ship for this purpose is a trial license with low runtime
limit, so it won't be of much use.
            ' But you get the point...

        End Sub
    End Class
End Namespace
```

13.2.2 Examples - Live Mapping

13.2.2.1 Examples - Live Mapping - Define OPC Data Access type-less mapping and read

The example below shows how to define a new type-less mapping which maps a specified OPC-DA item to the Value member of your target MyClass object. The mapper is then instructed to invoke the OPC read operation, which will in turn store the OPC item's value to your target object.

Type-less mapping (OPC DA) in C#

```
var mapper = new DAClientMapper();
var target = new MyClass2();

// Define a type-less mapping.

mapper.DefineMapping(
     new DAClientItemSource("OPCLabs.KitServer.2", "Simulation.Register_I4",
DADataSource.Cache),
     new DAClientItemMapping(typeof(Int32)),
     new ObjectMemberLinkingTarget(target.GetType(), target, "Value"));

// Perform a read operation.
mapper.Read();
```

Type-less mapping (OPC DA) in VB.NET

```
Public Shared Sub Main1()
    Dim mapper = New DAClientMapper()
    Dim target = New MyClass2()
```

```
    ' Define a type-less mapping.

    mapper.DefineMapping( _
             New DAClientItemSource( _
                 "OPCLabs.KitServer.2",
                 "Simulation.Register_I4",
                 New DAReadParameters(DADataSource.Cache)),
             New DAClientItemMapping(GetType(Int32)),
             New ObjectMemberLinkingTarget(target.GetType(), target, "Value"))

    ' Perform a read operation.
    mapper.Read()

    ' Display the result.
    Console.WriteLine(target.Value)
End Sub
```

13.2.2.2 Examples - Live Mapping - Define OPC Unified Architecture type-less mapping and read

The example below shows how to define a new type-less mapping which maps a specified node in OPC Unified Architecture (OPC-UA) server to the Value member of your target MyClass2 object. The mapper is then instructed to invoke the OPC read operation, which will in turn store the OPC node's value to your target object:

Type-less mapping (OPC UA) in C#

```
UAEndpointDescriptor endpointDescriptor =
    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
// or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
// or "https://opcua.demo-this.com:51212/UA/SampleServer/"

var mapper = new UAClientMapper();
var target = new MyClass2();

// Define a type-less mapping.

MemberInfo memberInfo = target.GetType().GetMember("Value").SingleOrDefault();
Debug.Assert(memberInfo != null);

mapper.DefineMapping(
    new UAClientDataMappingSource(
        endpointDescriptor,
        "nsu=http://test.org/UA/Data/;i=10389",
        UAAttributeId.Value,
        UAIndexRangeList.Empty,
```

```
        UAReadParameters.CacheMaximumAge),
    new UAClientDataMapping(typeof(Int32)),
    new ObjectMemberLinkingTarget(target.GetType(), target, memberInfo));

// Perform a read operation.
mapper.Read();
```

Type-less mapping (OPC UA) in VB.NET

```
Public Shared Sub Main1()

    ' Define which server we will work with.
    Dim endpointDescriptor As UAEndpointDescriptor =
            "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
    ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
    ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

    Dim mapper = New UAClientMapper()
    Dim target = New MyClass2()

    ' Define a type-less mapping.

    Dim memberInfo = target.GetType().GetMember("Value").SingleOrDefault()
    Debug.Assert(memberInfo IsNot Nothing)

    mapper.DefineMapping( _
        New UAClientDataMappingSource( _
            endpointDescriptor, _
            "nsu=http://test.org/UA/Data/;i=10389", _
            UAAttributeId.Value, _
            UAIndexRangeList.Empty, _
            UAReadParameters.CacheMaximumAge), _
        New UAClientDataMapping(GetType(Int32)), _
        New ObjectMemberLinkingTarget(target.GetType(), target, memberInfo))

    ' Perform a read operation.
    mapper.Read()

    ' Display results
    Console.WriteLine(target.Value)
End Sub
```

13.2.2.3 Examples - Live Mapping - Subscribe and subscribe with OPC Data Access type-less mapping

The example below shows how to subscribe and unsubscribe:

Subscribe and subscribe with type-less mapping in C#

```
// This example for OPC DA type-less mapping shows how to define a mapping and perform
subscribe and unsubscribe operations.

using System;
using System.Threading;
using OpcLabs.BaseLib.ComponentModel.Linking;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;

namespace DocExamples.DataAccess._DAClientMapper
{
    partial class DefineMapping
    {
        class MyClassSubscribe
        {
            public Double Value
            {
                set
                {
                    // Display the incoming value
                    Console.WriteLine(value);
                }
            }
        }

        public static void Subscribe()
        {
            var mapper = new DAClientMapper();
            var target = new MyClassSubscribe();

            // Define a type-less mapping.

            mapper.DefineMapping(
                 new DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp", 1000,
DADataSource.Cache),
                 new DAClientItemMapping(typeof(Double)),
                 new ObjectMemberLinkingTarget(target.GetType(), target, "Value"));

            // Perform a subscribe operation.
            mapper.Subscribe(true);

            Thread.Sleep(30 * 1000);

            // Perform an unsubscribe operation.
```

```
            mapper.Subscribe(false);
        }
    }
}
```

Subscribe and subscribe with type-less mapping in VB.NET

```
' This example for OPC DA type-less mapping shows how to define a mapping and perform
subscribe and unsubscribe operations.

Imports OpcLabs.BaseLib.ComponentModel.Linking
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping

Namespace DocExamples._DAClientMapper
    Partial Friend Class DefineMapping
        Class MyClassSubscribe
            Public WriteOnly Property Value As Double
                Set(value As Double)
                    ' Display the incoming value
                    Console.WriteLine(value)
                End Set
            End Property
        End Class

        Public Shared Sub Subscribe()
            Dim mapper = New DAClientMapper()
            Dim target = New MyClassSubscribe()

            ' Define a type-less mapping.

            mapper.DefineMapping(
                 New DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp", 1000,
DADataSource.Cache),
                 New DAClientItemMapping(GetType(Double)),
                 New ObjectMemberLinkingTarget(target.GetType(), target, "Value"))

            ' Perform a subscribe operation.
            mapper.Subscribe(True)

            Threading.Thread.Sleep(30 * 1000)

            ' Perform an unsubscribe operation.
            mapper.Subscribe(False)
        End Sub
    End Class
End Namespace
```

13.2.2.4 Examples - Live Mapping - Various data kinds with OPC Data Access type-less mapping

The example below shows how to make various kinds of mappings:

Various mapping kinds with type-less mapping in C#

```
// This example for OPC DA type-less mapping shows how to define mappings of various
kinds, and perform subscribe and
// unsubscribe operations.

using System;
using System.Threading;
using OpcLabs.BaseLib.ComponentModel.Linking;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Generic;
using OpcLabs.EasyOpc.DataAccess.LiveMapping;

namespace DocExamples.DataAccess._DAClientMapper
{
    partial class DefineMapping
    {
        class MyClassMappingKinds
        {
            public Double CurrentValue
            {
                set
                {
                    // Display the incoming value
                    Console.WriteLine("Value: {0}", value);
                }
            }

            public DAVtq<Double> CurrentVtq
            {
                set
                {
                    // Display the incoming Vtq
                    Console.WriteLine("Vtq: {0}", value);
                }
            }

            public Exception CurrentException
            {
                set
```

```
                {
                    // Display the incoming exception
                    Console.WriteLine("Exception: {0}", value);
                }
            }

            public DAVtqResult<Double> CurrentResult
            {
                set
                {
                    // Display the incoming result
                    Console.WriteLine("Result: {0}", value);
                }
            }
        }

        public static void MappingKinds()
        {
            var mapper = new DAClientMapper();
            var target = new MyClassMappingKinds();

            // Define several type-less mappings for the same source, with different
mapping kinds.

            Type targetType = target.GetType();
            var source = new DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp",
1000, DADataSource.Cache);

            mapper.DefineMapping(
                 source,
                 new DAClientItemMapping(typeof(Double)),
                 new ObjectMemberLinkingTarget(targetType, target, "CurrentValue"));

            mapper.DefineMapping(
                 source,
                 new DAClientItemMapping(typeof(Double), DAItemMappingKind.Vtq),
                 new ObjectMemberLinkingTarget(targetType, target, "CurrentVtq"));

            mapper.DefineMapping(
                 source,
                 new DAClientItemMapping(typeof(Double), DAItemMappingKind.Exception),
                 new ObjectMemberLinkingTarget(targetType, target, "CurrentException"));

            mapper.DefineMapping(
                 source,
                 new DAClientItemMapping(typeof(Double), DAItemMappingKind.Result),
                 new ObjectMemberLinkingTarget(targetType, target, "CurrentResult"));
```

```
            // Perform a subscribe operation.
            mapper.Subscribe(true);

            Thread.Sleep(30 * 1000);

            // Perform an unsubscribe operation.
            mapper.Subscribe(false);
        }
    }
}
```

Various mapping kinds with type-less mapping in VB.NET

```
' This example for OPC DA type-less mapping shows how to define mappings of various
kinds, and perform subscribe and
' unsubscribe operations.

Imports OpcLabs.BaseLib.ComponentModel.Linking
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Generic
Imports OpcLabs.EasyOpc.DataAccess.LiveMapping

Namespace DocExamples._DAClientMapper
    Partial Friend Class DefineMapping
        Class MyClassMappingKinds
            Public WriteOnly Property CurrentValue As Double
                Set(value As Double)
                    ' Display the incoming value
                    Console.WriteLine("Value: {0}", value)
                End Set
            End Property

            Public WriteOnly Property CurrentVtq As DAVtq(Of Double)
                Set(value As DAVtq(Of Double))
                    ' Display the incoming Vtq
                    Console.WriteLine("Vtq: {0}", value)
                End Set
            End Property

            Public WriteOnly Property CurrentException As Exception
                Set(value As Exception)
                    ' Display the incoming exception
                    Console.WriteLine("Exception: {0}", value)
                End Set
            End Property

            Public WriteOnly Property CurrentResult As DAVtqResult(Of Double)
                Set(value As DAVtqResult(Of Double))
```

```
                    ' Display the incoming result
                    Console.WriteLine("Result: {0}", value)
                End Set
            End Property
        End Class

        Public Shared Sub MappingKinds()
            Dim mapper = New DAClientMapper()
            Dim target = New MyClassMappingKinds()

            ' Define several type-less mappings for the same source, with different
mapping kinds.

            Dim targetType = target.GetType()
            Dim source = New DAClientItemSource("OPCLabs.KitServer.2", "Demo.Ramp",
1000, DADataSource.Cache)

            mapper.DefineMapping(
                 source,
                 New DAClientItemMapping(GetType(Double)),
                 New ObjectMemberLinkingTarget(targetType, target, "CurrentValue"))

            mapper.DefineMapping(
                 source,
                 New DAClientItemMapping(GetType(Double), DAItemMappingKind.Vtq),
                 New ObjectMemberLinkingTarget(targetType, target, "CurrentVtq"))

            mapper.DefineMapping(
                 source,
                 New DAClientItemMapping(GetType(Double), DAItemMappingKind.Exception),
                 New ObjectMemberLinkingTarget(targetType, target, "CurrentException"))

            mapper.DefineMapping(
                 source,
                 New DAClientItemMapping(GetType(Double), DAItemMappingKind.Result),
                 New ObjectMemberLinkingTarget(targetType, target, "CurrentResult"))

            ' Perform a subscribe operation.
            mapper.Subscribe(True)

            Threading.Thread.Sleep(30 * 1000)

            ' Perform an unsubscribe operation.
            mapper.Subscribe(False)
        End Sub
    End Class
End Namespace
```

13.2.3 Examples - OPC Alarms&Events

13.2.3.1 Examples - OPC Alarms&Events - Acknowledge a condition

C#

```
// This example shows how to acknowledge an event condition in the OPC server.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class AcknowledgeCondition
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        static volatile bool _done;

        public static void Main1()
        {
            var eventHandler = new
EasyAENotificationEventHandler(AEClient_Notification);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications for 1 minute...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1" }
            };
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // Give the refresh operation time to complete
            Thread.Sleep(5 * 1000);

            // Trigger an acknowledgeable event
            try
            {
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
```

```
opcException.GetBaseException().Message);
                return;
            }

            _done = false;
            DateTime endTime = DateTime.Now + new TimeSpan(0, 0, 5);
            while ((!_done) && (DateTime.Now < endTime))
                Thread.Sleep(1000);

            // Give some time to also receive the acknowledgement notification
            Thread.Sleep(5 * 1000);

            AEClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void AEClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
                Console.WriteLine("Event.AcknowledgeRequired: {0}",
eventData.AcknowledgeRequired);

                if (eventData.AcknowledgeRequired)
                {
                    Console.WriteLine(">>>>> ACKNOWLEDGING THIS EVENT");
                    try
                    {
                        AEClient.AcknowledgeCondition("", "OPCLabs.KitEventServer.2",
"Simulation.ConditionState1", "Simulated",
                            eventData.ActiveTime, eventData.Cookie);
                    }
                    catch (OpcException opcException)
                    {
                        Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                        return;
                    }
                    Console.WriteLine(">>>>> EVENT ACKNOWLEDGED");
                    _done = true;
                }
            }
```

```
        }
    }
}
```

VB.NET

```
' This example shows how to acknowledge an event condition in the OPC server.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class AcknowledgeCondition

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Private Shared _done As Boolean ' volatile

        Public Shared Sub Main1()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications for 1 minute...")
            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1"}
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter)

            ' Give the refresh operation time to complete
            Thread.Sleep(5 * 1000)

            ' Trigger an acknowledgeable event
            Try
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            _done = False
            Dim endTime As Date = Date.Now + New TimeSpan(0, 0, 5)
            Do While ((Not _done)) AndAlso (Date.Now < endTime)
                Thread.Sleep(1000)
            Loop

            ' Give some time to also receive the acknowledgement notification
            Thread.Sleep(5 * 1000)
```

```
            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            Console.WriteLine()
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Dim eventData As AEEventData = e.EventData
                Console.WriteLine("EventData.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("EventData.Message: {0}", eventData.Message)
                Console.WriteLine("EventData.Active: {0}", eventData.Active)
                Console.WriteLine("EventData.Acknowledged: {0}",
eventData.Acknowledged)
                Console.WriteLine("EventData.AcknowledgeRequired: {0}",
eventData.AcknowledgeRequired)

                If eventData.AcknowledgeRequired Then
                    Console.WriteLine(">>>>> ACKNOWLEDGING THIS EVENT")
                    Try
                        AEClient.AcknowledgeCondition("", "OPCLabs.KitEventServer.2",
"Simulation.ConditionState1", "Simulated", eventData.ActiveTime, eventData.Cookie)
                    Catch opcException As OpcException
                        Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                        Exit Sub
                    End Try
                    Console.WriteLine(">>>>> EVENT ACKNOWLEDGED")
                    _done = True
                End If
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to acknowledge an event condition in the OPC server.

Option Explicit

Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")

WScript.Echo "Hooking event handler..."
WScript.ConnectObject AEClient, "AEClient_"
```

```
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim SourceDescriptor: Set SourceDescriptor =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor.QualifiedName = "Simulation.ConditionState1"

WScript.Echo "Processing event notifications for 1 minute..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.Filter.Sources = Array(SourceDescriptor)
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = AEClient.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Give the refresh operation time to complete: Waiting for 5 seconds..."
WScript.Sleep 5*1000

WScript.Echo "Triggering an acknowledgeable event..."
On Error Resume Next
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim done: done = False
Dim endTime: endTime = Now() + 5*(1/24/60/60)
While (Not done) And (Now() < endTime)
    WScript.Sleep 1000
WEnd

WScript.Echo "Give some time to also receive the acknowledgement notification: Waiting
for 5 seconds..."
WScript.Sleep 5*1000

WScript.Echo "Unsubscribing events..."
AEClient.UnsubscribeEvents handle

WScript.Echo "Unhooking event handler..."
WScript.DisconnectObject AEClient

WScript.Echo "Finished."



Rem Notification event handler
Sub AEClient_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo
```

```
    WScript.Echo "Refresh: " & e.Refresh
    WScript.Echo "RefreshComplete: " & e.RefreshComplete
    If Not (e.EventData Is Nothing) Then
        With e.EventData
            WScript.Echo "EventData.QualifiedSourceName: " & .QualifiedSourceName
            WScript.Echo "EventData.Message: " & .Message
            WScript.Echo "EventData.Active: " & .Active
            WScript.Echo "EventData.Acknowledged: " & .Acknowledged
            WScript.Echo "EventData.AcknowledgeRequired: " & .AcknowledgeRequired

            If .AcknowledgeRequired Then
                WScript.Echo ">>>>> ACKNOWLEDGING THIS EVENT"
                On Error Resume Next
                AEClient.AcknowledgeCondition ServerDescriptor, SourceDescriptor,
"Simulated", _
                    .ActiveTime, .Cookie, "aUser", ""
                If Err.Number <> 0 Then
                    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
                    Exit Sub
                End If
                On Error Goto 0
                WScript.Echo ">>>>> EVENT ACKNOWLEDGED"
                done = True
            End If
        End With
    End If
End Sub
```

13.2.3.2 Examples - OPC Alarms&Events - Browse for areas

C#

```
// This example shows how to obtain all areas directly under the root (denoted by empty
string for the parent).

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class BrowseAreas
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AENodeElementCollection nodeElements;
            try
            {
                nodeElements = client.BrowseAreas("", "OPCLabs.KitEventServer.2", "");
            }
```

```
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AENodeElement nodeElement in nodeElements)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine("nodeElements[\"{0}\"]:", nodeElement.Name);
                Console.WriteLine("    .QualifiedName: {0}",
nodeElement.QualifiedName);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain all areas directly under the root (denoted by empty
string for the parent).

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class BrowseAreas
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim nodeElements As AENodeElementCollection
            Try
                nodeElements = client.BrowseAreas("", "OPCLabs.KitEventServer.2", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each nodeElement As AENodeElement In nodeElements
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine("nodeElements[""{0}""]:", nodeElement.Name)
                Console.WriteLine("    .QualifiedName: {0}", nodeElement.QualifiedName)
            Next nodeElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain all areas directly under the root (denoted by
```

```
empty string for the parent).

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseAreas("", "OPCLabs.KitEventServer.2",
"")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo "NodeElements(""" & NodeElement.Name & """):"
    With NodeElement
        WScript.Echo Space(4) & ".QualifiedName: " & .QualifiedName
    End With
Next
```

13.2.3.3 Examples - OPC Alarms&Events - Browse for servers

C#

```
// This example shows how to obtain all ProgIDs of all OPC Alarms and Events servers on
the local machine.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class BrowseServers
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            ServerElementCollection serverElements;
            try
            {
                serverElements = client.BrowseServers("");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
```

```
            foreach (ServerElement serverElement in serverElements)
            {
                Debug.Assert(serverElement != null);
                Console.WriteLine("serverElements[\"{0}\"].ProgId: {1}",
serverElement.Clsid, serverElement.ProgId);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain all ProgIDs of all OPC Alarms and Events servers on
the local machine.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class BrowseServers
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim serverElements As ServerElementCollection
            Try
                serverElements = client.BrowseServers("")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each serverElement As ServerElement In serverElements
                Debug.Assert(serverElement IsNot Nothing)
                Console.WriteLine("serverElements[""{0}""].ProgId: {1}",
serverElement.Clsid, serverElement.ProgId)
            Next serverElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain all ProgIDs of all OPC Alarms and Events servers
on the local machine.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim ServerElements: Set ServerElements = Client.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
```

```
End If
On Error Goto 0

Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "ServerElements(""" & ServerElement.UrlString & """).ProgId: " &
ServerElement.ProgId
Next
```

13.2.3.4 Examples - OPC Alarms&Events - Browse for sources

C#

```
// This example shows how to obtain all sources under the "Simulation" area.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class BrowseSources
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AENodeElementCollection nodeElements;
            try
            {
                nodeElements = client.BrowseSources("", "OPCLabs.KitEventServer.2",
"Simulation");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AENodeElement nodeElement in nodeElements)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine("nodeElements[\"{0}\"]:", nodeElement.Name);
                Console.WriteLine("    .QualifiedName: {0}",
nodeElement.QualifiedName);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain all sources under the "Simulation" area.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class BrowseSources
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim nodeElements As AENodeElementCollection
            Try
                nodeElements = client.BrowseSources("", "OPCLabs.KitEventServer.2",
"Simulation")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each nodeElement As AENodeElement In nodeElements
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine("nodeElements[""{0}""]:", nodeElement.Name)
                Console.WriteLine("    .QualifiedName: {0}", nodeElement.QualifiedName)
            Next nodeElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain all sources under the "Simulation" area.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseSources("",
"OPCLabs.KitEventServer.2", "Simulation")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo "NodeElements(""" & NodeElement.Name & """):"
    With NodeElement
        WScript.Echo Space(4) & ".QualifiedName: " & .QualifiedName
    End With
```

Next

13.2.3.5 Examples - OPC Alarms&Events - Callback using a lambda

C#

```
// This example shows how to subscribe to events and display the event message with
each notification, using a callback method
// specified using lambda expression.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    partial class SubscribeEvents
    {
        public static void CallbackLambda()
        {
            // Instantiate the client object
            var client = new EasyAEClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the event message
            client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
                (sender, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);
                    if (!eventArgs.Succeeded)
                    {
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                        return;
                    }
                    if (eventArgs.EventData != null)
                        Console.WriteLine(eventArgs.EventData.Message);
                });

            Console.WriteLine("Processing event notifications for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllEvents();

            Console.WriteLine("Waiting for 2 seconds...");
            Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events and display the event message with each
notification, using a callback method
' specified using lambda expression.

Imports OpcLabs.EasyOpc.AlarmsAndEvents

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
    Partial Friend Class SubscribeEvents
        Public Shared Sub CallbackLambda()
            ' Instantiate the client object
            Dim client = New EasyAEClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the event message
            client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
                    Sub(sender, eventArgs)
                        Debug.Assert(eventArgs IsNot Nothing)
                        If Not eventArgs.Succeeded Then
                            Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                            Exit Sub
                        End If
                        If eventArgs.EventData IsNot Nothing Then
                            Console.WriteLine(eventArgs.EventData.Message)
                        End If
                    End Sub)

            Console.WriteLine("Processing event notifications for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllEvents()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

13.2.3.6 Examples - OPC Alarms&Events - Change notification rate of a subscription

C#

```
// This example shows how to subscribe to events display the event message with each
notification. It also shows how to
// unsubscribe afterwards.

using System;
```

```
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class ChangeEventSubscription
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
500);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Changing subscription...");
                client.ChangeEventSubscription(handle, 5 * 1000);

                Console.WriteLine("Waiting for 50 seconds...");
                Thread.Sleep(50 * 1000);

                client.UnsubscribeEvents(handle);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events display the event message with each
notification. It also shows how to
' unsubscribe afterwards.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
```

```
Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class ChangeEventSubscription
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 500)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Changing subscription...")
                client.ChangeEventSubscription(handle, 5 * 1000)

                Console.WriteLine("Waiting for 50 seconds...")
                Thread.Sleep(50 * 1000)

                client.UnsubscribeEvents(handle)
            End Using
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to change the notification rate of an existing subscription.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 500
```

```
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Changing subscription..."
Client.ChangeEventSubscription handle, 5*1000, SubscriptionParameters.Filter, True

WScript.Echo "Waiting for 50 seconds..."
WScript.Sleep 50*1000

Client.UnsubscribeEvents handle



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

13.2.3.7 Examples - OPC Alarms&Events - Event pull

C#

```
// This example shows how to subscribe to events and obtain the notification events by
pulling them.

using System;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class PullNotification
    {
        public static void Main1()
        {
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            using (var client = new EasyAEClient { PullNotificationQueueCapacity = 1000
})
            {
                Console.WriteLine("Subscribing events...");
                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
1000);

                Console.WriteLine("Processing event notifications for 1 minute...");
```

```
                int endTick = Environment.TickCount + 60 * 1000;
                do
                {
                    EasyAENotificationEventArgs eventArgs = client.PullNotification(2 *
1000);
                    if (eventArgs != null)
                        // Handle the notification event
                        Console.WriteLine(eventArgs);
                } while (Environment.TickCount < endTick);

                Console.WriteLine("Unsubscribing events...");
                client.UnsubscribeEvents(handle);

                Console.WriteLine("Finished.");
            }
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events and obtain the notification events by
pulling them.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
    Partial Friend Class PullNotification
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                ' In order to use event pull, you must set a non-zero queue capacity
upfront.
                client.PullNotificationQueueCapacity = 1000

                Console.WriteLine("Subscribing events...")
                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Processing event notifications for 1 minute...")
                Dim endTick As Integer = Environment.TickCount + 60 * 1000
                Do
                    Dim eventArgs As EasyAENotificationEventArgs =
client.PullNotification(2 * 1000)
                    If Not eventArgs Is Nothing Then
                        ' Handle the notification event
                        Console.WriteLine(eventArgs)
                    End If
                Loop While Environment.TickCount < endTick

                Console.WriteLine("Unsubscribing events...")
                client.UnsubscribeEvents(handle)
            End Using

            Console.WriteLine("Finished.")
        End Sub
    End Class
```

```
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to events and obtain the notification events by
pulling them.

class procedure PullNotification.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyAEClient;
  EndTick: Cardinal;
  EventArgs: _EasyAENotificationEventArgs;
  Handle: Integer;
  ServerDescriptor: _ServerDescriptor;
  State: OleVariant;
  SubscriptionParameters: _AESubscriptionParameters;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitEventServer.2';

  // Instantiate the client object
  Client := CoEasyAEClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullNotificationQueueCapacity := 1000;

  WriteLn('Subscribing events...');
  SubscriptionParameters := CoAESubscriptionParameters.Create;
  SubscriptionParameters.NotificationRate := 1000;
  Handle := Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters, true,
State);

  WriteLn('Processing event notifications for 1 minute...');
  EndTick := Ticks + 60*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing events...');
  Client.UnsubscribeEvents(Handle);

  WriteLn('Finished.');
end;
```

PHP

```
// This example shows how to subscribe to events and obtain the notification events by
pulling them.

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitEventServer.2";

$Client = new COM("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
```

```
$Client->PullNotificationQueueCapacity = 1000;

print "Subscribing events...\n";
$SubscriptionParameters = new
COM("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters");
$SubscriptionParameters->NotificationRate = 1000;
$handle = $Client->SubscribeEvents($ServerDescriptor, $SubscriptionParameters, TRUE,
NULL);

print "Processing event notifications for 1 minute...\n";
$endTime = time() + 60;
do {
    $EventArgs = $Client->PullNotification(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        print $EventArgs->ToString();
        print "\n";
    }
} while (time() < $endTime);

print "Unsubscribing events...\n";
$Client->UnsubscribeEvents($handle);

print "Finished.\n";
```

VBScript

```
Rem This example shows how to subscribe to events and obtain the notification events by
pulling them.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullNotificationQueueCapacity = 1000

WScript.Echo "Subscribing events..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Processing event notifications for 1 minute..."
Dim endTime: endTime = Now() + 60*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullNotification(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime
```

```
WScript.Echo "Unsubscribing events..."
Client.UnsubscribeEvents handle

WScript.Echo "Finished."
```

13.2.3.8 Examples - OPC Alarms&Events - Filter events by category

C#

```
// This example shows how to filter the events by their category.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    partial class SubscribeEvents
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        public static void FilterByCategories()
        {
            var eventHandler = new
EasyAENotificationEventHandler(AEClient_Notification_FilterByCategories);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Categories = new long[] { 15531778 }
            };
            // You can also filter using event types, severity, areas, and sources.
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // Allow time for initial refresh
            Thread.Sleep(5 * 1000);

            // Set some events to active state.
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", true);

            Thread.Sleep(10 * 1000);

            AEClient.UnsubscribeEvents(handle);
```

```
        }

        // Notification event handler
        static void AEClient_Notification_FilterByCategories(object sender,
EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            Console.WriteLine(e);
            if (!e.Succeeded)
                return;

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.CategoryId: {0}", eventData.CategoryId);
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to filter the events by their category.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Partial Friend Class SubscribeEvents

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Public Shared Sub FilterByCategories()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification_FilterByCategories)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications...")
            Dim subscriptionFilter As New AESubscriptionFilter() With _
                {.Categories = New Long() {15531778}}
            ' You can also filter using event types, severity, areas, and sources.
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", Nothing, subscriptionFilter)

            ' Allow time for initial refresh
```

```
            Thread.Sleep(5 * 1000)

            ' Set some events to active state.
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
            DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True)

            Thread.Sleep(10 * 1000)

            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification_FilterByCategories(ByVal sender As
Object, ByVal e As EasyAENotificationEventArgs)
            Console.WriteLine()
            Console.WriteLine(e)
            If Not e.Succeeded Then
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            Dim eventData As AEEventData = e.EventData
            If e.EventData IsNot Nothing Then
                Console.WriteLine("Event.CategoryId: {0}", eventData.CategoryId)
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("Event.Message: {0}", eventData.Message)
                Console.WriteLine("Event.Active: {0}", eventData.Active)
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged)
            End If
        End Sub
    End Class
End Namespace
```

13.2.3.9 Examples - OPC Alarms&Events - Filter events by source

C#

```
// This example shows how to set the filtering criteria to be used for the event
subscription.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;
```

```
namespace DocExamples.AlarmsAndEvents._AESubscriptionFilter
{
    class Properties
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        public static void Main1()
        {
            var eventHandler = new
EasyAENotificationEventHandler(aeClient_Notification);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1",
"Simulation.ConditionState3" }
            };
            // You can also filter using event types, categories, severity, and areas.
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // Allow time for initial refresh
            Thread.Sleep(5 * 1000);

            // Set some events to active state.
            try
            {
                // The activation below will come from a source contained in a filter
and the notification will arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
                // The activation below will come from a source that is not contained
in a filter and the notification will not arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Thread.Sleep(10 * 1000);

            AEClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void aeClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
```

```
                return;
            }

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to set the filtering criteria to be used for the event
subscription.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._AESubscriptionFilter

    Friend Class Properties

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Public Shared Sub Main1()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications...")
            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1", "Simulation.ConditionState3"}
            ' You can also filter using event types, categories, severity, and areas.
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter)

            ' Allow time for initial refresh
            Thread.Sleep(5 * 1000)

            ' Set some events to active state.
            Try
                ' The activation below will come from a source contained in a filter
and the notification will arrive.
```

```
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
                ' The activation below will come from a source that is not contained in
a filter and the notification will not arrive.
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Thread.Sleep(10 * 1000)

            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            Console.WriteLine()
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Dim eventData As AEEventData = e.EventData
                Console.WriteLine("EventData.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("EventData.Message: {0}", eventData.Message)
                Console.WriteLine("EventData.Active: {0}", eventData.Active)
                Console.WriteLine("EventData.Acknowledged: {0}",
eventData.Acknowledged)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to set the filtering criteria to be used for the event
subscription.

Option Explicit

Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject AEClient, "AEClient_"

WScript.Echo "Processing event notifications..."
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"
```

```
Dim SubscriptionFilter: Set SubscriptionFilter =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionFilter")
Dim SourceDescriptor1: Set SourceDescriptor1 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor1.QualifiedName = "Simulation.ConditionState1"
Dim SourceDescriptor2: Set SourceDescriptor2 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor2.QualifiedName = "Simulation.ConditionState3"
SubscriptionFilter.Sources = Array(SourceDescriptor1, SourceDescriptor2)
Rem You can also filter using event types, categories, severity, and areas.
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.Filter = SubscriptionFilter
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = AEClient.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

Rem Allow time for initial refresh
WScript.Sleep 5*1000

WScript.Echo "Set some events to active state..."
On Error Resume Next
Rem The activation below will come from a source contained in a filter and the
notification will arrive.
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True
Rem The activation below will come from a source that is not contained in a filter and
the notification will not arrive.
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
AEClient.UnsubscribeEvents handle



Rem Notification event handler
Sub AEClient_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo
    WScript.Echo "Refresh: " & e.Refresh
    WScript.Echo "RefreshComplete: " & e.RefreshComplete

    If Not (e.EventData Is Nothing) Then
        With e.EventData
            WScript.Echo "EventData.QualifiedSourceName: " & .QualifiedSourceName
```

```
            WScript.Echo "EventData.Message: " & .Message
            WScript.Echo "EventData.Active: " & .Active
            WScript.Echo "EventData.Acknowledged: " & .Acknowledged
        End With
    End If
End Sub
```

13.2.3.10 Examples - OPC Alarms&Events - Get state of a condition

C#

```
// This example shows how to obtain current state information for the condition
instance corresponding to a Source and
// certain ConditionName.

using System;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class GetConditionState
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AEConditionState conditionState;
            try
            {
                conditionState = client.GetConditionState("",
"OPCLabs.KitEventServer.2",
                    "Simulation.ConditionState1", "Simulated");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("ConditionState:");
            Console.WriteLine("    .ActiveSubcondition: {0}",
conditionState.ActiveSubcondition);
            Console.WriteLine("    .Enabled: {0}", conditionState.Enabled);
            Console.WriteLine("    .Active: {0}", conditionState.Active);
            Console.WriteLine("    .Acknowledged: {0}", conditionState.Acknowledged);
            Console.WriteLine("    .Quality: {0}", conditionState.Quality);
            // Remark: IAEConditionState has many more properties
        }
    }
}
```

VB.NET

```
' This example shows how to obtain current state information for the condition instance
corresponding to a Source and
' certain ConditionName.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class GetConditionState
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim conditionState As AEConditionState
            Try
                conditionState = client.GetConditionState("",
"OPCLabs.KitEventServer.2", "Simulation.ConditionState1", "Simulated")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("ConditionState:")
            Console.WriteLine("    .ActiveSubcondition: {0}",
conditionState.ActiveSubcondition)
            Console.WriteLine("    .Enabled: {0}", conditionState.Enabled)
            Console.WriteLine("    .Active: {0}", conditionState.Active)
            Console.WriteLine("    .Acknowledged: {0}", conditionState.Acknowledged)
            Console.WriteLine("    .Quality: {0}", conditionState.Quality)
            ' Remark: IAEConditionState has many more properties
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to obtain current state information for the condition
instance corresponding to a Source and
Rem certain ConditionName.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim SourceDescriptor: Set SourceDescriptor =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor.QualifiedName = "Simulation.ConditionState1"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
```

```
Dim ConditionState: Set ConditionState = Client.GetConditionState(ServerDescriptor,
SourceDescriptor, "Simulated", Array())
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "ConditionState:"
With ConditionState
    WScript.Echo Space(4) & ".ActiveSubcondition: " & .ActiveSubcondition
    WScript.Echo Space(4) & ".Enabled: " & .Enabled
    WScript.Echo Space(4) & ".Active: " & .Active
    WScript.Echo Space(4) & ".Acknowledged: " & .Acknowledged
    WScript.Echo Space(4) & ".Quality: " & .Quality
    Rem Note that IAEConditionState has many more properties
End With
```

13.2.3.11 Examples - OPC Alarms&Events - Information about an event attribute

C#

```
// This example shows information available about OPC event attribute.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._AEAttributeElement
{
    class Properties
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
```

```
            {
                Debug.Assert(categoryElement != null);

                Console.WriteLine("Category {0}:", categoryElement);
                foreach (AEAttributeElement attributeElement in
categoryElement.AttributeElements)
                {
                    Debug.Assert(attributeElement != null);

                    Console.WriteLine("    Information about attribute {0}:",
attributeElement);
                    Console.WriteLine("        .AttributeId: {0}",
attributeElement.AttributeId);
                    Console.WriteLine("        .Description: {0}",
attributeElement.Description);
                    Console.WriteLine("        .DataType: {0}",
attributeElement.DataType);
                }
            }
        }
    }
}
```

VB.NET

```
' This example shows information available about OPC event attribute.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._AEAttributeElement

    Friend Class Properties
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)

                Console.WriteLine("Category {0}:", categoryElement)
                For Each attributeElement As AEAttributeElement In
categoryElement.AttributeElements
                    Debug.Assert(attributeElement IsNot Nothing)

                    Console.WriteLine("    Information about attribute {0}:",
attributeElement)
```

```
                    Console.WriteLine("        .AttributeId: {0}",
attributeElement.AttributeId)
                    Console.WriteLine("        .Description: {0}",
attributeElement.Description)
                    Console.WriteLine("        .DataType: {0}",
attributeElement.DataType)
                Next attributeElement
            Next categoryElement
        End Sub
    End Class

End Namespace
```

13.2.3.12 Examples - OPC Alarms&Events - Information about an event category

C#

```
// This example shows information available about OPC event category.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._AECategoryElement
{
    class Properties
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
            {
                Debug.Assert(categoryElement != null);
                Debug.Assert(categoryElement.AttributeElements.Keys != null);
                Debug.Assert(categoryElement.ConditionElements.Keys != null);
```

```
                Console.WriteLine("Information about category {0}:", categoryElement);
                Console.WriteLine("    .CategoryId: {0}", categoryElement.CategoryId);
                Console.WriteLine("    .Description: {0}",
categoryElement.Description);
                Console.WriteLine("    .ConditionElements:");
                foreach (string conditionKey in categoryElement.ConditionElements.Keys)
                    Console.WriteLine("        {0}", conditionKey);
                Console.WriteLine("    .AttributeElements:");
                foreach (long attributeKey in categoryElement.AttributeElements.Keys)
                    Console.WriteLine("        {0}", attributeKey);
            }
        }
    }
}
```

VB.NET

```
' This example shows information available about OPC event category.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._AECategoryElement

    Friend Class Properties
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)
                Debug.Assert(categoryElement.AttributeElements.Keys IsNot Nothing)
                Debug.Assert(categoryElement.ConditionElements.Keys IsNot Nothing)

                Console.WriteLine("Information about category {0}:", categoryElement)
                Console.WriteLine("    .CategoryId: {0}", categoryElement.CategoryId)
                Console.WriteLine("    .Description: {0}", categoryElement.Description)
                Console.WriteLine("    .ConditionElements:")
                For Each conditionKey As String In
categoryElement.ConditionElements.Keys
                    Console.WriteLine("        {0}", conditionKey)
                Next conditionKey
                Console.WriteLine("    .AttributeElements:")
                For Each attributeKey As Long In categoryElement.AttributeElements.Keys
                    Console.WriteLine("        {0}", attributeKey)
                Next attributeKey
            Next categoryElement
```

```
        End Sub
    End Class

End Namespace
```

13.2.3.13 Examples - OPC Alarms&Events - Information about an event condition

C#

```
// This example shows information available about OPC event condition.

using System;
using System.Diagnostics;
using System.Collections.Generic;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._AEConditionElement
{
    class Properties
    {
        static void DumpSubconditionNames(IEnumerable<string> subconditionNames)
        {
            foreach (string name in subconditionNames) Console.WriteLine("
{0}", name);
        }

        public static void Main1()
        {
            var client = new EasyAEClient();
            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
            {
                Debug.Assert(categoryElement != null);

                Console.WriteLine("Category {0}:", categoryElement);
                foreach (AEConditionElement conditionElement in
categoryElement.ConditionElements)
```

```
                {
                    Debug.Assert(conditionElement != null);

                    Console.WriteLine("    Information about condition \"{0}\":",
conditionElement);
                    Console.WriteLine("        .Name: {0}", conditionElement.Name);
                    Console.WriteLine("        .SubconditionNames:");
                    DumpSubconditionNames(conditionElement.SubconditionNames);
                }
            }
        }
    }
}
```

VB.NET

```
' This example shows information available about OPC event condition.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._AEConditionElement

    Friend Class Properties
        Private Shared Sub DumpSubconditionNames(ByVal subconditionNames As
IEnumerable(Of String))
            For Each name As String In subconditionNames
                Console.WriteLine("            {0}", name)
            Next name
        End Sub

        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)

                Console.WriteLine("Category {0}:", categoryElement)
                For Each conditionElement As AEConditionElement In
categoryElement.ConditionElements
                    Debug.Assert(conditionElement IsNot Nothing)

                    Console.WriteLine("    Information about condition ""{0}"":",
conditionElement)
                    Console.WriteLine("        .Name: {0}", conditionElement.Name)
                    Console.WriteLine("        .SubconditionNames:")
```

```
                    DumpSubconditionNames(conditionElement.SubconditionNames)
                Next conditionElement
            Next categoryElement
        End Sub
    End Class

End Namespace
```

13.2.3.14 Examples - OPC Alarms&Events - Query for event categories

C#

```
// This example shows how to enumerate all event categories provided by the OPC server.
For each category, it displays its Id
// and description.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class QueryEventCategories
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
            {
                Debug.Assert(categoryElement != null);
                Console.WriteLine("CategoryElements[\"{0}\"].Description: {1}",
categoryElement.CategoryId, categoryElement.Description);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to enumerate all event categories provided by the OPC server.
For each category, it displays its Id
' and description.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class QueryEventCategories
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"OPCLabs.KitEventServer.2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)
                Console.WriteLine("CategoryElements[""{0}""].Description: {1}",
categoryElement.CategoryId, categoryElement.Description)
            Next categoryElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to enumerate all event categories provided by the OPC
server. For each category, it displays its Id
Rem and description.

Option Explicit

Const AEEventTypes_All = 7

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim CategoryElements: Set CategoryElements =
Client.QueryEventCategories(ServerDescriptor, AEEventTypes_All)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
```

```
    WScript.Quit
End If
On Error Goto 0

Dim CategoryElement: For Each CategoryElement In CategoryElements
    WScript.Echo "CategoryElements(" & CategoryElement.CategoryId & ").Description: " &
CategoryElement.Description
Next
```

13.2.3.15 Examples - OPC Alarms&Events - Query for event conditions on a source

C#

```
// This example shows how to enumerate all event conditions associated with the
specified event source.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class QuerySourceConditions
    {
        public static void Main1()
        {
            var client = new EasyAEClient();
            AEConditionElementCollection conditionElements;
            try
            {
                conditionElements = client.QuerySourceConditions("",
"OPCLabs.KitEventServer.2",
                    "Simulation.ConditionState1");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AEConditionElement conditionElement in conditionElements)
            {
                Debug.Assert(conditionElement != null);
                Console.WriteLine("ConditionElements[\"{0}\"]: {1} subcondition(s)",
                    conditionElement.Name, conditionElement.SubconditionNames.Length);
            }
        }
    }
```

```
}
```

VB.NET

```
' This example shows how to enumerate all event conditions associated with the
specified event source.

Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class QuerySourceConditions
        Public Shared Sub Main1()
            Dim client = New EasyAEClient()

            Dim conditionElements As AEConditionElementCollection
            Try
                conditionElements = client.QuerySourceConditions( _
                    "", "OPCLabs.KitEventServer.2", "Simulation.ConditionState1")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each conditionElement As AEConditionElement In conditionElements
                Debug.Assert(conditionElement IsNot Nothing)
                Console.WriteLine("ConditionElements[""{0}""]: {1} subcondition(s)", _
                                  conditionElement.Name,
conditionElement.SubconditionNames.Length)
            Next conditionElement
        End Sub
    End Class

End Namespace
```

VBScript

```
Rem This example shows how to enumerate all event conditions associated with the
specified event source.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim SourceDescriptor: Set SourceDescriptor =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor.QualifiedName = "Simulation.ConditionState1"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
On Error Resume Next
Dim ConditionElements: Set ConditionElements =
Client.QuerySourceConditions(ServerDescriptor, SourceDescriptor)
```

```
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ConditionElement: For Each ConditionElement In ConditionElements
    WScript.Echo "ConditionElements(" & ConditionElement.Name & "): " &
(UBound(ConditionElement.SubconditionNames) + 1) & " subcondition(s)"
Next
```

13.2.3.16 Examples - OPC Alarms&Events - Refresh an event subscription

C#

```
// This example shows how to for a refresh for all active conditions and inactive,
unacknowledged conditions.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class RefreshEventSubscription
    {
        static readonly EasyAEClient AEClient = new EasyAEClient();
        static readonly EasyDAClient DAClient = new EasyDAClient();

        public static void Main1()
        {
            var eventHandler = new
EasyAENotificationEventHandler(AEClient_Notification);
            AEClient.Notification += eventHandler;

            Console.WriteLine("Processing event notifications...");
            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1",
"Simulation.ConditionState2", "Simulation.ConditionState3" }
            };
            int handle = AEClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter);

            // The component will perform auto-refresh at this point, give it time to
happen
            Console.WriteLine("Waiting for 10 seconds...");
            Thread.Sleep(10 * 1000);
```

```
            // Set some events to active state, which will cause them to appear in
refresh
            Console.WriteLine("Activating conditions and waiting for 10 seconds...");
            try
            {
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", true);
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
            Thread.Sleep(10 * 1000);

            Console.WriteLine("Refreshing subscription and waiting for 10 seconds...");
            AEClient.RefreshEventSubscription(handle);
            Thread.Sleep(10 * 1000);

            AEClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void AEClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            Console.WriteLine();
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }

            Console.WriteLine("Refresh: {0}", e.Refresh);
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete);
            AEEventData eventData = e.EventData;
            if (eventData != null)
            {
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName);
                Console.WriteLine("Event.Message: {0}", eventData.Message);
                Console.WriteLine("Event.Active: {0}", eventData.Active);
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to for a refresh for all active conditions and inactive,
unacknowledged conditions.

Imports System.Threading
```

```
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class RefreshEventSubscription

        _
        Private Shared ReadOnly AEClient As New EasyAEClient()

        _
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Public Shared Sub Main1()
            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
AEClient_Notification)
            AddHandler AEClient.Notification, eventHandler

            Console.WriteLine("Processing event notifications...")
            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1", "Simulation.ConditionState2",
"Simulation.ConditionState3"}
            Dim handle As Integer = AEClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter)

            ' The component will perform auto-refresh at this point, give it time to
happen
            Console.WriteLine("Waiting for 10 seconds...")
            Thread.Sleep(10 * 1000)

            ' Set some events to active state, which will cause them to appear in
refresh
            Console.WriteLine("Activating conditions and waiting for 10 seconds...")
            Try
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
                DAClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
            Thread.Sleep(10 * 1000)

            Console.WriteLine("Refreshing subscription and waiting for 10 seconds...")
            AEClient.RefreshEventSubscription(handle)
            Thread.Sleep(10 * 1000)

            AEClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub AEClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            Console.WriteLine()
```

```
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            Console.WriteLine("Refresh: {0}", e.Refresh)
            Console.WriteLine("RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Dim eventData As AEEventData = e.EventData
                Console.WriteLine("Event.QualifiedSourceName: {0}",
eventData.QualifiedSourceName)
                Console.WriteLine("Event.Message: {0}", eventData.Message)
                Console.WriteLine("Event.Active: {0}", eventData.Active)
                Console.WriteLine("Event.Acknowledged: {0}", eventData.Acknowledged)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to for a refresh for all active conditions and inactive,
unacknowledged conditions.

Option Explicit

Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim AEClient: Set AEClient =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject AEClient, "AEClient_"

WScript.Echo "Processing event notifications..."
Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"
Dim SourceDescriptor1: Set SourceDescriptor1 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor1.QualifiedName = "Simulation.ConditionState1"
Dim SourceDescriptor2: Set SourceDescriptor2 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor2.QualifiedName = "Simulation.ConditionState1"
Dim SourceDescriptor3: Set SourceDescriptor3 =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AENodeDescriptor")
SourceDescriptor3.QualifiedName = "Simulation.ConditionState1"
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.Filter.Sources = Array(SourceDescriptor1, SourceDescriptor2,
SourceDescriptor3)
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = AEClient.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

Rem The component will perform auto-refresh at this point, give it time to happen
WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

Rem Set some events to active state, which will cause them to appear in refresh
```

```
On Error Resume Next
WScript.Echo "Activating conditions and waiting for 10 seconds..."
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True
DAClient.WriteItemValue "", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState2.Activate", True
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Sleep 10*1000

WScript.Echo "Refreshing subscription and waiting for 10 seconds..."
AEClient.RefreshEventSubscription handle
WScript.Sleep 10*1000

AEClient.UnsubscribeEvents handle



Rem Notification event handler
Sub AEClient_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo
    WScript.Echo "Refresh: " & e.Refresh
    WScript.Echo "RefreshComplete: " & e.RefreshComplete

    If Not (e.EventData Is Nothing) Then
        With e.EventData
            WScript.Echo "EventData.QualifiedSourceName: " & .QualifiedSourceName
            WScript.Echo "EventData.Message: " & .Message
            WScript.Echo "EventData.Active: " & .Active
            WScript.Echo "EventData.Acknowledged: " & .Acknowledged
        End With
    End If
End Sub
```

13.2.3.17 Examples - OPC Alarms&Events - Retrieve attribute values

C#

```
// This example shows how to subscribe to events with specified event attributes, and
obtain the attribute values in event
// notifications.

using System;
```

```
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAENotificationEventArgs
{
    class AttributeValues
    {
        public static void Main1()
        {
            var aeClient = new EasyAEClient();
            var daClient = new EasyDAClient();

            var eventHandler = new
EasyAENotificationEventHandler(aeClient_Notification);
            aeClient.Notification += eventHandler;

            // Inactivate the event condition (we will later activate it and receive
the notification)
            daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Inactivate", true);

            var subscriptionFilter = new AESubscriptionFilter
            {
                Sources = new AENodeDescriptor[] { "Simulation.ConditionState1" }
            };

            // Prepare a dictionary holding requested event attributes for each event
category
            // The event category IDs and event attribute IDs are hard-coded here, but
can be obtained from the OPC
            // server by querying as well.
            var returnedAttributesByCategory = new AEAttributeSetDictionary
            {
                [0x00ECFF02] = new long[] {0x00EB0003, 0x00EB0008}
            };

            Console.WriteLine("Subscribing to events...");
            int handle = aeClient.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000,
null, subscriptionFilter,
                returnedAttributesByCategory);

            // Give the refresh operation time to complete
            Thread.Sleep(5 * 1000);

            // Trigger an event carrying specified attributes (activate the condition)
            try
            {
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
                    "SimulateEvents.ConditionState1.AttributeValues.15400963", 123456);
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
                    "SimulateEvents.ConditionState1.AttributeValues.15400968", "Some
string value");
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
```

```
"SimulateEvents.ConditionState1.Activate", true);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Processing event notifications for 10 seconds...");
            Thread.Sleep(10 * 1000);

            aeClient.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void aeClient_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (!e.Refresh && (e.EventData != null))
            {
                // Display all received event attribute IDs and their corresponding
values
                Console.WriteLine("Event attribute count: {0}",
e.EventData.AttributeValues.Count);
                foreach (KeyValuePair<long, object> pair in
e.EventData.AttributeValues)
                    Console.WriteLine("    {0}: {1}", pair.Key, pair.Value);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events with specified event attributes, and
obtain the attribute values in event
' notifications.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAENotificationEventArgs
    Friend Class AttributeValues
        Public Shared Sub Main1()
            Dim aeClient = New EasyAEClient()
            Dim daClient = New EasyDAClient()

            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
aeClient_Notification)
```

```
            AddHandler aeClient.Notification, eventHandler

            ' Inactivate the event condition (we will later activate it and receive the
notification)
            daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Inactivate", True)

            Dim subscriptionFilter As New AESubscriptionFilter
            subscriptionFilter.Sources = New AENodeDescriptor()
{"Simulation.ConditionState1"}

            ' Prepare a dictionary holding requested event attributes for each event
category
            ' The event category IDs and event attribute IDs are hard-coded here, but
can be obtained from the OPC
            ' server by querying as well.
            Dim returnedAttributesByCategory = New AEAttributeSetDictionary()
            returnedAttributesByCategory(&HECFF02) = New Long() {&HEB0003, &HEB0008}

            Console.WriteLine("Subscribing to events...")
            Dim handle As Integer = aeClient.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000, Nothing, subscriptionFilter,
returnedAttributesByCategory)

            ' Give the refresh operation time to complete
            Thread.Sleep(5 * 1000)

            ' Trigger an event carrying specified attributes (activate the condition)
            Try
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.AttributeValues.15400963", 123456)
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.AttributeValues.15400968", "Some string value")
                daClient.WriteItemValue("", "OPCLabs.KitServer.2",
"SimulateEvents.ConditionState1.Activate", True)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Processing event notifications for 10 seconds...")
            Thread.Sleep(10 * 1000)

            aeClient.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub aeClient_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If

            If (Not e.Refresh) AndAlso (e.EventData IsNot Nothing) Then
                ' Display all received event attribute IDs and their corresponding
```

```
values
                Console.WriteLine("Event attribute count: {0}",
e.EventData.AttributeValues.Count)
                For Each pair As KeyValuePair(Of Long, Object) In
e.EventData.AttributeValues
                    Console.WriteLine("    {0}: {1}", pair.Key, pair.Value)
                Next pair
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example subscribe to events, and displays rich information available with each
event notification.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Processing event notifications for 1 minute..."
WScript.Sleep 60*1000

Client.UnsubscribeEvents handle



Rem Notification event handler
Sub Client_Notification(Sender, e)
    On Error Resume Next
    WScript.Echo
    WScript.Echo "e.Exception.Message: " & e.Exception.Message
    WScript.Echo "e.Exception.Source: " & e.Exception.Source
    WScript.Echo "e.Exception.ErrorCode: " & e.Exception.ErrorCode
    WScript.Echo "e.Arguments.State: " & e.Arguments.State
    WScript.Echo "e.Arguments.ServerDescriptor.MachineName: " &
e.Arguments.ServerDescriptor.MachineName
    WScript.Echo "e.Arguments.ServerDescriptor.ServerClass: " &
e.Arguments.ServerDescriptor.ServerClass
    WScript.Echo "e.Arguments.SubscriptionParameters.Active: " &
e.Arguments.SubscriptionParameters.Active
    WScript.Echo "e.Arguments.SubscriptionParameters.NotificationRate: " &
e.Arguments.SubscriptionParameters.NotificationRate
    Rem IMPROVE: Display Arguments.SubscriptionParameters.Filter details
```

```
    WScript.Echo "e.Arguments.SubscriptionParameters.Filter: " &
e.Arguments.SubscriptionParameters.Filter
    Rem IMPROVE: Display Arguments.SubscriptionParameters.ReturnedAttributesByCategory
details
    WScript.Echo "e.Arguments.SubscriptionParameters.ReturnedAttributesByCategory: " &
e.Arguments.SubscriptionParameters.ReturnedAttributesByCategory
    WScript.Echo "e.Refresh: " & e.Refresh
    WScript.Echo "e.RefreshComplete: " & e.RefreshComplete
    WScript.Echo "e.EnabledChanged: " & e.EnabledChanged
    WScript.Echo "e.ActiveChanged: " & e.ActiveChanged
    WScript.Echo "e.AcknowledgedChanged: " & e.AcknowledgedChanged
    WScript.Echo "e.QualityChanged: " & e.QualityChanged
    WScript.Echo "e.SeverityChanged: " & e.SeverityChanged
    WScript.Echo "e.SubconditionChanged: " & e.SubconditionChanged
    WScript.Echo "e.MessageChanged: " & e.MessageChanged
    WScript.Echo "e.AttributeChanged: " & e.AttributeChanged
    WScript.Echo "e.EventData.QualifiedSourceName: " & e.EventData.QualifiedSourceName
    WScript.Echo "e.EventData.Time: " & e.EventData.Time
    WScript.Echo "e.EventData.TimeLocal: " & e.EventData.TimeLocal
    WScript.Echo "e.EventData.Message: " & e.EventData.Message
    WScript.Echo "e.EventData.EventType: " & e.EventData.EventType
    WScript.Echo "e.EventData.CategoryId: " & e.EventData.CategoryId
    WScript.Echo "e.EventData.Severity: " & e.EventData.Severity
    Rem IMPROVE: Display EventData.AttributeValues details
    WScript.Echo "e.EventData.AttributeValues: " & e.EventData.AttributeValues
    WScript.Echo "e.EventData.ConditionName: " & e.EventData.ConditionName
    WScript.Echo "e.EventData.SubconditionName: " & e.EventData.SubconditionName
    WScript.Echo "e.EventData.Enabled: " & e.EventData.Enabled
    WScript.Echo "e.EventData.Active: " & e.EventData.Active
    WScript.Echo "e.EventData.Acknowledged: " & e.EventData.Acknowledged
    WScript.Echo "e.EventData.Quality: " & e.EventData.Quality
    WScript.Echo "e.EventData.AcknowledgeRequired: " & e.EventData.AcknowledgeRequired
    WScript.Echo "e.EventData.ActiveTime: " & e.EventData.ActiveTime
    WScript.Echo "e.EventData.ActiveTimeLocal: " & e.EventData.ActiveTimeLocal
    WScript.Echo "e.EventData.Cookie: " & e.EventData.Cookie
    WScript.Echo "e.EventData.ActorId: " & e.EventData.ActorId
End Sub
```

13.2.3.18 Examples - OPC Alarms&Events - Subscribe to events

C#

```
// This example shows how to subscribe to events and display the event message with
each notification. It also shows how to
// unsubscribe afterwards.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
```

```
namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    partial class SubscribeEvents
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
1000);

                Console.WriteLine("Processing event notifications for 1 minute...");
                Thread.Sleep(60 * 1000);

                client.UnsubscribeEvents(handle);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to events and display the event message with each
notification. It also shows how to
' unsubscribe afterwards.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient
    Partial Friend Class SubscribeEvents
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000)
```

```
                Console.WriteLine("Processing event notifications for 1 minute...")
                Thread.Sleep(60 * 1000)

                client.UnsubscribeEvents(handle)
            End Using
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to events display the event message with each
notification. It also shows how to
// unsubscribe afterwards.

type
  TClientEventHandlers = class
    // Notification event handler
    procedure OnNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyAENotificationEventArgs);
  end;

procedure TClientEventHandlers.OnNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyAENotificationEventArgs);
begin
  if not eventArgs.Succeeded then
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
  if eventArgs.EventData <> nil then
      WriteLn(eventArgs.EventData.Message);
end;

class procedure SubscribeEvents.Main;
var
  Client: TEasyAEClient;
  ClientEventHandlers: TClientEventHandlers;
  Handle: Integer;
  ServerDescriptor: _ServerDescriptor;
  State: OleVariant;
  SubscriptionParameters: _AESubscriptionParameters;
begin
  ServerDescriptor := CoServerDescriptor.Create;
```

```
  ServerDescriptor.ServerClass := 'OPCLabs.KitEventServer.2';

  // Instantiate the client object and hook events
  Client := TEasyAEClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers.Create;
  Client.OnNotification := ClientEventHandlers.OnNotification;

  WriteLn('Subscribing events...');
  SubscriptionParameters := CoAESubscriptionParameters.Create;
  SubscriptionParameters.NotificationRate := 1000;
  Handle := Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters, true,
State);

  WriteLn('Processing event notifications for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing events...');
  Client.UnsubscribeEvents(Handle);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to events display the event message with each
notification. It also shows how to
// unsubscribe afterwards.

class DEasyEAClientEvents {
    function Notification($Sender, $E)
    {
        if (!($E->Succeeded))
        {
            printf("*** Failure: %s\n", $E->ErrorMessageBrief);
            Exit();
        }

        if (!is_null($E->EventData))
        {
            print $E->EventData->Message;
            print "\n";
        }
    }
}

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitEventServer.2";

$Client = new COM("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient");
$Events = new DEasyEAClientEvents();
com_event_sink($Client, $Events, "DEasyEAClientEvents");

print "Subscribing events...\n";
$SubscriptionParameters = new
COM("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters");
```

```
$SubscriptionParameters->NotificationRate = 1000;
$handle = $Client->SubscribeEvents($ServerDescriptor, $SubscriptionParameters, TRUE,
NULL);

print "Processing event notifications for 1 minute...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);

print "Unsubscribing events...\n";
$Client->UnsubscribeEvents($handle);

print "Finished.\n";
```

VBScript

```
Rem This example shows how to subscribe to events display the event message with each
notification. It also shows how to
Rem unsubscribe afterwards.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing events..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Processing event notifications for 1 minute..."
WScript.Sleep 60*1000

WScript.Echo "Unsubscribing events..."
Client.UnsubscribeEvents handle

WScript.Echo "Finished."



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not e.Succeeded Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

13.2.3.19 Examples - OPC Alarms&Events - Unsubscribe from a single event

C#

```
// This example shows how to unsubscribe from specific event notifications.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class UnsubscribeEvents
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeEvents("", "OPCLabs.KitEventServer.2",
1000);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeEvents(handle);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to unsubscribe from specific event notifications.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class UnsubscribeEvents
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeEvents("",
"OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeEvents(handle)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to unsubscribe from specific event notifications.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"
```

```
WScript.Echo "Subscribing..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Dim handle: handle = Client.SubscribeEvents(ServerDescriptor, SubscriptionParameters,
True, Nothing)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeEvents handle

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

13.2.3.20 Examples - OPC Alarms&Events - Unsubscribe from all events

C#

```
// This example shows how to unsubscribe from all event notifications.

using System;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;

namespace DocExamples.AlarmsAndEvents._EasyAEClient
{
    class UnsubscribeAllEvents
    {
        public static void Main1()
        {
            using (var client = new EasyAEClient())
            {
                var eventHandler = new
EasyAENotificationEventHandler(client_Notification);
                client.Notification += eventHandler;
```

```
                Console.WriteLine("Subscribing...");
                client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllEvents();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
        {
            if (!e.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
                return;
            }
            if (e.EventData != null)
                Console.WriteLine(e.EventData.Message);
        }
    }
}
```

VB.NET

```
' This example shows how to unsubscribe from all event notifications.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel

Namespace DocExamples.AlarmsAndEvents._EasyAEClient

    Friend Class UnsubscribeAllEvents
        Public Shared Sub Main1()
            Using client = New EasyAEClient()
                Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
                AddHandler client.Notification, eventHandler

                Console.WriteLine("Subscribing...")
                client.SubscribeEvents("", "OPCLabs.KitEventServer.2", 1000)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeAllEvents()

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
```

```
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If Not e.Succeeded Then
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
                Exit Sub
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine(e.EventData.Message)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to unsubscribe from all event notifications.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitEventServer.2"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim SubscriptionParameters: Set SubscriptionParameters =
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.AESubscriptionParameters")
SubscriptionParameters.NotificationRate = 1000
Client.SubscribeEvents ServerDescriptor, SubscriptionParameters, True, Nothing

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllEvents

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Rem Notification event handler
Sub Client_Notification(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    If Not e.EventData Is Nothing Then WScript.Echo e.EventData.Message
End Sub
```

13.2.4 Examples - OPC Classic Specialized

13.2.4.1 Examples - OPC Classic Specialized - Schneider Electric OPC Factory Server - Subscribe to multiple items

C#

```
// This example shows how subscribing to items in Schneider Electric OPC Factory
Server.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.Specialized
{
    class SchneiderElectric_OpcFactoryServer
    {
        public static void SubscribeMultipleItems()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_Main1_ItemChanged;

                client.SubscribeMultipleItems(
                    new[] {
                                new DAItemGroupArguments("", "Schneider-Aut.OFS", "
<<system>>!#OFSStatus", 1000, null),
                                new DAItemGroupArguments("", "Schneider-Aut.OFS", "
<<system>>!#ClientAlive", 1000, null),
                                new DAItemGroupArguments("", "Schneider-Aut.OFS", "
<<system>>!#OFSQualStatus", 1000, null),
                                new DAItemGroupArguments("", "Schneider-Aut.OFS",
"DevExample_1!#MastPeriod", 1000, null),
                                new DAItemGroupArguments("", "Schneider-Aut.OFS",
"DevExample_1!#WatchDogValue", 1000, null),
                                new DAItemGroupArguments("", "Schneider-Aut.OFS",
"DevExample_1!#DiagBuffConf", 1000, null),
                                new DAItemGroupArguments("", "Schneider-Aut.OFS",
"DevExample_1!#DiagBuffFull", 1000, null)
                            });

                Console.WriteLine("Processing item changed events for 1 minute...");
                Thread.Sleep(60 * 1000);
            }
        }

        // Item changed event handler
        static void client_Main1_ItemChanged(object sender, EasyDAItemChangedEventArgs
e)
```

```
        {
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId,
e.Vtq);
            else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief);
        }
    }
}
```

13.2.4.2 Examples - OPC Classic Specialized - Software Toolbox TOP Server - A&E

C#

```
// This example shows how to work with Software Tolbox TOP Server 5 Alarms and Events.
// Use simdemo_WithA&E.opf configuration file and write a value above 1000 to
Channel1.Device1.Tag1 or Channel1.Device1.Tag2.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.AlarmsAndEvents;
using OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace;
using OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.AlarmsAndEvents.SWToolbox
{
    class TOPServer_AE
    {
        public static void Main1()
        {
            var client = new EasyAEClient();

            // Browse for some areas and sources

            AENodeElementCollection areaElements;
            try
            {
                areaElements = client.BrowseAreas("", "SWToolbox.TOPServer_AE.V5", "");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AENodeElement areaElement in areaElements)
            {
                Debug.Assert(areaElement != null);
                Debug.Assert(areaElement.QualifiedName != null);
```

```
                Console.WriteLine("areaElements[\"{0}\"]:", areaElement.Name);
                Console.WriteLine("    .QualifiedName: {0}",
areaElement.QualifiedName);

                AENodeElementCollection sourceElements =
                    client.BrowseSources("", "SWToolbox.TOPServer_AE.V5",
areaElement.QualifiedName);
                foreach (AENodeElement sourceElement in sourceElements)
                {
                    Debug.Assert(sourceElement != null);

                    Console.WriteLine("    sourceElements[\"{0}\"]:",
sourceElement.Name);
                    Console.WriteLine("        .QualifiedName: {0}",
sourceElement.QualifiedName);
                }
            }

            // Query for event categories

            AECategoryElementCollection categoryElements;
            try
            {
                categoryElements = client.QueryEventCategories("",
"SWToolbox.TOPServer_AE.V5");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            foreach (AECategoryElement categoryElement in categoryElements)
            {
                Debug.Assert(categoryElement != null);
                Console.WriteLine("CategoryElements[\"{0}\"].Description: {1}",
categoryElement.CategoryId,
                                  categoryElement.Description);
            }

            // Subscribe to events, wait, and unsubscribe

            var eventHandler = new EasyAENotificationEventHandler(client_Notification);
            client.Notification += eventHandler;

            int handle = client.SubscribeEvents("", "SWToolbox.TOPServer_AE.V5", 1000);

            Console.WriteLine("Processing event notifications for 1 minute...");
            Thread.Sleep(60 * 1000);

            client.UnsubscribeEvents(handle);
        }

        // Notification event handler
        static void client_Notification(object sender, EasyAENotificationEventArgs e)
```

```
        {
            if (e.Exception != null) Console.WriteLine("e.Exception.Message: {0}",
e.Exception.Message);
            if (e.Exception != null) Console.WriteLine("e.Exception.Source: {0}",
e.Exception.Source);
            Console.WriteLine("e.Refresh: {0}", e.Refresh);
            Console.WriteLine("e.RefreshComplete: {0}", e.RefreshComplete);
            if (e.EventData != null)
Console.WriteLine("e.EventData.QualifiedSourceName: {0}",
e.EventData.QualifiedSourceName);
            if (e.EventData != null) Console.WriteLine("e.EventData.Time: {0}",
e.EventData.Time);
            if (e.EventData != null) Console.WriteLine("e.EventData.Message: {0}",
e.EventData.Message);
            if (e.EventData != null) Console.WriteLine("e.EventData.EventType: {0}",
e.EventData.EventType);
            if (e.EventData != null) Console.WriteLine("e.EventData.CategoryId: {0}",
e.EventData.CategoryId);
            if (e.EventData != null) Console.WriteLine("e.EventData.Severity: {0}",
e.EventData.Severity);
            if (e.EventData != null) Console.WriteLine("e.EventData.ConditionName:
{0}", e.EventData.ConditionName);
            if (e.EventData != null) Console.WriteLine("e.EventData.SubconditionName:
{0}", e.EventData.SubconditionName);
            if (e.EventData != null) Console.WriteLine("e.EventData.Enabled: {0}",
e.EventData.Enabled);
            if (e.EventData != null) Console.WriteLine("e.EventData.Active: {0}",
e.EventData.Active);
            if (e.EventData != null) Console.WriteLine("e.EventData.Acknowledged: {0}",
e.EventData.Acknowledged);
            if (e.EventData != null) Console.WriteLine("e.EventData.Quality: {0}",
e.EventData.Quality);
            if (e.EventData != null)
Console.WriteLine("e.EventData.AcknowledgeRequired: {0}",
e.EventData.AcknowledgeRequired);
            if (e.EventData != null) Console.WriteLine("e.EventData.ActiveTime: {0}",
e.EventData.ActiveTime);
        }
    }
}
```

VB.NET

```
' This example shows how to work with Software Tolbox TOP Server 5 Alarms and Events.
' Use simdemo_WithA&E.opf configuration file and write a value above 1000 to
Channel1.Device1.Tag1 or Channel1.Device1.Tag2.

Imports System.Threading
Imports OpcLabs.EasyOpc.AlarmsAndEvents
Imports OpcLabs.EasyOpc.AlarmsAndEvents.AddressSpace
Imports OpcLabs.EasyOpc.AlarmsAndEvents.OperationModel
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.AlarmsAndEvents.SWToolbox

    Friend Class TOPServer_AE
        Public Shared Sub Main1()
```

```
            Dim client = New EasyAEClient()

            ' Browse for some areas and sources

            Dim areaElements As AENodeElementCollection
            Try
                areaElements = client.BrowseAreas("", "SWToolbox.TOPServer_AE.V5", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each areaElement As AENodeElement In areaElements
                Debug.Assert(areaElement IsNot Nothing)
                Debug.Assert(areaElement.QualifiedName IsNot Nothing)

                Console.WriteLine("areaElements[""{0}""]:", areaElement.Name)
                Console.WriteLine("    .QualifiedName: {0}", areaElement.QualifiedName)

                Dim sourceElements As AENodeElementCollection =
client.BrowseSources("", "SWToolbox.TOPServer_AE.V5", areaElement.QualifiedName)
                For Each sourceElement As AENodeElement In sourceElements
                    Debug.Assert(sourceElement IsNot Nothing)

                    Console.WriteLine("    sourceElements[""{0}""]:",
sourceElement.Name)
                    Console.WriteLine("        .QualifiedName: {0}",
sourceElement.QualifiedName)
                Next sourceElement
            Next areaElement

            ' Query for event categories

            Dim categoryElements As AECategoryElementCollection
            Try
                categoryElements = client.QueryEventCategories("",
"SWToolbox.TOPServer_AE.V5")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            For Each categoryElement As AECategoryElement In categoryElements
                Debug.Assert(categoryElement IsNot Nothing)
                Console.WriteLine("CategoryElements[""{0}""].Description: {1}",
categoryElement.CategoryId, categoryElement.Description)
            Next categoryElement

            ' Subscribe to events, wait, and unsubscribe

            Dim eventHandler = New EasyAENotificationEventHandler(AddressOf
client_Notification)
            AddHandler client.Notification, eventHandler

            Dim handle As Integer = client.SubscribeEvents("",
```

```
"SWToolbox.TOPServer_AE.V5", 1000)

            Console.WriteLine("Processing event notifications for 1 minute...")
            Thread.Sleep(60 * 1000)

            client.UnsubscribeEvents(handle)
        End Sub

        ' Notification event handler
        Private Shared Sub client_Notification(ByVal sender As Object, ByVal e As
EasyAENotificationEventArgs)
            If e.Exception IsNot Nothing Then
                Console.WriteLine("e.Exception.Message: {0}", e.Exception.Message)
            End If
            If e.Exception IsNot Nothing Then
                Console.WriteLine("e.Exception.Source: {0}", e.Exception.Source)
            End If
            Console.WriteLine("e.Refresh: {0}", e.Refresh)
            Console.WriteLine("e.RefreshComplete: {0}", e.RefreshComplete)
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.QualifiedSourceName: {0}",
e.EventData.QualifiedSourceName)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Time: {0}", e.EventData.Time)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Message: {0}", e.EventData.Message)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.EventType: {0}", e.EventData.EventType)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.CategoryId: {0}",
e.EventData.CategoryId)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Severity: {0}", e.EventData.Severity)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.ConditionName: {0}",
e.EventData.ConditionName)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.SubconditionName: {0}",
e.EventData.SubconditionName)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Enabled: {0}", e.EventData.Enabled)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Active: {0}", e.EventData.Active)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Acknowledged: {0}",
e.EventData.Acknowledged)
            End If
```

```
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.Quality: {0}", e.EventData.Quality)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.AcknowledgeRequired: {0}",
e.EventData.AcknowledgeRequired)
            End If
            If e.EventData IsNot Nothing Then
                Console.WriteLine("e.EventData.ActiveTime: {0}",
e.EventData.ActiveTime)
            End If
        End Sub
    End Class

End Namespace
```

13.2.5 Examples - OPC Data Access

13.2.5.1 Examples - OPC Data Access - Browse for access paths

VBScript

```
Rem This example shows how to obtain all access paths available for an item.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.Random"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim accessPaths: accessPaths = Client.BrowseAccessPaths(ServerDescriptor,
NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim i: For i = LBound(accessPaths) To UBound(accessPaths)
    WScript.Echo "accessPaths(" & i & "): " & accessPaths(i)
Next
```

13.2.5.2 Examples - OPC Data Access - Browse for branches

VBScript

```
Rem This example shows how to obtain all branches at the root of the address space. For
each branch, it displays whether
Rem it may have child nodes.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim BranchElements: Set BranchElements = Client.BrowseBranches("",
"OPCLabs.KitServer.2", "")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim BranchElement: For Each BranchElement In BranchElements
    WScript.Echo "BranchElements(""" & BranchElement.Name & """).HasChildren: " &
BranchElement.HasChildren
Next
```

13.2.5.3 Examples - OPC Data Access - Browse for leaves

VBScript

```
Rem This example shows how to obtain all leaves under the "Simulation" branch of the
address space. For each leaf, it displays
Rem the ItemID of the node.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim LeafElements: Set LeafElements = Client.BrowseLeaves("", "OPCLabs.KitServer.2",
"Simulation")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim LeafElement: For Each LeafElement In LeafElements
    WScript.Echo "LeafElements(""" & LeafElement.Name & """).ItemId: " &
LeafElement.ItemId
Next
```

13.2.5.4 Examples - OPC Data Access - Browse for nodes

Object Pascal

```
// This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
// whether the node is a branch or a leaf.

class procedure BrowseNodes.Main;
var
  BrowseParameters: _DABrowseParameters;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Count: Cardinal;
  Element: OleVariant;
  ServerDescriptor: _ServerDescriptor;
  NodeDescriptor: _DANodeDescriptor;
  NodeElement: _DANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _DANodeElementCollection;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';

  NodeDescriptor := CoDANodeDescriptor.Create;
  NodeDescriptor.ItemId := 'Simulation';

  BrowseParameters := CoDABrowseParameters.Create;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    NodeElements := Client.BrowseNodes(
      ServerDescriptor,
      NodeDescriptor,
      BrowseParameters);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _DANodeElement;
//    WriteLn(NodeElement.Name, ': ', NodeElement.ItemId);
    WriteLn('BrowseElements("', NodeElement.Name, '"):');
    WriteLn('    .IsBranch: ', NodeElement.IsBranch);
    WriteLn('    .IsLeaf: ', NodeElement.IsLeaf);
  end;
end;
```

PHP

```
// This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
// whether the node is a branch or a leaf.

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";

$NodeDescriptor = new COM("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor");
$NodeDescriptor->ItemID = "Simulation";

$BrowseParameters = new COM("OpcLabs.EasyOpc.DataAccess.DABrowseParameters");

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $NodeElements = $Client->BrowseNodes($ServerDescriptor, $NodeDescriptor,
$BrowseParameters);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

foreach ($NodeElements as $NodeElement)
{
    printf("NodeElements(\"%s\"):\n", $NodeElement->Name);
    printf("    .IsBranch: %s\n", $NodeElement->IsBranch ? 'true' : 'false');
    printf("    .IsLeaf: %s\n", $NodeElement->IsLeaf ? 'true' : 'false');
}
```

PowerScript

```
// This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
// whether the node is a branch or a leaf.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare the arguments

OLEObject serverDescriptor
serverDescriptor = CREATE OLEObject
serverDescriptor.ConnectToNewObject("OpcLabs.EasyOpc.ServerDescriptor")
serverDescriptor.ServerClass = "OPCLabs.KitServer.2"

OLEObject nodeDescriptor
nodeDescriptor = CREATE OLEObject
```

```
nodeDescriptor.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
nodeDescriptor.ItemID = "Simulation"

OLEObject browseParameters
browseParameters = CREATE OLEObject
browseParameters.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.DABrowseParameters")

// Perform the operation
OLEObject nodeElements
TRY
    nodeElements = client.BrowseNodes(serverDescriptor, nodeDescriptor,
browseParameters)
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

OLEObject nodeElementList
nodeElementList = nodeElements.ToList()

// Display results
Int i
FOR i = 0 TO nodeElementList.Count - 1
    OLEObject nodeElement
    nodeElement = nodeElementList.Item[i]
    mle_outputtext.Text = mle_outputtext.Text + 'NodeElements("' + nodeElement.Name +
'"):' + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "    nodeElement.IsBranch: " +
String(nodeElement.IsBranch) + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "    nodeElement.IsLeaf: " +
String(nodeElement.IsLeaf) + "~r~n"
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

VBScript

```
Rem This example shows how to obtain all nodes under the "Simulation" branch of the
address space. For each node, it displays
Rem whether the node is a branch or a leaf.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation"

Dim BrowseParameters: Set BrowseParameters =
CreateObject("OpcLabs.EasyOpc.DataAccess.DABrowseParameters")

Dim Client: Set Client= CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
```

```
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseNodes(ServerDescriptor,
NodeDescriptor, BrowseParameters)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo "NodeElements(""" & NodeElement.Name & """):"
    With NodeElement
        WScript.Echo Space(4) & ".IsBranch: " & .IsBranch
        WScript.Echo Space(4) & ".IsLeaf: " & .IsLeaf
    End With
Next
```

13.2.5.5 Examples - OPC Data Access - Browse for properties

Object Pascal

```
// This example shows how to enumerate all properties of an OPC item. For each
property, it displays its Id and description.

class procedure BrowseProperties.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Count: Cardinal;
  Element: OleVariant;
  ServerDescriptor: _ServerDescriptor;
  NodeDescriptor: _DANodeDescriptor;
  PropertyElement: _DAPropertyElement;
  PropertyElementEnumerator: IEnumVariant;
  PropertyElements: _DAPropertyElementCollection;
begin
  ServerDescriptor := CoServerDescriptor.Create;
  ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';

  NodeDescriptor := CoDANodeDescriptor.Create;
  NodeDescriptor.ItemId := 'Simulation.Random';

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    PropertyElements := Client.BrowseProperties(
      ServerDescriptor,
      NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
```

```
  PropertyElementEnumerator := PropertyElements.GetEnumerator;
  while (PropertyElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    PropertyElement := IUnknown(Element) as _DAPropertyElement;
    WriteLn('PropertyElements("', PropertyElement.PropertyId.NumericalValue,
'").Description: ', PropertyElement.Description);
  end;
end;
```

PHP

```
// This example shows how to enumerate all properties of an OPC item. For each
property, it displays its Id and description.

$ServerDescriptor = new COM("OpcLabs.EasyOpc.ServerDescriptor");
$ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";

$NodeDescriptor = new COM("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor");
$NodeDescriptor->ItemID = "Simulation.Random";

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $PropertyElements = $Client->BrowseProperties($ServerDescriptor, $NodeDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

foreach ($PropertyElements as $PropertyElement)
{
    printf("PropertyElements(\"%s\").Description: %s\n", $PropertyElement->PropertyID-
>NumericalValue, $PropertyElement->Description);
}
```

VBScript

```
Rem This example shows how to enumerate all properties of an OPC item. For each
property, it displays its Id and description.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.Random"

Dim EasyDAClient: Set EasyDAClient =
CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
```

```
Dim PropertyElements: Set PropertyElements =
EasyDAClient.BrowseProperties(ServerDescriptor, NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim PropertyElement: For Each PropertyElement In PropertyElements
    WScript.Echo "PropertyElements(""" & PropertyElement.PropertyID.NumericalValue &
""").Description: " & PropertyElement.Description
Next
```

13.2.5.6 Examples - OPC Data Access - Browse for servers

JScript

```
// This example shows how to obtain all ProgIDs of all OPC Data Access servers on the
local machine.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
var ServerElements = Client.BrowseServers("")

for (var objEnum = new Enumerator(ServerElements) ; !objEnum.atEnd() ;
objEnum.moveNext()) {
    var ServerElement = objEnum.item();
    WScript.Echo("ServerElements(\"" + ServerElement.UrlString + "\").ProgId: " +
ServerElement.ProgId);
}
```

PHP

```
// This example shows how to obtain all ProgIDs of all OPC Data Access servers on the
local machine.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $ServerElements = $Client->BrowseServers("");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

foreach ($ServerElements as $ServerElement)
{
    printf("ServerElements(\"%s\").ProgId: %s\n", $ServerElement->ClsidString,
$ServerElement->ProgId);
}
```

VBScript

```
Rem This example shows how to obtain all ProgIDs of all OPC Data Access servers on the
local machine.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim ServerElements: Set ServerElements = Client.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "ServerElements(""" & ServerElement.ClsidString & """).ProgId: " &
ServerElement.ProgId
Next
```

VBScript

```
Rem This example shows all information available about categories that particular OPC
servers do support.

Option Explicit

Sub DumpServerElements(ByVal ServerElements)
Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "Categories of """ & ServerElement.ProgID & """:"
    With ServerElement.ServerCategories
        WScript.Echo Space(4) & ".OpcAlarmsAndEvents10: " & .OpcAlarmsAndEvents10
        WScript.Echo Space(4) & ".OpcDataAccess10: " & .OpcDataAccess10
        WScript.Echo Space(4) & ".OpcDataAccess20: " & .OpcDataAccess20
        WScript.Echo Space(4) & ".OpcDataAccess30: " & .OpcDataAccess30
        WScript.Echo Space(4) & ".ToString(): " & .ToString()
    End With
Next
End Sub



Dim DAClient: Set DAClient = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.Echo
WScript.Echo "OPC DATA ACCESS"
On Error Resume Next
Dim DAServerElements: Set DAServerElements = DAClient.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
DumpServerElements DAServerElements

Dim AEClient: Set AEClient =
```

```
CreateObject("OpcLabs.EasyOpc.AlarmsAndEvents.EasyAEClient")
WScript.Echo
WScript.Echo "OPC ALARMS AND EVENTS"
On Error Resume Next
Dim AEServerElements: Set AEServerElements = AEClient.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
DumpServerElements AEServerElements
```

VBScript

```
Rem This example shows all information available about OPC servers.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim ServerElements: Set ServerElements = Client.BrowseServers("")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

Dim ServerElement: For Each ServerElement In ServerElements
    WScript.Echo "Information about server """ & ServerElement & """:"
    With ServerElement
        WScript.Echo Space(4) & ".ServerClass: " & .ServerClass
        WScript.Echo Space(4) & ".ClsidString: " & .ClsidString
        WScript.Echo Space(4) & ".ProgId: " & .ProgId
        WScript.Echo Space(4) & ".Description: " & .Description
        WScript.Echo Space(4) & ".Vendor: " & .Vendor
        WScript.Echo Space(4) & ".ServerCategories.ToString(): " &
.ServerCategories.ToString()
        WScript.Echo Space(4) & ".VersionIndependentProgId: " &
.VersionIndependentProgId
    End With
Next
```

13.2.5.7 Examples - OPC Data Access - Browse nodes recursively

C#

```
// This example shows how to recursively browse the nodes in the OPC address space.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
```

```
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class BrowseNodes
    {
        public static void Recursive()
        {
            var stopwatch = new Stopwatch();
            stopwatch.Start();

            var client = new EasyDAClient();
            _branchCount = 0;
            _leafCount = 0;

            try
            {
                BrowseFromNode(client, "OPCLabs.KitServer.2", "");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            stopwatch.Stop();
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds);
            Console.WriteLine("Branch count: {0}", _branchCount);
            Console.WriteLine("Leaf count: {0}", _leafCount);
        }

        private static void BrowseFromNode(
            EasyDAClient client,
            ServerDescriptor serverDescriptor,
            DANodeDescriptor parentNodeDescriptor)
        {
            Debug.Assert(client != null);
            Debug.Assert(serverDescriptor != null);
            Debug.Assert(parentNodeDescriptor != null);

            // Obtain all node elements under parentNodeDescriptor
            var browseParameters = new DABrowseParameters();    // no filtering
whatsoever
            DANodeElementCollection nodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters);
            // Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            // it, here omitted for brevity.

            foreach (DANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);
```

```
                Console.WriteLine(nodeElement);

                // If the node is a branch, browse recursively into it.
                if (nodeElement.IsBranch)
                {
                    _branchCount++;
                    BrowseFromNode(client, serverDescriptor, nodeElement);
                }
                else
                {
                    _leafCount++;
                }
            }
        }

        private static int _branchCount;
        private static int _leafCount;
    }
}
```

VB.NET

```
' This example shows how to recursively browse the nodes in the OPC address space.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class BrowseNodes

        Shared Sub Recursive()
            Dim stopwatch = New Stopwatch()
            stopwatch.Start()

            Dim client = New EasyDAClient()
            _branchCount = 0
            _leafCount = 0

            Try
                BrowseFromNode(client, "OPCLabs.KitServer.2", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            stopwatch.Stop()
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds)
            Console.WriteLine("Branch count: {0}", _branchCount)
            Console.WriteLine("Leaf count: {0}", _leafCount)
        End Sub

        Private Shared Sub BrowseFromNode( _
```

```
            client As EasyDAClient,
            serverDescriptor As ServerDescriptor,
            parentNodeDescriptor As DANodeDescriptor)

            Debug.Assert(client IsNot Nothing)
            Debug.Assert(serverDescriptor IsNot Nothing)
            Debug.Assert(parentNodeDescriptor IsNot Nothing)

            ' Obtain all node elements under parentNodeDescriptor
            Dim browseParameters = New DABrowseParameters() ' no filtering whatsoever
            Dim nodeElementCollection As DANodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters)
            ' Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            ' it, here omitted for brevity.

            For Each nodeElement As DANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine(nodeElement)

                ' If the node is a branch, browse recursively into it.
                If nodeElement.IsBranch Then
                    _branchCount += 1
                    BrowseFromNode(client, serverDescriptor, nodeElement)
                Else
                    _leafCount += 1
                End If
            Next nodeElement
        End Sub

        Private Shared _branchCount As Integer
        Private Shared _leafCount As Integer
    End Class
End Namespace
```

13.2.5.8 Examples - OPC Data Access - Browse nodes recursively and read their values

C#

```
// Recursively browses and displays the nodes in the OPC address space, and attempts to
read and display values of all OPC
// items it finds.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
```

```
    partial class BrowseNodes
    {
        const string ServerClass = "OPCLabs.KitServer.2";

        static readonly EasyDAClient Client = new EasyDAClient();

        static void BrowseAndReadFromNode(string parentItemId)
        {
            // Obtain all node elements under parentItemId
            var browseParameters = new DABrowseParameters(); // no filtering whatsoever
            DANodeElementCollection nodeElementCollection = Client.BrowseNodes("",
ServerClass, parentItemId,
                browseParameters);
            // Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for it, here
            // omitted for brevity.

            foreach (DANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);

                // If the node is a leaf, it might be possible to read from it
                if (nodeElement.IsLeaf)
                {
                    // Determine what the display - either the value read, or exception
message in case of failure.
                    string display;
                    try
                    {
                        object value = Client.ReadItemValue("", ServerClass,
nodeElement);
                        display = String.Format("{0}", value);
                    }
                    catch (OpcException exception)
                    {
                        display = String.Format("** {0} **",
exception.GetBaseException().Message);
                    }

                    Console.WriteLine("{0} -> {1}", nodeElement.ItemId, display);
                }
                // If the node is not a leaf, just display its itemId
                else
                    Console.WriteLine("{0}", nodeElement.ItemId);

                // If the node is a branch, browse recursively into it.
                if (nodeElement.IsBranch &&
                    (nodeElement.ItemId != "SimulateEvents")
                    /* this branch is too big for the purpose of this example */)
                    BrowseAndReadFromNode(nodeElement);
            }
        }

        public static void RecursiveWithRead()
        {
            Console.WriteLine("Browsing and reading values...");
            // Set timeout to only wait 1 second - default would be 1 minute to wait
```

```
for good quality that may never come.
            Client.InstanceParameters.Timeouts.ReadItem = 1000;

            // Do the actual browsing and reading, starting from root of OPC address
space (denoted by empty string for itemId)
            try
            {
                BrowseAndReadFromNode("");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' Recursively browses and displays the nodes in the OPC address space, and attempts to
read and display values of all OPC
' items it finds.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class BrowseNodes
        Private Const ServerClass As String = "OPCLabs.KitServer.2"

        Private Shared ReadOnly Client As New EasyDAClient()

        Private Shared Sub BrowseAndReadFromNode(parentItemId As String)
            ' Obtain all node elements under parentItemId
            Dim browseParameters = New DABrowseParameters() ' no filtering whatsoever
            Dim nodeElementCollection As DANodeElementCollection =
Client.BrowseNodes("", ServerClass, parentItemId, browseParameters)
            ' Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for it, here
            ' omitted for brevity.

            For Each nodeElement As DANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)

                ' If the node is a leaf, it might be possible to read from it
                If nodeElement.IsLeaf Then
                    ' Determine what the display - either the value read, or exception
message in case of failure.
                    Dim display As String
                    Try
                        Dim value As Object = Client.ReadItemValue("", ServerClass,
nodeElement.ItemId)
                        display = String.Format("{0}", value)
                    Catch exception As OpcException
                        display = String.Format("** {0} **",
```

```
exception.GetBaseException().Message)
                    End Try

                    Console.WriteLine("{0} -> {1}", nodeElement.ItemId, display)
                    ' If the node is not a leaf, just display its itemId
                Else
                    Console.WriteLine("{0}", nodeElement.ItemId)
                End If

                ' If the node is a branch, browse recursively into it.
                If nodeElement.IsBranch AndAlso (nodeElement.ItemId <>
"SimulateEvents") Then ' this branch is too big for the purpose of this example
                    BrowseAndReadFromNode(nodeElement.ItemId)
                End If
            Next nodeElement
        End Sub

        Shared Sub RecursiveWithRead()
            Console.WriteLine("Browsing and reading values...")
            ' Set timeout to only wait 1 second - default would be 1 minute to wait for
good quality that may never come.
            Client.InstanceParameters.Timeouts.ReadItem = 1000

            ' Do the actual browsing and reading, starting from root of OPC address
space (denoted by empty string for itemId)
            Try
                BrowseAndReadFromNode("")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

13.2.5.9 Examples - OPC Data Access - Callback using a lambda

C#

```
// This example shows how subscribe to changes of a single item and display the value
of the item with each change,
// using a callback method specified using lambda expression.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.DataAccess._EasyDAClient
{
```

```
    partial class SubscribeItem
    {
        public static void CallbackLambda()
        {
            // Instantiate the client object
            var client = new EasyDAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the value
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000,
                (sender, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);

                    if (eventArgs.Succeeded)
                    {
                        Debug.Assert(eventArgs.Vtq != null);
                        Console.WriteLine(eventArgs.Vtq.ToString());
                    }
                    else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                });

            Console.WriteLine("Processing item changed events for 10 seconds...");
            Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Waiting for 2 seconds...");
            Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of a single item and display the value of
the item with each change,
' using a callback method specified using lambda expression.

Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeItem
        Shared Sub CallbackLambda()
            ' Instantiate the client object
            Dim client = New EasyDAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the value
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000,
                    Sub(sender, eventArgs)
                        Debug.Assert(eventArgs IsNot Nothing)
                        If eventArgs.Succeeded Then
                            Debug.Assert(eventArgs.Vtq IsNot Nothing)
```

```
                            Console.WriteLine(eventArgs.Vtq.ToString())
                        Else
                            Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                        End If
                    End Sub)

            Console.WriteLine("Processing item changed events for 10 seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

13.2.5.10 Examples - OPC Data Access - Change percent deadband of a subscription

C#

```
// This example shows how change the percent deadband of an existing subscription.

using System;
using System.Threading;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ChangeItemSubscription
    {
        public static void PercentDeadband()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged_PercentDeadband;

                Console.WriteLine("Subscribing with 10% deadband...");
                int handle = client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Ramp 0-100 (10 s)",
                    VarTypes.Empty, 100, 10.0f, null);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Changing subscription to 0% deadband...");
                client.ChangeItemSubscription(handle, new DAGroupParameters(100,
```

```
0.0f));

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllItems();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Item changed event handler
        static void client_ItemChanged_PercentDeadband(object sender,
EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how change the percent deadband of an existing subscription.

Imports System.Threading
Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess._EasyDAClient

    Partial Friend Class ChangeItemSubscription
        Public Shared Sub PercentDeadband()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf
client_ItemChanged_PercentDeadband

                Console.WriteLine("Subscribing with 10% deadband...")
                Dim handle As Integer = client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Ramp 0-100 (10 s)", VarTypes.Empty, 100, 10.0F, Nothing)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Changing subscription to 0% deadband...")
                client.ChangeItemSubscription(handle, New DAGroupParameters(100, 0.0F))

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeAllItems()
```

```
                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged_PercentDeadband(ByVal sender As Object,
ByVal e As EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

13.2.5.11 Examples - OPC Data Access - Change update rate of a single subscription

C#

```
// This example shows how change the update rate of an existing subscription.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ChangeItemSubscription
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000);

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Changing subscription...");
                client.ChangeItemSubscription(handle, new DAGroupParameters(100));

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
```

```
                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllItems();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Item changed event handler
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how change the update rate of an existing subscription.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient

    Partial Friend Class ChangeItemSubscription
        Public Shared Sub Main1()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000)

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Changing subscription...")
                client.ChangeItemSubscription(handle, New DAGroupParameters(100))

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
                client.UnsubscribeAllItems()

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged(ByVal sender As Object, ByVal e As
```

```
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how change the update rate of an existing subscription.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Changing subscription..."
Client.ChangeItemSubscription handle, 100

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.DisconnectObject Client
Set Client = Nothing



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq
End Sub
```

13.2.5.12 Examples - OPC Data Access - Change update rate of multiple subscriptions

VBScript

```
Rem This example shows how change the update rate of multiple existing subscriptions.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handleArray: handleArray = Client.SubscribeMultipleItems(arguments)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Changing subscriptions..."

Dim HandleGroupArguments1: Set HandleGroupArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAHandleGroupArguments")
HandleGroupArguments1.Handle = handleArray(0)
HandleGroupArguments1.GroupParameters.RequestedUpdateRate = 100
```

```
Dim HandleGroupArguments2: Set HandleGroupArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAHandleGroupArguments")
HandleGroupArguments2.Handle = handleArray(1)
HandleGroupArguments2.GroupParameters.RequestedUpdateRate = 100

Dim subscriptionChangeArguments(1)
Set subscriptionChangeArguments(0) = HandleGroupArguments1
Set subscriptionChangeArguments(1) = HandleGroupArguments2

Client.ChangeMultipleItemSubscriptions subscriptionChangeArguments

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.DisconnectObject Client
Set Client = Nothing



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq
End Sub
```

13.2.5.13 Examples - OPC Data Access - Event pull

C#

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    class PullItemChanged
    {
        public static void Main1()
        {
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
```

```
            var client = new EasyDAClient { PullItemChangedQueueCapacity = 1000 };

            Console.WriteLine("Subscribing item changes...");
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

            Console.WriteLine("Processing item changes for 1 minute...");
            int endTick = Environment.TickCount + 60 * 1000;
            do
            {
                EasyDAItemChangedEventArgs eventArgs = client.PullItemChanged(2 *
1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);

            Console.WriteLine("Unsubscribing item changes...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to item changes and obtain the events by pulling
them.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class PullItemChanged
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            client.PullItemChangedQueueCapacity = 1000

            Console.WriteLine("Subscribing item changes...")
            client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000)

            Console.WriteLine("Processing item changes for 1 minute...")
            Dim endTick As Integer = Environment.TickCount + 60 * 1000
            Do
                Dim eventArgs As EasyDAItemChangedEventArgs = client.PullItemChanged(2
* 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
                End If
            Loop While Environment.TickCount < endTick

            Console.WriteLine("Unsubscribing item changes...")
            client.UnsubscribeAllItems()
```

```
            Console.WriteLine("Finished.")
        End Sub
    End Class
End Namespace
```

JScript

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullItemChangedQueueCapacity = 1000;

WScript.Echo("Subscribing item changes...");
Client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

WScript.Echo("Processing item changes for 1 minute...");
var endTime = new Date().getTime() + 60*1000
do {
    var EventArgs = Client.PullItemChanged(2*1000);
    if (EventArgs !== null) {
        //  Handle the notification event
        WScript.Echo(EventArgs);
    }
} while(new Date().getTime() < endTime);

WScript.Echo("Unsubscribing item changes...");
Client.UnsubscribeAllItems();

WScript.Echo("Finished.");
```

Object Pascal

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

class procedure PullItemChanged.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  EndTick: Cardinal;
  EventArgs: _EasyDAItemChangedEventArgs;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullItemChangedQueueCapacity := 1000;

  WriteLn('Subscribing item changes...');
  Client.SubscribeItem('', 'OPCLabs.KitServer.2', 'Simulation.Random', 1000);

  WriteLn('Processing item changes for 1 minute...');
  EndTick := Ticks + 60*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullItemChanged(2*1000);
    if EventArgs <> nil then
```

```
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing item changes...');
  Client.UnsubscribeAllItems;

  WriteLn('Finished.');
end;
```

PHP

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
$Client->PullItemChangedQueueCapacity = 1000;

print "Subscribing item changes...\n";
$Client->SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

print "Processing item changed events for 1 minute...\n";
$endTime = time() + 60;
do {
    $EventArgs = $Client->PullItemChanged(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        print $EventArgs->ToString();
        print "\n";
    }
} while (time() < $endTime);
```

PowerScript

```
// This example shows how to subscribe to item changes and obtain the events by pulling
them.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
// In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullItemChangedQueueCapacity = 1000

mle_outputtext.Text = mle_outputtext.Text + "Subscribing item changes..." + "~r~n"
client.SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000)

mle_outputtext.Text = mle_outputtext.Text + "Processing item changes for 1 minute..." +
"~r~n"

Time endTime = RelativeTime(Now(), 60)
DO
    OLEObject eventArgs
    eventArgs = client.PullItemChanged(2*1000)
```

```
    IF NOT IsNull(eventArgs) THEN
        // Handle the notification event
        mle_outputtext.Text = mle_outputtext.Text + eventArgs.DisplayString + "~r~n"
    END IF
LOOP WHILE Now() < endTime

mle_outputtext.Text = mle_outputtext.Text + "Unsubscribing item changes..." + "~r~n"
client.UnsubscribeAllItems()

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to subscribe to item changes and obtain the events by pulling
them.

import time
import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.DataAccess.EasyDAClient')
# In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullItemChangedQueueCapacity = 1000

print('Subscribing item changes...')
client.SubscribeItem('', 'OPCLabs.KitServer.2', 'Simulation.Random', 1000)

print('Processing item changes for 1 minute...')
endTime = time.time() + 60
while time.time() < endTime:
    eventArgs = client.PullItemChanged(2*1000)
    if eventArgs is not None:
        # Handle the notification event
        print(eventArgs)

print('Unsubscribing item changes...')
client.UnsubscribeAllItems()

print('Finished.')
```

VBScript

```
Rem This example shows how to subscribe to item changes and obtain the events by
pulling them.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullItemChangedQueueCapacity = 1000

WScript.Echo "Subscribing item changes..."
Client.SubscribeItem "", "OPCLabs.KitServer.2", "Simulation.Random", 1000

WScript.Echo "Processing item changes for 1 minute..."
Dim endTime: endTime = Now() + 60*(1/24/60/60)
```

```
Do
    Dim EventArgs: Set EventArgs = Client.PullItemChanged(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime

WScript.Echo "Unsubscribing item changes..."
Client.UnsubscribeAllItems

WScript.Echo "Finished."
```

13.2.5.14 Examples - OPC Data Access - Get a record with property values

C#

```
// This example shows how to obtain a structure containing property values for an OPC
item, and display some property values.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Extensions;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientExtension
{
    class GetItemPropertyRecord
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            // Get a structure containing values of all well-known properties
            DAItemPropertyRecord itemPropertyRecord;
            try
            {
                itemPropertyRecord = client.GetItemPropertyRecord("",
"OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            // Display some of the obtained property values
            Console.WriteLine("itemPropertyRecord.AccessRights: {0}",
itemPropertyRecord.AccessRights);
            Console.WriteLine("itemPropertyRecord.DataType: {0}",
itemPropertyRecord.DataType);
            Console.WriteLine("itemPropertyRecord.Timestamp: {0}",
```

```
itemPropertyRecord.Timestamp);
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a structure containing property values for an OPC
item, and display some property values.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Extensions
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientExtension
    Friend Class GetItemPropertyRecord
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            ' Get a structure containing values of all well-known properties
            Dim itemPropertyRecord As DAItemPropertyRecord
            Try
                itemPropertyRecord = client.GetItemPropertyRecord("",
"OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Display some of the obtained property values
            Console.WriteLine("itemPropertyRecord.AccessRights: {0}",
itemPropertyRecord.AccessRights)
            Console.WriteLine("itemPropertyRecord.DataType: {0}",
itemPropertyRecord.DataType)
            Console.WriteLine("itemPropertyRecord.Timestamp: {0}",
itemPropertyRecord.Timestamp)
        End Sub
    End Class
End Namespace
```

13.2.5.15 Examples - OPC Data Access - Get a value of single property

C#

```
// This example shows how to get a value of a single OPC property.
//
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
```

```
// IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to obtain
better property values. Your code may
// also subscribe to the item in order to assure that it remains active.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class GetPropertyValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            object value;
            try
            {
                value = client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random",
                    DAPropertyIds.Timestamp);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine(value);
        }
    }
}
```

PHP

```
// This example shows how to get a value of a single OPC property.
//
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to obtain
better property values. Your code may
// also subscribe to the item in order to assure that it remains active.

const Timestamp = 4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $value = $Client->GetPropertyValue("", "OPCLabs.KitServer.2", "Simulation.Random",
Timestamp);
}
catch (com_exception $e)
```

```
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

printf("%s\n", $value);
```

VB.NET

```
' This example shows how to get a value of a single OPC property.
'
' Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
' case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
' IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to obtain
better property values. Your code may
' also subscribe to the item in order to assure that it remains active.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class GetPropertyValue
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim value As Object
            Try
                value = client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random", DAPropertyIds.Timestamp)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine(value)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to get a value of a single OPC property.
Rem
Rem Note that some properties may not have a useful value initially (e.g. until the
item is activated in a group), which also the
Rem case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
Rem IEasyDAClient.ReadItemValue.Main.vbs shortly before this example, in order to
obtain better property values. Your code may
Rem also subscribe to the item in order to assure that it remains active.

Option Explicit

Const Timestamp = 4
```

```
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

On Error Resume Next
Dim value: value = Client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random", Timestamp)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo value
```

VBScript

```
Rem This example measures the time needed to get values of all OPC properties of a
single OPC item "one by one".

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.ReadValue_I4"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim PropertyElementCollection
On Error Resume Next
Set PropertyElementCollection = Client.BrowseProperties(ServerDescriptor,
NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

'EasyDAClient.ReadItemValue "", "OPCLabs.KitServer.2", "Simulation.ReadValue_I4"
Dim startTime: startTime = Timer
Dim PropertyElement: For Each PropertyElement In PropertyElementCollection
    Dim propertyID: Set propertyID = PropertyElement.PropertyID
    On Error Resume Next
    Dim value: value = Client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.ReadValue_I4", propertyID.NumericalValue)
    If Err.Number <> 0 Then
        WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
        WScript.Quit
    End If
    On Error Goto 0
    'WScript.Echo value
Next
WScript.Echo "Time taken (milliseconds): " & (Timer - startTime)*1000
```

The GetPropertyValue Method returns a generic object, regardless of the property involved. You may therefore need to cast the returned value to the expected type, and possibly do further processing on the value, as in the the example below.

C#

```
// This example shows how to obtain a data type of an OPC item.

using System;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class GetPropertyValue
    {
        public static void DataType()
        {
            var client = new EasyDAClient();

            // Get the value of DataType property; it is a 16-bit signed integer
            short dataType;
            try
            {
                dataType = (short)client.GetPropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random",
                    DAPropertyIds.DataType);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
            // Convert the data type to VarType
            var varType = (VarType)dataType;

            // Display the obtained data type
            Console.WriteLine("DataType: {0}", dataType);   // Display data type as
numerical value
            Console.WriteLine("VarType: {0}", varType);     // Display data type
symbolically

            // Code below illustrates how decisions can be made based on type
            switch (varType.InternalValue)
            {
                case VarTypes.R8:
                    Console.WriteLine("The data type is VarTypes.R8, as we expected.");
                    break;

                // other cases may come here ...

                default:
                    Console.WriteLine("The data type is not as we expected!");
                    break;
```

```
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of an OPC item.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class GetPropertyValue
        Public Shared Sub DataType()
            Dim client = New EasyDAClient()

            ' Get the value of DataType property; it is a 16-bit signed integer
            Dim aDataType As Short
            Try
                aDataType = CShort(Fix(client.GetPropertyValue("",
"OPCLabs.KitServer.2", "Simulation.Random", DAPropertyIds.DataType)))
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Convert the data type to VarType
            Dim varType = CType(aDataType, VarType)

            ' Display the obtained data type
            Console.WriteLine("DataType: {0}", aDataType) ' Display data type as
numerical value
            Console.WriteLine("VarType: {0}", varType) ' Display data type symbolically

            ' Code below illustrates how decisions can be made based on type
            Select Case varType
                Case VarTypes.R8
                    Console.WriteLine("The data type is VarTypes.R8, as we expected.")

                    ' other cases may come here ...

                Case Else
                    Console.WriteLine("The data type is not as we expected!")
            End Select
        End Sub
    End Class
End Namespace
```

13.2.5.16 Examples - OPC Data Access - Get data type of an item

C#

```
// This example shows how to obtain a data type of an OPC item.

using System;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Extensions;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientExtension
{
    class GetDataTypePropertyValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            // Get the DataType property value, already converted to VarType
            VarType varType;
            try
            {
                varType = client.GetDataTypePropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            // Display the obtained data type
            Console.WriteLine("VarType: {0}", varType); // Display data type
symbolically
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of an OPC item.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Extensions
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientExtension
    Friend Class GetDataTypePropertyValue
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()
```

```
            ' Get the DataType property value, already converted to VarType
            Dim varType As VarType
            Try
                varType = client.GetDataTypePropertyValue("", "OPCLabs.KitServer.2",
"Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Display the obtained data type
            Console.WriteLine("VarType: {0}", varType) ' Display data type symbolically
        End Sub
    End Class
End Namespace
```

13.2.5.17 Examples - OPC Data Access - Get dictionary of property values

C#

```
// This example shows how to obtain a dictionary of OPC property values for an OPC
item, and extract property values.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.Extensions;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientExtension
{
    class GetPropertyValueDictionary
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            // Get dictionary of property values, for all well-known properties
            DAPropertyValueDictionary propertyValueDictionary;
            try
            {
                propertyValueDictionary = client.GetPropertyValueDictionary("",
"OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
```

```
            // Display some of the obtained property values
            // The production code should also check for the .Exception first, before
getting .Value

Console.WriteLine("propertyValueDictionary[DAPropertyId.AccessRights].Value: {0}",
                propertyValueDictionary[DAPropertyIds.AccessRights].Value);
            Console.WriteLine("propertyValueDictionary[DAPropertyId.DataType].Value:
{0}",
                propertyValueDictionary[DAPropertyIds.DataType].Value);
            Console.WriteLine("propertyValueDictionary[DAPropertyId.Timestamp].Value:
{0}",
                propertyValueDictionary[DAPropertyIds.Timestamp].Value);
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a dictionary of OPC property values for an OPC item,
and extract property values.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.Extensions
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientExtension
    Friend Class GetPropertyValueDictionary
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            ' Get dictionary of property values, for all well-known properties
            Dim propertyValueDictionary As DAPropertyValueDictionary
            Try
                propertyValueDictionary = client.GetPropertyValueDictionary("",
"OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            ' Display some of the obtained property values
            ' The production code should also check for the .Exception first, before
getting .Value

Console.WriteLine("propertyValueDictionary[DAPropertyId.AccessRights].Value: {0}",
propertyValueDictionary(DAPropertyIds.AccessRights).Value)
            Console.WriteLine("propertyValueDictionary[DAPropertyId.DataType].Value:
{0}", propertyValueDictionary(DAPropertyIds.DataType).Value)
            Console.WriteLine("propertyValueDictionary[DAPropertyId.Timestamp].Value:
{0}", propertyValueDictionary(DAPropertyIds.Timestamp).Value)
        End Sub
    End Class
End Namespace
```

13.2.5.18 Examples - OPC Data Access - Get values of multiple properties

Object Pascal

```
// This example shows how to get value of multiple OPC properties.
//
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadMultipleItemValues.Main.vbs shortly before this example, in order
to obtain better property values. Your
// code may also subscribe to the items in order to assure that they remain active.

class procedure GetMultiplePropertyValues.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  PropertyArguments1: _DAPropertyArguments;
  PropertyArguments2: _DAPropertyArguments;
  PropertyArguments3: _DAPropertyArguments;
  PropertyArguments4: _DAPropertyArguments;
  ValueResult: _ValueResult;
  Results: OleVariant;
begin
  PropertyArguments1 := CoDAPropertyArguments.Create;
  PropertyArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments1.NodeDescriptor.ItemID := 'Simulation.Random';
  PropertyArguments1.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_Timestamp;

  PropertyArguments2 := CoDAPropertyArguments.Create;
  PropertyArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments2.NodeDescriptor.ItemID := 'Simulation.Random';
  PropertyArguments2.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_AccessRights;

  PropertyArguments3 := CoDAPropertyArguments.Create;
  PropertyArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments3.NodeDescriptor.ItemID := 'Trends.Ramp (1 min)';
  PropertyArguments3.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_Timestamp;

  PropertyArguments4 := CoDAPropertyArguments.Create;
  PropertyArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  PropertyArguments4.NodeDescriptor.ItemID := 'Trends.Ramp (1 min)';
  PropertyArguments4.PropertyDescriptor.PropertyId.NumericalValue :=
DAPropertyIds_AccessRights;

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := PropertyArguments1;
  Arguments[1] := PropertyArguments2;
```

```
  Arguments[2] := PropertyArguments3;
  Arguments[3] := PropertyArguments4;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.GetMultiplePropertyValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
      ValueResult := IInterface(Results[I]) as _ValueResult;

      // Check if there has been an error getting the property value
      if ValueResult.Exception <> nil then
      begin
        WriteLn(Format('%s *** Failure: %s', [Arguments[I].NodeDescriptor.NodeId,
ValueResult.Exception.Message]));
        Continue;
      end;

      WriteLn('results(', i, ').Value: ', ValueResult.Value);
  end;
end;
```

PHP

```
// This example shows how to get value of multiple OPC properties, and handle errors.
//
// Note that some properties may not have a useful value initially (e.g. until the item
is activated in a group), which also the
// case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
// IEasyDAClient.ReadMultipleItemValues.Main.vbs shortly before this example, in order
to obtain better property values. Your
// code may also subscribe to the items in order to assure that they remain active.

const Timestamp = 4;
const AccessRights = 5;

$PropertyArguments1 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments1->NodeDescriptor->ItemID = "Simulation.Random";
$PropertyArguments1->PropertyDescriptor->PropertyID->NumericalValue = Timestamp;

$PropertyArguments2 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments2->NodeDescriptor->ItemID = "Simulation.Random";
$PropertyArguments2->PropertyDescriptor->PropertyID->NumericalValue = AccessRights;

$PropertyArguments3 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
```

```
$PropertyArguments3->NodeDescriptor->ItemID = "Trends.Ramp (1 min)";
$PropertyArguments3->PropertyDescriptor->PropertyID->NumericalValue = Timestamp;

$PropertyArguments4 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments");
$PropertyArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$PropertyArguments4->NodeDescriptor->ItemID = "Trends.Ramp (1 min)";
$PropertyArguments4->PropertyDescriptor->PropertyID->NumericalValue = AccessRights;

$arguments[0] = $PropertyArguments1;
$arguments[1] = $PropertyArguments2;
$arguments[2] = $PropertyArguments3;
$arguments[3] = $PropertyArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

$results = $Client->GetMultiplePropertyValues($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $attributeDataResult = $results[$i];
    if ($results[$i]->Succeeded)
        printf("results[%d].Value: %s\n", $i, $results[$i]->Value);
    else
        printf("results[%d]: *** Failure: %s\n", $i, $results[$i]->ErrorMessageBrief);
}
```

VBScript

```
Rem This example shows how to get value of multiple OPC properties, and handle errors.
Rem
Rem Note that some properties may not have a useful value initially (e.g. until the
item is activated in a group), which also the
Rem case with Timestamp property as implemented by the demo server. This behavior is
server-dependent, and normal. You can run
Rem IEasyDAClient.ReadMultipleItemValues.Main.vbs shortly before this example, in order
to obtain better property values. Your
Rem code may also subscribe to the items in order to assure that they remain active.

Option Explicit

Const Timestamp = 4
Const AccessRights = 5

Dim PropertyArguments1: Set PropertyArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments1.NodeDescriptor.ItemID = "Simulation.Random"
PropertyArguments1.PropertyDescriptor.PropertyID.NumericalValue = Timestamp

Dim PropertyArguments2: Set PropertyArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments2.NodeDescriptor.ItemID = "Simulation.Random"
PropertyArguments2.PropertyDescriptor.PropertyID.NumericalValue = AccessRights

Dim PropertyArguments3: Set PropertyArguments3 =
```

```
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments3.NodeDescriptor.ItemID = "Trends.Ramp (1 min)"
PropertyArguments3.PropertyDescriptor.PropertyID.NumericalValue = Timestamp

Dim PropertyArguments4: Set PropertyArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
PropertyArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
PropertyArguments4.NodeDescriptor.ItemID = "Trends.Ramp (1 min)"
PropertyArguments4.PropertyDescriptor.PropertyID.NumericalValue = AccessRights

Dim arguments(3)
Set arguments(0) = PropertyArguments1
Set arguments(1) = PropertyArguments2
Set arguments(2) = PropertyArguments3
Set arguments(3) = PropertyArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.GetMultiplePropertyValues(arguments)

Dim i: For i = LBound(results) To UBound(results)
    If results(i).Exception Is Nothing Then
        WScript.Echo "results(" & i & ").Value: " & results(i).Value
    Else
        WScript.Echo "results(" & i & ").Exception.Message: " &
results(i).Exception.Message
    End If
Next
```

VBScript

```
Rem This example measures the time needed to get values of all OPC properties of a
single OPC item all at once.
Rem This example shows how to get value of multiple OPC properties.

Option Explicit

Dim ServerDescriptor: Set ServerDescriptor =
CreateObject("OpcLabs.EasyOpc.ServerDescriptor")
ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.DataAccess.DANodeDescriptor")
NodeDescriptor.ItemID = "Simulation.ReadValue_I4"

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim PropertyElementCollection
On Error Resume Next
Set PropertyElementCollection = Client.BrowseProperties(ServerDescriptor,
NodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

```
Dim count: count = PropertyElementCollection.Count

Dim arguments(): Redim arguments(count - 1)
Dim i: i = 0
Dim PropertyElement: For Each PropertyElement In PropertyElementCollection
    Dim PropertyArguments: Set PropertyArguments =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAPropertyArguments")
    PropertyArguments.ServerDescriptor = ServerDescriptor
    PropertyArguments.NodeDescriptor = NodeDescriptor
    PropertyArguments.PropertyDescriptor.PropertyID = PropertyElement.PropertyId

    Set arguments(i) = PropertyArguments
    i = i + 1
Next

'EasyDAClient.ReadItemValue "", "OPCLabs.KitServer.2", "Simulation.ReadValue_I4"
Dim startTime: startTime = Timer
Dim results: results = Client.GetMultiplePropertyValues(arguments)
WScript.Echo "Time taken (milliseconds): " & (Timer - startTime)*1000

'For i = LBound(results) To UBound(results)
'    If results(i).Exception Is Nothing Then
'        WScript.Echo "results(" & i & ").Value: " & results(i).Value
'    Else
'        WScript.Echo "results(" & i & ").Exception.Message: " &
results(i).Exception.Message
'    End If
''Next
```

The example below is a bit more complex, and combines the ability to browse for items with getting the data type property for each of the items obtained.

C#

```
// This example shows how to obtain a data type of all OPC items under a branch.

using System;
using System.Linq;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;

namespace DocExamples.DataAccess._EasyDAClient
{
    class GetMultiplePropertyValues
    {
        public static void DataType()
        {
            var client = new EasyDAClient();
            ServerDescriptor serverDescriptor = "OPCLabs.KitServer.2";

            // Browse for all leaves under the "Simulation" branch
            DANodeElementCollection nodeElementCollection =
```

```
client.BrowseLeaves(serverDescriptor, "Simulation");

            // Create list of node descriptors, one for each leaf obtained
            DANodeDescriptor[] nodeDescriptorArray = nodeElementCollection
                .Where(element => !element.IsHint)  // filter out hint leafs that do
not represent real OPC items (rare)
                .Select(element => new DANodeDescriptor(element))
                .ToArray();

            // Get the value of DataType property; it is a 16-bit signed integer
            ValueResult[] valueResultArray =
client.GetMultiplePropertyValues(serverDescriptor,
                nodeDescriptorArray, DAPropertyIds.DataType);

            for (int i = 0; i < valueResultArray.Length; i++)
            {
                DANodeDescriptor nodeDescriptor = nodeDescriptorArray[i];

                // Check if there has been an error getting the property value
                ValueResult valueResult = valueResultArray[i];
                if (valueResult.Exception != null)
                {
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message);
                    continue;
                }

                // Convert the data type to VarType
                var varType = (VarType)(short)valueResult.Value;

                // Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of all OPC items under a branch.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess._EasyDAClient
    Friend Class GetMultiplePropertyValues
        Public Shared Sub DataType()
            Dim client = New EasyDAClient()
            Dim serverDescriptor As ServerDescriptor = "OPCLabs.KitServer.2"

            ' Browse for all leaves under the "Simulation" branch
            Dim nodeElementCollection = client.BrowseLeaves(serverDescriptor,
"Simulation")

            ' Create list of node descriptors, one for each leaf obtained
```

```
            ' filter out hint leafs that do not represent real OPC items (rare)
            Dim nodeDescriptorArray() As DANodeDescriptor = nodeElementCollection _
                .Where(Function(element) Not element.IsHint) _
                .Select(Function(element) New DANodeDescriptor(element)) _
                .ToArray()

            ' Get the value of DataType property; it is a 16-bit signed integer
            Dim valueResultArray() As ValueResult =
client.GetMultiplePropertyValues(serverDescriptor,
                    nodeDescriptorArray, DAPropertyIds.DataType)

            For i = 0 To valueResultArray.Length - 1
                Dim nodeDescriptor = nodeDescriptorArray(i)

                ' Check if there has been an error getting the property value
                Dim valueResult As ValueResult = valueResultArray(i)
                If valueResult.Exception IsNot Nothing Then
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message)
                    Continue For
                End If

                ' Convert the data type to VarType
                Dim varType = CType(CShort(valueResult.Value), VarType)

                ' Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType)
            Next i
        End Sub
    End Class
End Namespace
```

13.2.5.19 Examples - OPC Data Access - Obtain data types by browsing and filtering

C#

```
// This example shows how to obtain data types of leaves in the OPC-DA address
// space by browsing and filtering, i.e. without the use of OPC properties.
// This technique allows determining the data types with servers that only
// support OPC-DA 1.0. It can also be more effective than the use of
// GetMultiplePropertyValues, if there is large number of leaves, and
// relatively small number of data types to be checked.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class BrowseNodes
```

```
    {
        public static void DataTypes()
        {
            var client = new EasyDAClient();

            // Define the list of data types we will be checking for.
            // Change as needed for your application.
            // This technique is only usable if there is a known list of
            // data types you are interested in. If you are interested in
            // all leaves, even those that are of data types not explicitly
            // listed, always include VarTypes.Empty as the first data type.
            // The leaves of "unlisted" data types will have VarTypes.Empty
            // associated with them.
            var dataTypes = new VarType[] { VarTypes.Empty, VarTypes.I2, VarTypes.R4 };

            // For each leaf found, this dictionary wil hold its associated data type.
            var dataTypeDictionary = new Dictionary<DANodeElement, VarType>();

            // For each data type, browse for leaves of this data type.
            foreach (VarType dataType in dataTypes)
            {
                var browseParameters = new DABrowseParameters(DABrowseFilter.Leaves,
"", "", dataType);
                DANodeElementCollection nodeElements;
                try
                {
                    nodeElements = client.BrowseNodes("", "OPCLabs.KitServer.2",
"Greenhouse", browseParameters);
                }
                catch (OpcException opcException)
                {
                    Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                    return;
                }

                // Store the leaf information into the dictionary, and
                // associate the current data type with it.
                foreach (var nodeElement in nodeElements)
                    dataTypeDictionary[nodeElement] = dataType;
            }

            // Display each leaf found, and its associated data type.
            foreach (KeyValuePair<DANodeElement, VarType> pair in dataTypeDictionary)
            {
                DANodeElement nodeElement = pair.Key;
                VarType dataType = pair.Value;
                Console.WriteLine("{0}: {1}", nodeElement, dataType);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain data types of leaves in the OPC-DA address
' space by browsing and filtering, i.e. without the use of OPC properties.
```

```
' This technique allows determining the data types with servers that only
' support OPC-DA 1.0. It can also be more effective than the use of
' GetMultiplePropertyValues, if there is large number of leaves, and
' relatively small number of data types to be checked.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class BrowseNodes
        Shared Sub DataTypes()
            Dim client = New EasyDAClient()

            ' Define the list of data types we will be checking for.
            ' Change as needed for your application.
            ' This technique is only usable if there is a known list of
            ' data types you are interested in. If you are interested in
            ' all leaves, even those that are of data types not explicitly
            ' listed, always include VarTypes.Empty as the first data type.
            ' The leaves of "unlisted" data types will have VarTypes.Empty
            ' associated with them.
            Dim dataTypes() = New VarType() {VarTypes.Empty, VarTypes.I2, VarTypes.R4}

            ' For each leaf found, this dictionary wil hold its associated data type.
            Dim dataTypeDictionary = New Dictionary(Of DANodeElement, VarType)()

            ' For each data type, browse for leaves of this data type.
            For Each dataType As VarType In dataTypes
                Dim browseParameters = New DABrowseParameters(DABrowseFilter.Leaves,
"", "", dataType)
                Dim nodeElements As DANodeElementCollection
                Try
                    nodeElements = client.BrowseNodes("", "OPCLabs.KitServer.2",
"Greenhouse", browseParameters)
                Catch opcException As OpcException
                    Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                    Exit Sub
                End Try

                ' Store the leaf information into the dictionary, and
                ' associate the current data type with it.
                For Each nodeElement In nodeElements
                    dataTypeDictionary(nodeElement) = dataType
                Next nodeElement
            Next dataType

            For Each pair In dataTypeDictionary
                Dim nodeElement As DANodeElement = pair.Key
                Dim dataType As VarType = pair.Value
                Console.WriteLine("{0}: {1}", nodeElement, dataType)
            Next pair
        End Sub
    End Class
End Namespace
```

13.2.5.20 Examples - OPC Data Access - Read a single item

C#

```
// This example shows how to read a single item, and display its value, timestamp and
quality.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadItem
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            DAVtq vtq;
            try
            {
                vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Vtq: {0}", vtq);
        }
    }
}
```

PHP

```
// This example shows how to read a single item, and display its value, timestamp and
quality.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $Vtq = $Client->ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}
```

```
printf("Vtq: %s\n", $Vtq);
```

VB.NET

```
' This example shows how to read a single item, and display its value, timestamp and
quality.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItem
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim vtq As DAVtq
            Try
                vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Vtq: {0}", vtq)
        End Sub
    End Class
End Namespace
```

JScript

```
// This example shows how to read a single item, and display its value, timestamp and
quality.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
var VTQ = Client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
WScript.Echo("VTQ.ToString(): " + VTQ.ToString());
```

Object Pascal

```
// This example shows how to read a single item, and display its value, timestamp and
quality.

class procedure ReadItem.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Vtq: _DAVtq;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    Vtq := Client.ReadItem('', 'OPCLabs.KitServer.2', 'Simulation.Random');
  except
    on E: EOleException do
    begin
```

```
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('Vtq: ', Vtq.ToString);
end;
```

PowerScript

```
// This example shows how to read a single item, and display its value, timestamp and
quality.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Obtain value/timestamp/quality
OLEObject vtq
TRY
    vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Demo.Ramp")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + "Vtq: " + vtq.DisplayString + "~r~n"
```

VBScript

```
Rem This example shows how to read a single item, and display its value, timestamp and
quality.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim Vtq: Set Vtq = Client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "Vtq: " & Vtq
```

13.2.5.21 Examples - OPC Data Access - Read a single item using browse path

C#

```
// This example shows how to read a single item using a browse path, and display its
value, timestamp and quality.

using System;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadItem
    {
        public static void BrowsePath()
        {
            var client = new EasyDAClient();

            DAVtq vtq;
            try
            {
                vtq = client.ReadItem(
                    new ServerDescriptor("", "OPCLabs.KitServer.2"),
                    new DAItemDescriptor(null, "/Simulation/Random"));
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Vtq: {0}", vtq);
        }
    }
}
```

VB.NET

```
' This example shows how to read a single item using a browse path, and display its
value, timestamp and quality.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItem
        Public Shared Sub BrowsePath()
            Dim client = New EasyDAClient()

            Dim vtq As DAVtq
            Try
                vtq = client.ReadItem(New ServerDescriptor("", "OPCLabs.KitServer.2"),
New DAItemDescriptor(Nothing, "/Simulation/Random"))
            Catch opcException As OpcException
```

```
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Vtq: {0}", vtq)
        End Sub
    End Class
End Namespace
```

13.2.5.22 Examples - OPC Data Access - Read a single value

C#

```
// This example shows how to read and display value of a single item.
// One of the shortest examples possible.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    class ReadItemValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Reading item value...");
            object value;
            try
            {
                value = client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Ramp");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine(value);
        }
    }
}
```

PHP

```
// This example shows how to read value of a single item, and display it.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
```

```
try
{
    $value = $Client->ReadItemValue("", "OPCLabs.KitServer.2", "Simulation.Random");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

printf("value: %s\n", $value);
```

VB.NET

```
' This example shows how to read and display value of a single item.
' One of the shortest examples possible.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItemValue
        Public Shared Sub Main1()
            Dim client As New EasyDAClient()

            Console.WriteLine("Reading item...")
            Try
                Console.WriteLine(client.ReadItemValue("", "OPCLabs.KitServer.2",
"Demo.Ramp"))
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

JScript

```
// This example shows how to read values of a single item, and display it.

var Client = new ActiveXObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
var value = Client.ReadItemValue("", "OPCLabs.KitServer.2", "Simulation.Random");
WScript.Echo("value: " + value);
```

Object Pascal

```
// This example shows how to read value of a single item, and display it.

class procedure ReadItemValue.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  Value: OleVariant;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;
```

```
  try
    Value := Client.ReadItemValue('', 'OPCLabs.KitServer.2', 'Simulation.Random');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('Value: ', Value);
end;
```

PowerScript

```
// This example shows how to read and display value of a single item.
// One of the shortest examples possible.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Obtain value of a node
mle_outputtext.Text = mle_outputtext.Text + "Reading item value..." + "~r~n"
Any value
TRY
    value = client.ReadItemValue("", "OPCLabs.KitServer.2", "Demo.Ramp")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + String(value) + "~r~n"
```

Python

```
# This example shows how to read value of a single item, and display it.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.DataAccess.EasyDAClient')

# Perform the operation
value = client.ReadItemValue('', 'OPCLabs.KitServer.2', 'Demo.Single')

# Display results
print('value: ', value)
```

VBScript

```
Rem This example shows how to read value of a single item, and display it.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim value: value = Client.ReadItemValue("", "OPCLabs.KitServer.2", "Simulation.Random")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "value: " & value
```

VBScript

```
Rem This example shows how to read value of a single item, and display it, using CLSID
instead of ProgID of the OPC Server.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Dim value: value = Client.ReadItemValue("", "{C8A12F17-1E03-401E-B53D-6C654DD576DA}",
"Simulation.Random")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "value: " & value
```

13.2.5.23 Examples - OPC Data Access - Read an item and get a type code

C#

```
// This example shows how to read a single item and obtains a type code of the received
value.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadItem
    {
        public static void GetTypeCode()
```

```
        {
            var client = new EasyDAClient();

            DAVtq vtq;
            try
            {
                vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            if (vtq.Value != null)
            {
                TypeCode typeCode = Type.GetTypeCode(vtq.Value.GetType());

                Console.WriteLine("TypeCode: {0}", typeCode);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to read a single item and obtains a type code of the received
value.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItem
        Public Shared Sub GetTypeCode()
            Dim client = New EasyDAClient()

            Dim vtq As DAVtq
            Try
                vtq = client.ReadItem("", "OPCLabs.KitServer.2", "Simulation.Random")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            If vtq.Value IsNot Nothing Then
                Dim typeCode As TypeCode = Type.GetTypeCode(vtq.Value.GetType())

                Console.WriteLine("TypeCode: {0}", typeCode)
            End If
        End Sub
    End Class
End Namespace
```

13.2.5.24 Examples - OPC Data Access - Read items from the device

VBScript

```
Rem This example shows how to read 4 items from the device, and display their values,
timestamps and qualities.

Option Explicit

' Selects the data source for OPC reads (from device, from OPC cache, or dynamically
determined).
' The data source (memory, OPC cache or OPC device) selection will be based on the
desired value age and current status of
' data received from the server.
Const DADataSource_ByValueAge = 0
' OPC reads will be fulfilled from the cache in the OPC server.
Const DADataSource_Cache = 1
' OPC reads will be fulfilled from the device by the OPC server.
Const DADataSource_Device = 2

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ReadItemArguments1.ReadParameters.DataSource = DADataSource_Device  ' read will be from
device

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ReadItemArguments2.ReadParameters.DataSource = DADataSource_Device  ' read will be from
device

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ReadItemArguments3.ReadParameters.DataSource = DADataSource_Device  ' read will be from
device

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ReadItemArguments4.ReadParameters.DataSource = DADataSource_Device  ' read will be from
device

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4
```

```
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

13.2.5.25 Examples - OPC Data Access - Read items of various data types

C#

```
// Shows how different data types can be read, including rare types and arrays of
values.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadItem
    {
        static readonly EasyDAClient Client = new EasyDAClient();

        static void ReadAndDisplay(string itemId)
        {
            Console.WriteLine();
            Console.WriteLine("Reading \"{0}\"...", itemId);

            DAVtq vtq;
            try
            {
                vtq = Client.ReadItem("OPCLabs.KitServer.2", itemId);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }
            Console.WriteLine("Vtq: {0}", vtq);
        }

        public static void DataTypes()
        {
            ReadAndDisplay("Simulation.Register_EMPTY");
```

```
            ReadAndDisplay("Simulation.Register_NULL");
            ReadAndDisplay("Simulation.Register_DISPATCH");

            ReadAndDisplay("Simulation.ReadValue_I2");
            ReadAndDisplay("Simulation.ReadValue_I4");
            ReadAndDisplay("Simulation.ReadValue_R4");
            ReadAndDisplay("Simulation.ReadValue_R8");
            ReadAndDisplay("Simulation.ReadValue_CY");
            ReadAndDisplay("Simulation.ReadValue_DATE");
            ReadAndDisplay("Simulation.ReadValue_BSTR");
            ReadAndDisplay("Simulation.ReadValue_BOOL");
            ReadAndDisplay("Simulation.ReadValue_DECIMAL");
            ReadAndDisplay("Simulation.ReadValue_I1");
            ReadAndDisplay("Simulation.ReadValue_UI1");
            ReadAndDisplay("Simulation.ReadValue_UI2");
            ReadAndDisplay("Simulation.ReadValue_UI4");
            ReadAndDisplay("Simulation.ReadValue_INT");
            ReadAndDisplay("Simulation.ReadValue_UINT");

            ReadAndDisplay("Simulation.ReadValue_ArrayOfI2");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfI4");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfR4");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfR8");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfCY");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfDATE");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfBSTR");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfBOOL");
            //ReadAndDisplay("Simulation.ReadValue_ArrayOfDECIMAL");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfI1");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUI1");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUI2");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUI4");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfINT");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUINT");
        }
    }
}
```

VB.NET

```
' Shows how different data types can be read, including rare types and arrays of
values.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadItem
        Private Shared ReadOnly DAClient As New EasyDAClient()

        Private Shared Sub ReadAndDisplay(itemId As String)
            Console.WriteLine()
            Console.WriteLine("Reading ""{0}""...", itemId)

            Dim vtq As DAVtq
            Try
                vtq = DAClient.ReadItem("OPCLabs.KitServer.2", itemId)
```

```
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            Console.WriteLine("Vtq.ToString(): {0}", vtq.ToString())
        End Sub

        Shared Sub DataTypes()
            ReadAndDisplay("Simulation.Register_EMPTY")
            ReadAndDisplay("Simulation.Register_NULL")
            ReadAndDisplay("Simulation.Register_DISPATCH")

            ReadAndDisplay("Simulation.ReadValue_I2")
            ReadAndDisplay("Simulation.ReadValue_I4")
            ReadAndDisplay("Simulation.ReadValue_R4")
            ReadAndDisplay("Simulation.ReadValue_R8")
            ReadAndDisplay("Simulation.ReadValue_CY")
            ReadAndDisplay("Simulation.ReadValue_DATE")
            ReadAndDisplay("Simulation.ReadValue_BSTR")
            ReadAndDisplay("Simulation.ReadValue_BOOL")
            ReadAndDisplay("Simulation.ReadValue_DECIMAL")
            ReadAndDisplay("Simulation.ReadValue_I1")
            ReadAndDisplay("Simulation.ReadValue_UI1")
            ReadAndDisplay("Simulation.ReadValue_UI2")
            ReadAndDisplay("Simulation.ReadValue_UI4")
            ReadAndDisplay("Simulation.ReadValue_INT")
            ReadAndDisplay("Simulation.ReadValue_UINT")

            ReadAndDisplay("Simulation.ReadValue_ArrayOfI2")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfI4")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfR4")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfR8")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfCY")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfDATE")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfBSTR")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfBOOL")
            'ReadAndDisplay("Simulation.ReadValue_ArrayOfDECIMAL");
            ReadAndDisplay("Simulation.ReadValue_ArrayOfI1")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUI1")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUI2")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUI4")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfINT")
            ReadAndDisplay("Simulation.ReadValue_ArrayOfUINT")
        End Sub
    End Class
End Namespace
```

13.2.5.26 Examples - OPC Data Access - Read multiple item values

C#

```
// This example shows how to read 4 items at once, and display their values, timestamps and
```

```
qualities.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadMultipleItems
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            DAVtqResult[] vtqResults = client.ReadMultipleItems("OPCLabs.KitServer.2",
                new DAItemDescriptor[]
                    {
                        "Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1 min)",
"Simulation.Register_I4"
                    });

            for (int i = 0; i < vtqResults.Length; i++)
            {
                Debug.Assert(vtqResults[i] != null);

                if (vtqResults[i].Succeeded)
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults[i].Vtq);
                else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults[i].ErrorMessageBrief);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadMultipleItems
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems(
                "OPCLabs.KitServer.2",
                New DAItemDescriptor() {"Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1
min)", "Simulation.Register_I4"})

            For i = 0 To vtqResults.Length - 1
                Debug.Assert(vtqResults(i) IsNot Nothing)

                If vtqResults(i).Succeeded Then
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults(i).Vtq)
                Else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults(i).ErrorMessageBrief)
                End If
            Next i
        End Sub
    End Class
```

```
End Namespace
```

Object Pascal

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

class procedure ReadMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ReadItemArguments1: _DAReadItemArguments;
  ReadItemArguments2: _DAReadItemArguments;
  ReadItemArguments3: _DAReadItemArguments;
  ReadItemArguments4: _DAReadItemArguments;
  VtqResult: _DAVtqResult;
  Results: OleVariant;
begin
  ReadItemArguments1 := CoDAReadItemArguments.Create;
  ReadItemArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments1.ItemDescriptor.ItemID := 'Simulation.Random';

  ReadItemArguments2 := CoDAReadItemArguments.Create;
  ReadItemArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments2.ItemDescriptor.ItemID := 'Trends.Ramp (1 min)';

  ReadItemArguments3 := CoDAReadItemArguments.Create;
  ReadItemArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments3.ItemDescriptor.ItemID := 'Trends.Sine (1 min)';

  ReadItemArguments4 := CoDAReadItemArguments.Create;
  ReadItemArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments4.ItemDescriptor.ItemID := 'Simulation.Register_I4';

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ReadItemArguments1;
  Arguments[1] := ReadItemArguments2;
  Arguments[2] := ReadItemArguments3;
  Arguments[3] := ReadItemArguments4;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.ReadMultipleItems(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
      VtqResult := IInterface(Results[I]) as _DAVtqResult;
      if VtqResult.Succeeded then
        WriteLn('results(', i, ').Vtq.ToString(): ', VtqResult.Vtq.ToString)
      else
        WriteLn('results(', i, ') *** Failure: ', VtqResult.ErrorMessageBrief);
  end;
end;
```

PowerScript

```
// This example shows how to read the values of 4 different items at once.

mle_outputtext.Text = ""
```

```
// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare arguments.

OLEObject readItemArguments1
readItemArguments1 = CREATE OLEObject
readItemArguments1.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

OLEObject readItemArguments2
readItemArguments2 = CREATE OLEObject
readItemArguments2.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

OLEObject readItemArguments3
readItemArguments3 = CREATE OLEObject
readItemArguments3.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

OLEObject readItemArguments4
readItemArguments4 = CREATE OLEObject
readItemArguments4.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

OLEObject readItemArgumentsList
readItemArgumentsList = CREATE OLEObject
readItemArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readItemArgumentsList.Add(readItemArguments1)
readItemArgumentsList.Add(readItemArguments2)
readItemArgumentsList.Add(readItemArguments3)
readItemArgumentsList.Add(readItemArguments4)

// Obtain values.
OLEObject valueResultList
valueResultList = client.ReadItemValueList(readItemArgumentsList)

// Display results
Int i
FOR i = 0 TO valueResultList.Count - 1
    OLEObject valueResult
    valueResult = valueResultList.Item[i]
    IF valueResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "valueResult[" + String(i) + "].Value: " +
String(valueResult.Value) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "valueResult[" + String(i) + "] *** Failure: " +
valueResult.ErrorMessageBrief + "~r~n"
    END IF
NEXT
```

VBScript

```
Rem This example shows how to read 4 items at once, and display their values, timestamps and
qualities.
```

```
Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

13.2.5.27 Examples - OPC Data Access - Read multiple items

C#

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadMultipleItems
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            DAVtqResult[] vtqResults = client.ReadMultipleItems("OPCLabs.KitServer.2",
```

```
                new DAItemDescriptor[]
                    {
                        "Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1 min)",
"Simulation.Register_I4"
                    });

            for (int i = 0; i < vtqResults.Length; i++)
            {
                Debug.Assert(vtqResults[i] != null);

                if (vtqResults[i].Succeeded)
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults[i].Vtq);
                else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults[i].ErrorMessageBrief);
            }
        }
    }
}
```

PHP

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

$ReadItemArguments1 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments1->ItemDescriptor->ItemID = "Simulation.Random";

$ReadItemArguments2 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments2->ItemDescriptor->ItemID = "Trends.Ramp (1 min)";

$ReadItemArguments3 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments3->ItemDescriptor->ItemID = "Trends.Sine (1 min)";

$ReadItemArguments4 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments");
$ReadItemArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ReadItemArguments4->ItemDescriptor->ItemID = "Simulation.Register_I4";

$arguments[0] = $ReadItemArguments1;
$arguments[1] = $ReadItemArguments2;
$arguments[2] = $ReadItemArguments3;
$arguments[3] = $ReadItemArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

$results = $Client->ReadMultipleItems($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $VtqResult = $results[$i];
    if ($VtqResult->Succeeded)
        printf("results[%d].Vtq.ToString(): %s\n", $i, $VtqResult->Vtq->ToString);
    else
        printf("results[%d]: *** Failure: %s\n", $i, $VtqResult->ErrorMessageBrief);
}
```

VB.NET

```
' This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Imports OpcLabs.EasyOpc.DataAccess
```

```
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadMultipleItems
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems(
                "OPCLabs.KitServer.2",
                New DAItemDescriptor() {"Simulation.Random", "Trends.Ramp (1 min)", "Trends.Sine (1
min)", "Simulation.Register_I4"})

            For i = 0 To vtqResults.Length - 1
                Debug.Assert(vtqResults(i) IsNot Nothing)

                If vtqResults(i).Succeeded Then
                    Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults(i).Vtq)
                Else
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults(i).ErrorMessageBrief)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

class procedure ReadMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ReadItemArguments1: _DAReadItemArguments;
  ReadItemArguments2: _DAReadItemArguments;
  ReadItemArguments3: _DAReadItemArguments;
  ReadItemArguments4: _DAReadItemArguments;
  VtqResult: _DAVtqResult;
  Results: OleVariant;
begin
  ReadItemArguments1 := CoDAReadItemArguments.Create;
  ReadItemArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments1.ItemDescriptor.ItemID := 'Simulation.Random';

  ReadItemArguments2 := CoDAReadItemArguments.Create;
  ReadItemArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments2.ItemDescriptor.ItemID := 'Trends.Ramp (1 min)';

  ReadItemArguments3 := CoDAReadItemArguments.Create;
  ReadItemArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments3.ItemDescriptor.ItemID := 'Trends.Sine (1 min)';

  ReadItemArguments4 := CoDAReadItemArguments.Create;
  ReadItemArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ReadItemArguments4.ItemDescriptor.ItemID := 'Simulation.Register_I4';

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ReadItemArguments1;
  Arguments[1] := ReadItemArguments2;
  Arguments[2] := ReadItemArguments3;
  Arguments[3] := ReadItemArguments4;

  // Instantiate the client object
```

```
  Client := CoEasyDAClient.Create;

  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.ReadMultipleItems(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
      VtqResult := IInterface(Results[I]) as _DAVtqResult;
      if VtqResult.Succeeded then
        WriteLn('results(', i, ').Vtq.ToString(): ', VtqResult.Vtq.ToString)
      else
        WriteLn('results(', i, ') *** Failure: ', VtqResult.ErrorMessageBrief);
  end;
end;
```

PowerScript

```
// This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare arguments.

OLEObject readItemArguments1
readItemArguments1 = CREATE OLEObject
readItemArguments1.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

OLEObject readItemArguments2
readItemArguments2 = CREATE OLEObject
readItemArguments2.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

OLEObject readItemArguments3
readItemArguments3 = CREATE OLEObject
readItemArguments3.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

OLEObject readItemArguments4
readItemArguments4 = CREATE OLEObject
readItemArguments4.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")

readItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
readItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

OLEObject readItemArgumentsList
readItemArgumentsList = CREATE OLEObject
readItemArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readItemArgumentsList.Add(readItemArguments1)
readItemArgumentsList.Add(readItemArguments2)
readItemArgumentsList.Add(readItemArguments3)
readItemArgumentsList.Add(readItemArguments4)
```

```
// Obtain values/timestamps/qualities.
OLEObject vtqResultList
vtqResultList = client.ReadItemList(readItemArgumentsList)

// Display results
Int i
FOR i = 0 TO vtqResultList.Count - 1
    OLEObject vtqResult
    vtqResult = vtqResultList.Item[i]
    IF vtqResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "vtqResult[" + String(i) + "].Vtq: " +
String(vtqResult.Vtq) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "vtqResult[" + String(i) + "] *** Failure: " +
vtqResult.ErrorMessageBrief + "~r~n"
    END IF
NEXT
```

VBScript

```
Rem This example shows how to read 4 items at once, and display their values, timestamps and
qualities.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

Dim arguments(3)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

VBScript

```
Rem This example reads a large number of items.

Option Explicit

Const repeatCount = 10
Const numberOfItems = 1000

WScript.Echo "Creating array of arguments..."
Dim arguments(): ReDim arguments(numberOfItems - 1)
Dim i: For i = 0 To numberOfItems - 1
    Dim copy: copy = Int(i / 100) + 1
    Dim phase: phase = i Mod 100
    Dim itemId: itemId = "Simulation.Incrementing.Copy_" & copy & ".Phase_" & phase
    WScript.Echo itemId

    Dim ReadItemArguments: Set ReadItemArguments =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
    ReadItemArguments.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
    ReadItemArguments.ItemDescriptor.ItemID = itemId

    Set arguments(i) = ReadItemArguments
Next

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

Dim iRepeat: For iRepeat = 1 To repeatCount
    WScript.Echo "Reading items..."
    Dim results: results = Client.ReadMultipleItems(arguments)

    Dim successCount: successCount = 0
    For i = LBound(results) To UBound(results)
        Dim VtqResult: Set VtqResult = results(i)
        If VtqResult.Succeeded Then successCount = successCount + 1
    Next
    WScript.Echo "Success count: " & successCount
Next
```

VBScript

```
Rem This example shows how to read 4 items at once synchronously, and display their values,
timestamps and qualities.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"

Dim ReadItemArguments4: Set ReadItemArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"

Dim arguments(3)
```

```
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3
Set arguments(3) = ReadItemArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

' Specify that only synchronous method is allowed. By default, both synchronous and asynchronous
methods are allowed, and
' the components picks a suitable method automatically. Disallowing asynchronous method leaves only
the synchronous method
' available for selection.
Client.InstanceParameters.Mode.AllowAsynchronousMethod = False

Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

VBScript

```
Rem This example shows how to read 2 items (first valid, second invalid), test for success of each
read and display either
Rem the DAVtq or the Exception.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "UnknownItem"

Dim arguments(1)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim VtqResult: Set VtqResult = results(i)
    If VtqResult.Succeeded Then
        WScript.Echo "results(" & i & ").Vtq.ToString(): " & VtqResult.Vtq.ToString()
    Else
        WScript.Echo "results(" & i & ") *** Failure: " & VtqResult.ErrorMessageBrief
    End If
Next
```

VBScript

```
Rem This example attempts to read 3 items (first valid, second invalid, third valid again), and
display their values,
Rem timestamps and qualities. Without testing for a success, a run-time error occurs when accessing
the Vtq property
```

```
Rem of a failed result.

Option Explicit

Dim ReadItemArguments1: Set ReadItemArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments1.ItemDescriptor.ItemID = "Simulation.Random"

Dim ReadItemArguments2: Set ReadItemArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments2.ItemDescriptor.ItemID = "UnknownItem"

Dim ReadItemArguments3: Set ReadItemArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAReadItemArguments")
ReadItemArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ReadItemArguments3.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"

Dim arguments(2)
Set arguments(0) = ReadItemArguments1
Set arguments(1) = ReadItemArguments2
Set arguments(2) = ReadItemArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.ReadMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    ' This is an anti-example - you should not access the Vtq property without testing the success
first!
    WScript.Echo "results(" & i & ").Vtq.ToString(): " & results(i).Vtq.ToString()
Next
```

13.2.5.28 Examples - OPC Data Access - Read multiple items and measure time

C#

```
// This example measures the time needed to read 2000 items all at once, and in 20 groups by 100
items.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class ReadMultipleItems
    {
        const int NumberOfGroups = 100;
        const int ItemsInGroup = 20;
        private const int TotalItems = NumberOfGroups * ItemsInGroup;

        // Main method
        public static void TimeMeasurements()
        {
            // Make the measurements 10 times; note that first time the times might be longer.
            for (int i = 1; i <= 10; i++)
            {
```

```
                // Pause - we do not want the component to use the values it has in memory
                Thread.Sleep(2 * 1000);

                // Read all items at once, and measure the time
                var stopwatch1 = new Stopwatch();
                stopwatch1.Start();
                ReadAllAtOnce();
                stopwatch1.Stop();
                Console.WriteLine("ReadAllAtOnce has taken (milliseconds): {0}",
stopwatch1.ElapsedMilliseconds);

                // Pause - we do not want the component to use the values it has in memory
                Thread.Sleep(2 * 1000);

                // Read items in groups, and measure the time
                var stopwatch2 = new Stopwatch();
                stopwatch2.Start();
                ReadInGroups();
                stopwatch2.Stop();
                Console.WriteLine("ReadInGroups has taken (milliseconds): {0}",
stopwatch2.ElapsedMilliseconds);
            }

            // Example output (measured with Version 5.20, Release configuration):
            /*
                ReadAllAtOnce has taken (milliseconds): 3432
                ReadInGroups has taken (milliseconds): 1563
                ReadAllAtOnce has taken (milliseconds): 539
                ReadInGroups has taken (milliseconds): 1625
                ReadAllAtOnce has taken (milliseconds): 579
                ReadInGroups has taken (milliseconds): 1594
                ReadAllAtOnce has taken (milliseconds): 638
                ReadInGroups has taken (milliseconds): 1610
                ...
            */

            // Note that Version 5.12 and earlier were yielding much larger penalty to repeated
reads.
            // Example output (measured with Version 5.12, Release configuration):
            /*
                ReadAllAtOnce has taken (milliseconds): 4241
                ReadInGroups has taken (milliseconds): 8094
                ReadAllAtOnce has taken (milliseconds): 269
                ReadInGroups has taken (milliseconds): 7813
                ReadAllAtOnce has taken (milliseconds): 285
                ReadInGroups has taken (milliseconds): 7813
                ReadAllAtOnce has taken (milliseconds): 283
                ReadInGroups has taken (milliseconds): 7844
                ...
            */
        }

        // Read all items at once
        private static void ReadAllAtOnce()
        {
            var client = new EasyDAClient();

            // Create an array of item descriptors for all items
            var itemDescriptors = new DAItemDescriptor[TotalItems];
            int index = 0;
            for (int iLoop = 0; iLoop < NumberOfGroups; iLoop++)
                for (int iItem = 0; iItem < ItemsInGroup; iItem++)
                    itemDescriptors[index++] = new DAItemDescriptor(
```

```
                        String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", iLoop + 1,
iItem + 1));

            // Perform the OPC read
            DAVtqResult[] vtqResults = client.ReadMultipleItems("OPCLabs.KitServer.2",
itemDescriptors);

            // Count successful results
            int successCount = 0;
            for (int iItem = 0; iItem < TotalItems; iItem++)
            {
                Debug.Assert(vtqResults[iItem] != null);
                if (vtqResults[iItem].Succeeded)
                    successCount++;
            }

            if (successCount != TotalItems)
                Console.WriteLine("Warning: There were some failures, success count is {0}",
successCount);
        }

        // Read items in groups
        private static void ReadInGroups()
        {
            var client = new EasyDAClient();

            int successCount = 0;
            for (int iLoop = 0; iLoop < NumberOfGroups; iLoop++)
            {
                // Create an array of item descriptors for items in one group
                var itemDescriptors = new DAItemDescriptor[ItemsInGroup];
                for (int iItem = 0; iItem < ItemsInGroup; iItem++)
                    itemDescriptors[iItem] = new DAItemDescriptor(
                        String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", iLoop + 1,
iItem + 1));

                // Perform the OPC read
                DAVtqResult[] vtqResults = client.ReadMultipleItems("OPCLabs.KitServer.2",
itemDescriptors);

                // Count successful results (totalling to previous value)
                for (int iItem = 0; iItem < ItemsInGroup; iItem++)
                {
                    Debug.Assert(vtqResults[iItem] != null);
                    if (vtqResults[iItem].Succeeded) successCount++;
                }
            }

            if (successCount != TotalItems)
                Console.WriteLine("Warning: There were some failures, success count is {0}",
successCount);
        }
    }
}
```

VB.NET

```
' This example measures the time needed to read 2000 items all at once, and in 20 groups by 100
items.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
```

```
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class ReadMultipleItems
        Private Const NumberOfGroups As Integer = 100
        Private Const ItemsInGroup As Integer = 20
        Private Const TotalItems As Integer = NumberOfGroups * ItemsInGroup

        ' Main method
        Public Shared Sub TimeMeasurements()
            ' Make the measurements 10 times; note that first time the times might be longer.
            For i As Integer = 1 To 10
                ' Pause - we do not want the component to use the values it has in memory
                Thread.Sleep(2 * 1000)

                ' Read all items at once, and measure the time
                Dim stopwatch1 = New Stopwatch()
                stopwatch1.Start()
                ReadAllAtOnce()
                stopwatch1.Stop()
                Console.WriteLine("ReadAllAtOnce has taken (milliseconds): {0}",
stopwatch1.ElapsedMilliseconds)

                ' Pause - we do not want the component to use the values it has in memory
                Thread.Sleep(2 * 1000)

                ' Read items in groups, and measure the time
                Dim stopwatch2 = New Stopwatch()
                stopwatch2.Start()
                ReadInGroups()
                stopwatch2.Stop()
                Console.WriteLine("ReadInGroups has taken (milliseconds): {0}",
stopwatch2.ElapsedMilliseconds)
            Next i

            ' Example output (measured with Version 5.20, Release configuration):
            '
            '                     ReadAllAtOnce has taken (milliseconds): 3432
            '                     ReadInGroups has taken (milliseconds): 1563
            '                     ReadAllAtOnce has taken (milliseconds): 539
            '                     ReadInGroups has taken (milliseconds): 1625
            '                     ReadAllAtOnce has taken (milliseconds): 579
            '                     ReadInGroups has taken (milliseconds): 1594
            '                     ReadAllAtOnce has taken (milliseconds): 638
            '                     ReadInGroups has taken (milliseconds): 1610
            '                     ...
            '

            ' Note that Version 5.12 and earlier were yielding much larger penalty to repeated
reads.
            ' Example output (measured with Version 5.12, Release configuration):
            '
            '                     ReadAllAtOnce has taken (milliseconds): 4241
            '                     ReadInGroups has taken (milliseconds): 8094
            '                     ReadAllAtOnce has taken (milliseconds): 269
            '                     ReadInGroups has taken (milliseconds): 7813
            '                     ReadAllAtOnce has taken (milliseconds): 285
            '                     ReadInGroups has taken (milliseconds): 7813
            '                     ReadAllAtOnce has taken (milliseconds): 283
            '                     ReadInGroups has taken (milliseconds): 7844
            '                     ...
            '
        End Sub
```

```
        ' Read all items at once
        Private Shared Sub ReadAllAtOnce()
            Dim client = New EasyDAClient()

            ' Create an array of item descriptors for all items
            Dim itemDescriptors = New DAItemDescriptor(TotalItems - 1) {}
            Dim index As Integer = 0
            For iLoop As Integer = 0 To NumberOfGroups - 1
                For iItem As Integer = 0 To ItemsInGroup - 1
                    itemDescriptors(index) = New
DAItemDescriptor(String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", iLoop + 1, iItem +
1))
                    index += 1
                Next iItem
            Next iLoop

            ' Perform the OPC read
            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems("OPCLabs.KitServer.2",
itemDescriptors)

            ' Count successful results
            Dim successCount As Integer = 0
            For iItem As Integer = 0 To TotalItems - 1
                Debug.Assert(vtqResults(iItem) IsNot Nothing)
                If vtqResults(iItem).Succeeded Then
                    successCount += 1
                End If
            Next iItem

            If successCount <> TotalItems Then
                Console.WriteLine("Warning: There were some failures, success count is {0}",
successCount)
            End If
        End Sub

        ' Read items in groups
        Private Shared Sub ReadInGroups()
            Dim client = New EasyDAClient()

            Dim successCount As Integer = 0
            For iLoop As Integer = 0 To NumberOfGroups - 1
                ' Create an array of item descriptors for items in one group
                Dim itemDescriptors = New DAItemDescriptor(ItemsInGroup - 1) {}
                For iItem As Integer = 0 To ItemsInGroup - 1
                    itemDescriptors(iItem) = New
DAItemDescriptor(String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", iLoop + 1, iItem +
1))
                Next iItem

                ' Perform the OPC read
                Dim vtqResults() As DAVtqResult =
client.ReadMultipleItems("OPCLabs.KitServer.2", itemDescriptors)

                ' Count successful results (totalling to previous value)
                For iItem As Integer = 0 To ItemsInGroup - 1
                    Debug.Assert(vtqResults(iItem) IsNot Nothing)
                    If vtqResults(iItem).Succeeded Then
                        successCount += 1
                    End If
                Next iItem
            Next iLoop
```

```
            If successCount <> TotalItems Then
                Console.WriteLine("Warning: There were some failures, success count is {0}",
successCount)
            End If
        End Sub
    End Class
End Namespace
```

13.2.5.29 Examples - OPC Data Access - Setting a hold period

C#

```
// This example shows how the OPC server can quickly be disconnected after writing a
value into one of its OPC items.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClientHoldPeriods
{
    class TopicWrite
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            client.InstanceParameters.HoldPeriods.TopicWrite = 100; // in milliseconds

            try
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how the OPC server can quickly be disconnected after writing a
value into one of its OPC items.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClientHoldPeriods
    Friend Class TopicWrite
```

```
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            client.InstanceParameters.HoldPeriods.TopicWrite = 100 ' in milliseconds

            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
            End Try
        End Sub
    End Class
End Namespace
```

13.2.5.30 Examples - OPC Data Access - Subscribe to a single item

C#

```
// Hooking up events and receiving OPC item changes.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeItem
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                var eventHandler = new
EasyDAItemChangedEventHandler(client_ItemChanged);
                client.ItemChanged += eventHandler;

                Console.WriteLine("Subscribing item...");
                client.SubscribeItem("", "OPCLabs.KitServer.2", "Demo.Ramp", 200);
                Thread.Sleep(30 * 1000);
                client.UnsubscribeAllItems();
                client.ItemChanged -= eventHandler;
            }
        }

        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
```

```
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' Hooking up events and receiving OPC item changes.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeItem
        Shared Sub Main1()
            Using client = New EasyDAClient()
                Dim eventHandler = New EasyDAItemChangedEventHandler(AddressOf
client_ItemChanged)
                AddHandler client.ItemChanged, eventHandler

                Console.WriteLine("Subscribing item...")
                client.SubscribeItem("", "OPCLabs.KitServer.2", "Demo.Ramp", 200)
                Thread.Sleep(30 * 1000)
                client.UnsubscribeAllItems()
                RemoveHandler client.ItemChanged, eventHandler
            End Using
        End Sub

        Private Shared Sub client_ItemChanged(sender As Object, e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a single item and display the
value of the item with each change.

type
  TSubscribeItem_ClientEventHandlers = class
    // Item changed event handler
    procedure OnItemChanged(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyDAItemChangedEventArgs);
  end;

procedure TSubscribeItem_ClientEventHandlers.OnItemChanged(
  ASender: TObject;
```

```
  sender: OleVariant;
  const eventArgs: _EasyDAItemChangedEventArgs);
begin
  if eventArgs.Succeeded then
    WriteLn(eventArgs.Vtq.ToString)
  else
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
end;

class procedure SubscribeItem.Main;
var
  Client: TEasyDAClient;
  ClientEventHandlers: TSubscribeItem_ClientEventHandlers;
begin
  // Instantiate the client object and hook events
  Client := TEasyDAClient.Create(nil);
  ClientEventHandlers := TSubscribeItem_ClientEventHandlers.Create;
  Client.OnItemChanged := ClientEventHandlers.OnItemChanged;

  Client.SubscribeItem('', 'OPCLabs.KitServer.2', 'Simulation.Random', 1000);

  WriteLn('Processing item changed events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a single item and display the
value of the item with each change.
//
// Some related documentation: http://php.net/manual/en/function.com-event-sink.php .
Pay attention to the comment that says
// "Be careful how you use this feature; if you are doing something similar to the
example below, then it doesn't really make
// sense to run it in a web server context.". What they are trying to say is that
processing a web request should be
// a short-lived code, which does not fit well with the idea of being subscribed to
events and received them over longer time.
// It is possible to write such code, but it is only useful when processing the request
is allowed to take relatively long. Or,
// when you are using PHP from command-line, or otherwise - not to serve a web page
directly.
//
// Subscribing to QuickOPC-COM events in the context of PHP Web application, while not
imposing the limitations to the request
// processing time, has to be "worked around", e.g. using the "event pull" mechanism.
```

```
class DEasyDAClientEvents {
    function ItemChanged($varSender, $varE)
    {
        if ($varE->Succeeded)
        {
        print $varE->Vtq->ToString();
            print "\n";
        }
        else
        {
            printf("*** Failure: %s\n", $varE->ErrorMessageBrief);
        }
    }
}

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$Events = new DEasyDAClientEvents();
com_event_sink($Client, $Events, "DEasyDAClientEvents");

$Client->SubscribeItem("", "OPCLabs.KitServer.2", "Simulation.Random", 1000);

print "Processing item changed events for 1 minute...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

VBScript

```
Rem This example shows how to subscribe to changes of a single item and display the
value of the item with each change.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

Client.SubscribeItem "", "OPCLabs.KitServer.2", "Simulation.Random", 1000

WScript.Echo "Processing item changed events for 1 minute..."
WScript.Sleep 60*1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq.ToString
End Sub
```

VBScript

```
Rem This example subscribes to changes of 2 items separately, and displays rich
information available with each item changed
Rem event notification.

Option Explicit
```

```
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

Client.SubscribeItem "", "OPCLabs.KitServer.2", "Simulation.Random", 5*1000
Client.SubscribeItem "", "OPCLabs.KitServer.2", "Trends.Ramp (1 min)", 5*1000

WScript.Echo "Processing item changed events for 1 minute..."
WScript.Sleep 60*1000



Sub Client_ItemChanged(Sender, e)
    On Error Resume Next
    WScript.Echo
    WScript.Echo "e.Arguments.State: " & e.Arguments.State
    WScript.Echo "e.Arguments.ServerDescriptor.MachineName: " &
e.Arguments.ServerDescriptor.MachineName
    WScript.Echo "e.Arguments.ServerDescriptor.ServerClass: " &
e.Arguments.ServerDescriptor.ServerClass
    WScript.Echo "e.Arguments.ItemDescriptor.ItemId: " &
e.Arguments.ItemDescriptor.ItemId
    WScript.Echo "e.Arguments.ItemDescriptor.AccessPath: " &
e.Arguments.ItemDescriptor.AccessPath
    WScript.Echo "e.Arguments.ItemDescriptor.RequestedDataType: " &
e.Arguments.ItemDescriptor.RequestedDataType
    WScript.Echo "e.Arguments.GroupParameters.Locale: " &
e.Arguments.GroupParameters.Locale
    WScript.Echo "e.Arguments.GroupParameters.RequestedUpdateRate: " &
e.Arguments.GroupParameters.RequestedUpdateRate
    WScript.Echo "e.Arguments.GroupParameters.PercentDeadband: " &
e.Arguments.GroupParameters.PercentDeadband
    WScript.Echo "e.Exception.Message: " & e.Exception.Message
    WScript.Echo "e.Exception.Source: " & e.Exception.Source
    WScript.Echo "e.Exception.ErrorCode: " & e.Exception.ErrorCode
    WScript.Echo "e.Vtq.Value: " & e.Vtq.Value
    WScript.Echo "e.Vtq.Timestamp: " & e.Vtq.Timestamp
    WScript.Echo "e.Vtq.TimestampLocal: " & e.Vtq.TimestampLocal
    WScript.Echo "e.Vtq.Quality: " & e.Vtq.Quality
End Sub
```

13.2.5.31 Examples - OPC Data Access - Subscribe to items of various data types

C#

```
// Shows how different data types can be subscribed to, including rare types and arrays
of values.

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading;
```

```
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeMultipleItems
    {
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            Console.WriteLine();
            Console.WriteLine("ItemDescriptor.Arguments.ItemId: {0}",
e.Arguments.ItemDescriptor.ItemId);

            if (e.Succeeded)
            {
                Debug.Assert(e.Vtq != null);
                Console.WriteLine("Vtq: {0}", e.Vtq);
            }
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }

        public static void DataTypes()
        {
            IEnumerable<DAItemGroupArguments> arguments = new[]
                {
                    "Simulation.Register_EMPTY",
                    "Simulation.Register_NULL",
                    "Simulation.Register_DISPATCH",

                    "Simulation.ReadValue_I2",
                    "Simulation.ReadValue_I4",
                    "Simulation.ReadValue_R4",
                    "Simulation.ReadValue_R8",
                    "Simulation.ReadValue_CY",
                    "Simulation.ReadValue_DATE",
                    "Simulation.ReadValue_BSTR",
                    "Simulation.ReadValue_BOOL",
                    "Simulation.ReadValue_DECIMAL",
                    "Simulation.ReadValue_I1",
                    "Simulation.ReadValue_UI1",
                    "Simulation.ReadValue_UI2",
                    "Simulation.ReadValue_UI4",
                    "Simulation.ReadValue_INT",
                    "Simulation.ReadValue_UINT",

                    "Simulation.ReadValue_ArrayOfI2",
                    "Simulation.ReadValue_ArrayOfI4",
                    "Simulation.ReadValue_ArrayOfR4",
                    "Simulation.ReadValue_ArrayOfR8",
                    "Simulation.ReadValue_ArrayOfCY",
                    "Simulation.ReadValue_ArrayOfDATE",
                    "Simulation.ReadValue_ArrayOfBSTR",
                    "Simulation.ReadValue_ArrayOfBOOL",
                    //"Simulation.ReadValue_ArrayOfDECIMAL",
                    "Simulation.ReadValue_ArrayOfI1",
                    "Simulation.ReadValue_ArrayOfUI1",
```

```
                    "Simulation.ReadValue_ArrayOfUI2",
                    "Simulation.ReadValue_ArrayOfUI4",
                    "Simulation.ReadValue_ArrayOfINT",
                    "Simulation.ReadValue_ArrayOfUINT",
                }.Select(itemId => new DAItemGroupArguments("", "OPCLabs.KitServer.2",
itemId, 3 * 1000, null));

            var eventHandler = new EasyDAItemChangedEventHandler(client_ItemChanged);
            var client = new EasyDAClient();
            client.ItemChanged += eventHandler;

            Console.WriteLine("Subscribing items...");
            client.SubscribeMultipleItems(arguments.ToArray());
            Thread.Sleep(30 * 1000);
            client.UnsubscribeAllItems();
            client.ItemChanged -= eventHandler;
        }
    }
}
```

VB.NET

```
' Shows how different data types can be subscribed to, including rare types and arrays
of values.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeMultipleItems
        Private Shared Sub client_ItemChanged(sender As Object, e As
EasyDAItemChangedEventArgs)

            Console.WriteLine()
            Console.WriteLine("ItemDescriptor.Arguments.ItemId: {0}",
e.Arguments.ItemDescriptor.ItemId)
            If e.Succeeded Then
                Debug.Assert(e.Vtq IsNot Nothing)
                Console.WriteLine("Vtq: {0}", e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub

        Shared Sub DataTypes()
            Dim arguments As IEnumerable(Of DAItemGroupArguments) = New String() _
            { _
                "Simulation.Register_EMPTY",
                "Simulation.Register_NULL",
                "Simulation.Register_DISPATCH",
 _
                "Simulation.ReadValue_I2",
                "Simulation.ReadValue_I4",
                "Simulation.ReadValue_R4",
                "Simulation.ReadValue_R8",
                "Simulation.ReadValue_CY",
```

```
                "Simulation.ReadValue_DATE",
                "Simulation.ReadValue_BSTR",
                "Simulation.ReadValue_BOOL",
                "Simulation.ReadValue_DECIMAL",
                "Simulation.ReadValue_I1",
                "Simulation.ReadValue_UI1",
                "Simulation.ReadValue_UI2",
                "Simulation.ReadValue_UI4",
                "Simulation.ReadValue_INT",
                "Simulation.ReadValue_UINT",
 _
                "Simulation.ReadValue_ArrayOfI2",
                "Simulation.ReadValue_ArrayOfI4",
                "Simulation.ReadValue_ArrayOfR4",
                "Simulation.ReadValue_ArrayOfR8",
                "Simulation.ReadValue_ArrayOfCY",
                "Simulation.ReadValue_ArrayOfDATE",
                "Simulation.ReadValue_ArrayOfBSTR",
                "Simulation.ReadValue_ArrayOfBOOL",
                "Simulation.ReadValue_ArrayOfI1",
                "Simulation.ReadValue_ArrayOfUI1",
                "Simulation.ReadValue_ArrayOfUI2",
                "Simulation.ReadValue_ArrayOfUI4",
                "Simulation.ReadValue_ArrayOfINT",
                "Simulation.ReadValue_ArrayOfUINT"
            } _
            .Select(Function(itemId) New DAItemGroupArguments("",
"OPCLabs.KitServer.2", itemId, 3 * 1000, Nothing))

            Console.WriteLine()

            Dim client As New EasyDAClient()
            Dim eventHandler = New EasyDAItemChangedEventHandler(AddressOf
client_ItemChanged)
            AddHandler client.ItemChanged, eventHandler

            Console.WriteLine("Subscribing items...")
            client.SubscribeMultipleItems(arguments.ToArray())
            Thread.Sleep(30 * 1000)
            client.UnsubscribeAllItems()
            RemoveHandler client.ItemChanged, eventHandler
        End Sub
    End Class
End Namespace
```

13.2.5.32 Examples - OPC Data Access - Subscribe to large number of items

C#

```
// This example shows how subscribe to large number of items.
```

```
using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeMultipleItems
    {
        public static void ManyItems()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged_ManyItems;

                const int numberOfItems = 1000;

                Console.WriteLine("Preparing arguments...");
                var argumentArray = new DAItemGroupArguments[numberOfItems];
                for (int i = 0; i < numberOfItems; i++)
                {
                    int copy = (i / 100) + 1;
                    int phase = (i % 100) + 1;
                    string itemId =
String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", copy, phase);
                    argumentArray[i] = new DAItemGroupArguments("",
"OPCLabs.KitServer.2", itemId, 50, null);
                }

                Console.WriteLine("Subscribing to {0} items...", numberOfItems);
                client.SubscribeMultipleItems(argumentArray);

                Console.WriteLine("Processing item changed events for 1 minute...");
                Thread.Sleep(60 * 1000);
            }
        }

        // Item changed event handler
        static void client_ItemChanged_ManyItems(object sender,
EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId,
e.Vtq);
            else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to large number of items.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
```

```
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeMultipleItems
        Public Shared Sub ManyItems()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged_ManyItems

                Const numberOfItems As Integer = 1000

                Console.WriteLine("Preparing arguments...")
                Dim argumentArray = New DAItemGroupArguments(numberOfItems - 1) {}
                For i As Integer = 0 To numberOfItems - 1
                    Dim copy As Integer = (i \ 100) + 1
                    Dim phase As Integer = (i Mod 100) + 1
                    Dim itemId As String =
String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}", copy, phase)
                    argumentArray(i) = New DAItemGroupArguments("",
"OPCLabs.KitServer.2", itemId, 50, Nothing)
                Next i

                Console.WriteLine("Subscribing to {0} items...", numberOfItems)
                client.SubscribeMultipleItems(argumentArray)

                Console.WriteLine("Processing item changed events for 1 minute...")
                Thread.Sleep(60 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged_ManyItems(ByVal sender As Object, ByVal e
As EasyDAItemChangedEventArgs)
            ' Display the data
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq)
            Else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

13.2.5.33 Examples - OPC Data Access - Subscribe to multiple items

C#

```
// This example shows how subscribe to changes of multiple items and display the value
of the item with each change.

using System;
```

```
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class SubscribeMultipleItems
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_Main1_ItemChanged;

                client.SubscribeMultipleItems(
                    new[] {
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Sine (1 min)", 1000, null),
                            new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, null)
                        });

                Console.WriteLine("Processing item changed events for 1 minute...");
                Thread.Sleep(60 * 1000);
            }
        }

        // Item changed event handler
        static void client_Main1_ItemChanged(object sender, EasyDAItemChangedEventArgs
e)
        {
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq);
            else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of multiple items and display the value of
the item with each change.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class SubscribeMultipleItems
        Public Shared Sub Main1()
            Using client = New EasyDAClient()
```

```
                AddHandler client.ItemChanged, AddressOf client_ItemChanged_Main1

                client.SubscribeMultipleItems(New DAItemGroupArguments() { _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, Nothing), _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Ramp (1
min)", 1000, Nothing), _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2", "Trends.Sine (1
min)", 1000, Nothing), _
                 New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, Nothing) _
                })

                Console.WriteLine("Processing item changed events for 1 minute...")
                Thread.Sleep(60 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged_Main1(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs)
            ' Display the data
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq)
            Else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to subscribe to changes of multiple items and display the
value of the item with each change.

class DEasyDAClientEvents {
    function ItemChanged($varSender, $varE)
    {
        if ($varE->Succeeded)
        {
            printf("%s: %s\n", $varE->Arguments->ItemDescriptor->ItemId, $varE->Vtq-
>ToString());
        }
        else
        {
            printf("*** Failure: %s\n", $varE->ErrorMessageBrief);
        }
    }
}

$ItemSubscriptionArguments1 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments1->ItemDescriptor->ItemID = "Simulation.Random";
$ItemSubscriptionArguments1->GroupParameters->RequestedUpdateRate = 1000;
```

```
$ItemSubscriptionArguments2 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments2->ItemDescriptor->ItemID = "Trends.Ramp (1 min)";
$ItemSubscriptionArguments2->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments3 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments3->ItemDescriptor->ItemID = "Trends.Sine (1 min)";
$ItemSubscriptionArguments3->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments4 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments4->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemSubscriptionArguments4->GroupParameters->RequestedUpdateRate = 1000;

$arguments[0] = $ItemSubscriptionArguments1;
$arguments[1] = $ItemSubscriptionArguments2;
$arguments[2] = $ItemSubscriptionArguments3;
$arguments[3] = $ItemSubscriptionArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$Events = new DEasyDAClientEvents();
com_event_sink($Client, $Events, "DEasyDAClientEvents");

$Client->SubscribeMultipleItems($arguments);

print "Processing item changed events for 1 minute...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

Object Pascal

```
// This example shows how to subscribe to changes of multiple items and display the
value of the item with each change.

type
  TSubscribeMultipleItems_ClientEventHandlers = class
    // Item changed event handler
    procedure OnItemChanged(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyDAItemChangedEventArgs);
  end;

procedure TSubscribeMultipleItems_ClientEventHandlers.OnItemChanged(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyDAItemChangedEventArgs);
begin
  if eventArgs.Succeeded then
    WriteLn(eventArgs.Arguments.ItemDescriptor.ItemId, ': ', eventArgs.Vtq.ToString)
  else
    WriteLn(eventArgs.Arguments.ItemDescriptor.ItemId, ' *** Failure: ',
eventArgs.ErrorMessageBrief);
```

```
end;

class procedure SubscribeMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyDAClient;
  ClientEventHandlers: TSubscribeMultipleItems_ClientEventHandlers;
  HandleArray: OleVariant;
  ItemSubscriptionArguments1: _EasyDAItemSubscriptionArguments;
  ItemSubscriptionArguments2: _EasyDAItemSubscriptionArguments;
  ItemSubscriptionArguments3: _EasyDAItemSubscriptionArguments;
  ItemSubscriptionArguments4: _EasyDAItemSubscriptionArguments;
begin
  ItemSubscriptionArguments1 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments1.ItemDescriptor.ItemID := 'Simulation.Random';
  ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate := 1000;

  ItemSubscriptionArguments2 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments2.ItemDescriptor.ItemID := 'Trends.Ramp (1 min)';
  ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate := 1000;

  ItemSubscriptionArguments3 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments3.ItemDescriptor.ItemID := 'Trends.Sine (1 min)';
  ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate := 1000;

  ItemSubscriptionArguments4 := CoEasyDAItemSubscriptionArguments.Create;
  ItemSubscriptionArguments4.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemSubscriptionArguments4.ItemDescriptor.ItemID := 'Simulation.Register_I4';
  ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate := 1000;

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ItemSubscriptionArguments1;
  Arguments[1] := ItemSubscriptionArguments2;
  Arguments[2] := ItemSubscriptionArguments3;
  Arguments[3] := ItemSubscriptionArguments4;

  // Instantiate the client object and hook events
  Client := TEasyDAClient.Create(nil);
  ClientEventHandlers := TSubscribeMultipleItems_ClientEventHandlers.Create;
  Client.OnItemChanged := ClientEventHandlers.OnItemChanged;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleItems(Arguments));

  WriteLn('Processing item changed events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);
```

```
  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VBScript

```
Rem This example shows how to subscribe to changes of multiple items and display the
value of the item with each change.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

Client.SubscribeMultipleItems arguments

WScript.Echo "Processing item changed events for 1 minute..."
WScript.Sleep 60*1000

Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
```

```
        Exit Sub
    End If

    WScript.Echo e.Arguments.ItemDescriptor.ItemId & ": " & e.Vtq
End Sub
```

13.2.5.34 Examples - OPC Data Access - Unsubscribe from a single item

C#

```
// This example shows how subscribe to changes of multiple items, and unsubscribe from
one of them.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    class UnsubscribeItem
    {
        public static void Main1()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged;

                int[] handleArray = client.SubscribeMultipleItems(
                    new[] {
                                new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, null),
                                new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, null),
                                new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Sine (1 min)", 1000, null),
                                new DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, null)
                           });

                Console.WriteLine("Processing item changed events for 30 seconds...");
                Thread.Sleep(30 * 1000);

                Console.WriteLine("Unsubscribing from the first item...");
                client.UnsubscribeItem(handleArray[0]);

                Console.WriteLine();

                Console.WriteLine("Processing item changed events for 30 seconds...");
                Thread.Sleep(30 * 1000);
            }
```

```
        }

        // Item changed event handler
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId,
e.Vtq);
            else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of multiple items, and unsubscribe from
one of them.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class UnsubscribeItem
        Shared Sub Main1()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged

                Dim handleArray = client.SubscribeMultipleItems(
                        New DAItemGroupArguments() _
                        { _
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000, Nothing),
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Ramp (1 min)", 1000, Nothing),
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Trends.Sine (1 min)", 1000, Nothing),
                            New DAItemGroupArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 1000, Nothing)
                        })

                Console.WriteLine("Processing item changed events for 30 seconds...")
                Thread.Sleep(30 * 1000)

                Console.WriteLine("Unsubscribing from the first item...")
                client.UnsubscribeItem(handleArray(0))

                Console.WriteLine()

                Console.WriteLine("Processing item changed events for 30 seconds...")
                Thread.Sleep(30 * 1000)
            End Using
        End Sub

        ' Item changed event handler
```

```
        Private Shared Sub client_ItemChanged(sender As Object, e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.ItemDescriptor.ItemId, e.Vtq)
            Else
                Console.WriteLine("{0} *** Failure: {1}",
e.Arguments.ItemDescriptor.ItemId, e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how unsubscribe from changes of a single item.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeItem("", "OPCLabs.KitServer.2",
"Simulation.Random", 1000)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeItem handle

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Vtq
End Sub
```

13.2.5.35 Examples - OPC Data Access - Unsubscribe from all items

VBScript

```
Rem This example shows how to unsubscribe from changes of all items.
```

```
Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handleArray: handleArray = Client.SubscribeMultipleItems(arguments)

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 10 seconds..."
WScript.Sleep 10*1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
```

```
    End If

    WScript.Echo e.Arguments.ItemDescriptor.ItemId & ": " & e.Vtq
End Sub
```

13.2.5.36 Examples - OPC Data Access - Unsubscribe from multiple items

VBScript

```
Rem This example shows how to unsubscribe from changes of multiple items.

Option Explicit

Dim ItemSubscriptionArguments1: Set ItemSubscriptionArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments1.ItemDescriptor.ItemID = "Simulation.Random"
ItemSubscriptionArguments1.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments2: Set ItemSubscriptionArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments2.ItemDescriptor.ItemID = "Trends.Ramp (1 min)"
ItemSubscriptionArguments2.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments3: Set ItemSubscriptionArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments3.ItemDescriptor.ItemID = "Trends.Sine (1 min)"
ItemSubscriptionArguments3.GroupParameters.RequestedUpdateRate = 1000

Dim ItemSubscriptionArguments4: Set ItemSubscriptionArguments4 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments")

ItemSubscriptionArguments4.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemSubscriptionArguments4.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemSubscriptionArguments4.GroupParameters.RequestedUpdateRate = 1000

Dim arguments(3)
Set arguments(0) = ItemSubscriptionArguments1
Set arguments(1) = ItemSubscriptionArguments2
Set arguments(2) = ItemSubscriptionArguments3
Set arguments(3) = ItemSubscriptionArguments4

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
```

```
Dim handleArray: handleArray = Client.SubscribeMultipleItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from two items..."
Dim handles1(1)
handles1(0) = handleArray(1)
handles1(1) = handleArray(2)
Client.UnsubscribeMultipleItems handles1

WScript.Echo "Processing item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from all remaining items..."
Client.UnsubscribeAllItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_ItemChanged(Sender, e)
    If Not (e.Succeeded) Then
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
        Exit Sub
    End If

    WScript.Echo e.Arguments.ItemDescriptor.ItemId & ": " & e.Vtq
End Sub
```

PHP

```
// This example shows how to unsubscribe from changes of multiple items.

class DEasyDAClientEvents {
    function ItemChanged($varSender, $varE)
    {
        if ($varE->Succeeded)
        {
            printf("%s: %s\n", $varE->Arguments->ItemDescriptor->ItemId, $varE->Vtq-
>ToString());
        }
        else
        {
            printf("*** Failure: %s\n", $varE->ErrorMessageBrief);
        }
    }
}

$ItemSubscriptionArguments1 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
```

```
$ItemSubscriptionArguments1->ItemDescriptor->ItemID = "Simulation.Random";
$ItemSubscriptionArguments1->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments2 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments2->ItemDescriptor->ItemID = "Trends.Ramp (1 min)";
$ItemSubscriptionArguments2->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments3 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments3->ItemDescriptor->ItemID = "Trends.Sine (1 min)";
$ItemSubscriptionArguments3->GroupParameters->RequestedUpdateRate = 1000;

$ItemSubscriptionArguments4 = new
COM("OpcLabs.EasyOpc.DataAccess.OperationModel.EasyDAItemSubscriptionArguments");
$ItemSubscriptionArguments4->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemSubscriptionArguments4->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemSubscriptionArguments4->GroupParameters->RequestedUpdateRate = 1000;

$arguments[0] = $ItemSubscriptionArguments1;
$arguments[1] = $ItemSubscriptionArguments2;
$arguments[2] = $ItemSubscriptionArguments3;
$arguments[3] = $ItemSubscriptionArguments4;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$Events = new DEasyDAClientEvents();
com_event_sink($Client, $Events, "DEasyDAClientEvents");

print "Subscribing...\n";
$handleArray = $Client->SubscribeMultipleItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %s\n", $i, $handleArray[$i]);
}

print "Processing item changed events for 10 seconds...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

print "Unsubscribing from two items...\n";
$handles1[0] = $handleArray[1];
$handles1[1] = $handleArray[2];
$Client->UnsubscribeMultipleItems($handles1);

print "Processing item changed events for 10 seconds...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

print "Unsubscribing from all remaining items...\n";
$Client->UnsubscribeAllItems;

print "Waiting for 5 seconds...\n";
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

13.2.5.37 Examples - OPC Data Access - Write a single value

C#

```
// This example shows how to write a value into a single item.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteItemValue
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            try
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345);
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

PHP

```
// This example shows how to write a value into a single item.

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");

try
{
    $value = $Client->WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}
```

VB.NET

```
' Shows how to write into an OPC item that is of array type, and read the array value
back.

Imports OpcLabs.EasyOpc.DataAccess
```

```
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteItemValue
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to write a value into a single item.

class procedure WriteItemValue.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    Client.WriteItemValue('', 'OPCLabs.KitServer.2', 'Simulation.Register_I4', 12345);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PowerScript

```
// This example shows how to write a value into a single item.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Modify value of an item
TRY
    client.WriteItemValue("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345)
CATCH (OLERuntimeError oleRuntimeError)
```

```
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

VBScript

```
Rem This example shows how to write a value into a single item.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Client.WriteItemValue "", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

13.2.5.38 Examples - OPC Data Access - Write a single value, specify requested data type

C#

```
// This example shows how to write a value into a single item, specifying its requested
data type.

using System;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteItemValue
    {
        public static void RequestedDataType()
        {
            var client = new EasyDAClient();

            try
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345,
                    VarTypes.I4);   // <-- the requested data type
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
```

```
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single item, specifying its requested
data type.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteItemValue
        Public Shared Sub RequestedDataType()
            Dim client = New EasyDAClient()

            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_I4", 12345,
                                      VarTypes.I4) ' <-- the requested data type
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

13.2.5.39 Examples - OPC Data Access - Write an array value

C#

```
// Shows how to write into an OPC item that is of array type, and read the array value
back.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteItemValue
    {
        public static void Array()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Writing array value...");
            try
```

```
            {
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_ArrayOfI2", new Int16[] { 1234, 2345, 3456 });
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            Console.WriteLine("Reading array value...");
            short[] value;
            try
            {
                value = (Int16[])client.ReadItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_ArrayOfI2");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            if (value != null)
            {
                Console.WriteLine(value[0]);
                Console.WriteLine(value[1]);
                Console.WriteLine(value[2]);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single item.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteItemValue
        Public Shared Sub Array()
            Dim client As New EasyDAClient()

            Console.WriteLine("Writing array value...")
            Try
                client.WriteItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_ArrayOfI2", New Int16() {1234, 2345, 3456})
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
```

```
            Console.WriteLine("Reading array value...")
            Dim value As Int16()
            Try
                value = client.ReadItemValue("", "OPCLabs.KitServer.2",
"Simulation.Register_ArrayOfI2")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            If Not IsNothing(value) Then
                Console.WriteLine(value(0))
                Console.WriteLine(value(1))
                Console.WriteLine(value(2))
            End If
        End Sub
    End Class
End Namespace
```

13.2.5.40 Examples - OPC Data Access - Write multiple values

C#

```
// Shows how to write into multiple OPC items using a single method call, and read multiple item values
back.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteMultipleItemValues
    {
        public static void Main1()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Writing multiple item values...");
            OperationResult[] resultArray = client.WriteMultipleItemValues(
                new[] {
                    new DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_I4",
12345),
                    new DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_BOOL",
true),
                    new DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_R4",
234.56)
                });


            for (int i = 0; i < resultArray.Length; i++)
            {
                Debug.Assert(resultArray[i] != null);
                if (resultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
```

```
                {
                    Debug.Assert(resultArray[i].Exception != null);
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray[i].ErrorMessageBrief);
                }
            }

            Console.WriteLine("Reading multiple item values...");
            ValueResult[] valueResultArray = client.ReadMultipleItemValues("OPCLabs.KitServer.2",
                new DAItemDescriptor[] {
                        "Simulation.Register_I4",
                        "Simulation.Register_BOOL",
                        "Simulation.Register_R4" });

            for (int i = 0; i < valueResultArray.Length; i++)
            {
                Debug.Assert(valueResultArray[i] != null);
                Console.WriteLine("valueResultArray[{0}]: {1}", i, valueResultArray[i]);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write values into multiple items.

Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteMultipleItemValues
        Public Shared Sub Main1()
            Dim client = New EasyDAClient()

            Dim argumentsArray = New DAItemValueArguments() { _
                New DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345), _
                New DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_BOOL", True),
_
                New DAItemValueArguments("", "OPCLabs.KitServer.2", "Simulation.Register_R4", 234.56) _
            }

            Dim resultArray As OperationResult() = client.WriteMultipleItemValues(argumentsArray)

            For i = 0 To resultArray.Length - 1
                Debug.Assert(resultArray(i) IsNot Nothing)

                If resultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray(i).ErrorMessageBrief)
                End If
            Next i

            Console.WriteLine("Reading multiple item values...")
            Dim valueResultArray() As ValueResult =
client.ReadMultipleItemValues("OPCLabs.KitServer.2", _
                New DAItemDescriptor() { _
                    "Simulation.Register_I4",
                    "Simulation.Register_BOOL",
                    "Simulation.Register_R4"})

            For i = 0 To valueResultArray.Length - 1
                Debug.Assert(valueResultArray(i) IsNot Nothing)
                Console.WriteLine("valueResultArray[{0}]: {1}", i, valueResultArray(i))
```

```
            Next i

        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to write values into 3 items at once.

class procedure WriteMultipleItemValues.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ItemValueArguments1: _DAItemValueArguments;
  ItemValueArguments2: _DAItemValueArguments;
  ItemValueArguments3: _DAItemValueArguments;
  Results: OleVariant;
  OperationResult: _OperationResult;

begin
  ItemValueArguments1 := CoDAItemValueArguments.Create;
  ItemValueArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemValueArguments1.ItemDescriptor.ItemID := 'Simulation.Register_I4';
  ItemValueArguments1.Value := 23456;

  ItemValueArguments2 := CoDAItemValueArguments.Create;
  ItemValueArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemValueArguments2.ItemDescriptor.ItemID := 'Simulation.Register_R8';
  ItemValueArguments2.Value := 2.34567890;

  ItemValueArguments3 := CoDAItemValueArguments.Create;
  ItemValueArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemValueArguments3.ItemDescriptor.ItemID := 'Simulation.Register_BSTR';
  ItemValueArguments3.Value := 'ABC';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := ItemValueArguments1;
  Arguments[1] := ItemValueArguments2;
  Arguments[2] := ItemValueArguments3;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.WriteMultipleItemValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    OperationResult := IInterface(Results[I]) as _OperationResult;
    if OperationResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ' *** Failure: ', OperationResult.Exception.GetBaseException.Message);
  end;

end;
```

PHP

```
// This example shows how to write values into 3 items at once.

$ItemValueArguments1 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
```

```
$ItemValueArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments1->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemValueArguments1->Value = 23456;

$ItemValueArguments2 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments2->ItemDescriptor->ItemID = "Simulation.Register_R8";
$ItemValueArguments2->Value = 2.34567890;

$ItemValueArguments3 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments3->ItemDescriptor->ItemID = "Simulation.Register_BSTR";
$ItemValueArguments3->Value = "ABC";

$arguments[0] = $ItemValueArguments1;
$arguments[1] = $ItemValueArguments2;
$arguments[2] = $ItemValueArguments3;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$results = $Client->WriteMultipleItemValues($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $OperationResult = $results[$i];
    if ($OperationResult->Succeeded)
        printf("Result %d: success\n", $i);
    else
        printf("Result %d: %s\n", $i, $OperationResult->ErrorMessageBrief);
}
```

PowerScript

```
// Shows how to write into multiple OPC items using a single method call, and itemValue multiple item
values back.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")

// Prepare arguments.

OLEObject itemValueArguments1
itemValueArguments1 = CREATE OLEObject
itemValueArguments1.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")

itemValueArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
itemValueArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
itemValueArguments1.Value = 12345

OLEObject itemValueArguments2
itemValueArguments2 = CREATE OLEObject
itemValueArguments2.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")

itemValueArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
itemValueArguments2.ItemDescriptor.ItemID = "Simulation.Register_BOOL"
itemValueArguments2.Value = TRUE

OLEObject itemValueArguments3
itemValueArguments3 = CREATE OLEObject
itemValueArguments3.ConnectToNewObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")

itemValueArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
itemValueArguments3.ItemDescriptor.ItemID = "Simulation.Register_R4"
itemValueArguments3.Value = 234.56
```

```
OLEObject itemValueArgumentsList
itemValueArgumentsList = CREATE OLEObject
itemValueArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
itemValueArgumentsList.Add(itemValueArguments1)
itemValueArgumentsList.Add(itemValueArguments2)
itemValueArgumentsList.Add(itemValueArguments3)

// Modify value of nodes

OLEObject operationResultList
operationResultList = client.WriteItemValueList(itemValueArgumentsList)

// Display results
Int i
FOR i = 0 TO operationResultList.Count - 1
    OLEObject operationResult
    operationResult = operationResultList.Item[i]
    IF operationResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": success" + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": " +
operationResult.Exception.GetBaseException().Message + "~r~n"
    END IF
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

VBScript

```
Rem This example shows how to write values into 3 items at once.

Option Explicit

Dim ItemValueArguments1: Set ItemValueArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemValueArguments1.Value = 23456

Dim ItemValueArguments2: Set ItemValueArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments2.ItemDescriptor.ItemID = "Simulation.Register_R8"
ItemValueArguments2.Value = 2.34567890

Dim ItemValueArguments3: Set ItemValueArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments3.ItemDescriptor.ItemID = "Simulation.Register_BSTR"
ItemValueArguments3.Value = "ABC"

Dim arguments(2)
Set arguments(0) = ItemValueArguments1
Set arguments(1) = ItemValueArguments2
Set arguments(2) = ItemValueArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.WriteMultipleItemValues(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim OperationResult: Set OperationResult = results(i)
    If OperationResult.Succeeded Then
        WScript.Echo "Result " & i & ": success"
    Else
        WScript.Echo "Result " & i & ": " & OperationResult.Exception.GetBaseException.Message
```

```
    End If
Next
```

VBScript

```
Rem This example shows how to write values into 3 items at once, test for success of each write and
display the exception
Rem message in case of failure.

Option Explicit

Dim ItemValueArguments1: Set ItemValueArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemValueArguments1.Value = 23456

Dim ItemValueArguments2: Set ItemValueArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments2.ItemDescriptor.ItemID = "Simulation.Register_R8"
ItemValueArguments2.Value = "This string cannot be converted to VT_R8"

Dim ItemValueArguments3: Set ItemValueArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments")
ItemValueArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemValueArguments3.ItemDescriptor.ItemID = "UnknownItem"
ItemValueArguments3.Value = "ABC"

Dim arguments(2)
Set arguments(0) = ItemValueArguments1
Set arguments(1) = ItemValueArguments2
Set arguments(2) = ItemValueArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.WriteMultipleItemValues(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim OperationResult: Set OperationResult = results(i)
    If OperationResult.Succeeded Then
        WScript.Echo "Result " & i & ": success"
    Else
        WScript.Echo "Result " & i & ": " & OperationResult.Exception.GetBaseException.Message
    End If
Next
```

PHP

```
// This example shows how to write values into 3 items at once, test for success of each write and
display the exception
// message in case of failure.

$ItemValueArguments1 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments1->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments1->ItemDescriptor->ItemID = "Simulation.Register_I4";
$ItemValueArguments1->Value = 23456;

$ItemValueArguments2 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments2->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments2->ItemDescriptor->ItemID = "Simulation.Register_R8";
$ItemValueArguments2->Value = "This string cannot be converted to VT_R8";

$ItemValueArguments3 = new COM("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemValueArguments");
$ItemValueArguments3->ServerDescriptor->ServerClass = "OPCLabs.KitServer.2";
$ItemValueArguments3->ItemDescriptor->ItemID = "UnknownItem";
$ItemValueArguments3->Value = "ABC";
```

```
$arguments[0] = $ItemValueArguments1;
$arguments[1] = $ItemValueArguments2;
$arguments[2] = $ItemValueArguments3;

$Client = new COM("OpcLabs.EasyOpc.DataAccess.EasyDAClient");
$results = $Client->WriteMultipleItemValues($arguments);

for ($i = 0; $i < count($results); $i++)
{
    $OperationResult = $results[$i];
    if ($OperationResult->Succeeded)
        printf("Result %d: success\n", $i);
    else
        printf("Result %d: %s\n", $i, $OperationResult->ErrorMessageBrief);
}
```

13.2.5.41 Examples - OPC Data Access - Write multiple values and measure time

C#

```
// This example measures the time needed to write 2000 item values all at once, and in
20 groups by 100 items.
// Note that the writes will currently all fail, as we do not have the appropriate
writeable items available.

using System;
using System.Diagnostics;
using System.Threading;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.DataAccess.OperationModel;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteMultipleItemValues
    {
        const int NumberOfGroups = 100;
        const int ItemsInGroup = 20;
        private const int TotalItems = NumberOfGroups * ItemsInGroup;

        // Main method
        public static void TimeMeasurements()
        {
            // Make the measurements 10 times; note that first time the times might be
longer.
            for (int i = 1; i <= 10; i++)
            {
                // Pause - we do not want the component to use the values it has in
memory
                Thread.Sleep(2 * 1000);

                // Write all item values at once, and measure the time
                var stopwatch1 = new Stopwatch();
                stopwatch1.Start();
                WriteAllAtOnce();
```

```
                stopwatch1.Stop();
                Console.WriteLine("WriteAllAtOnce has taken (milliseconds): {0}",
stopwatch1.ElapsedMilliseconds);

                // Pause - we do not want the component to use the values it has in
memory
                Thread.Sleep(2 * 1000);

                // Write item values in groups, and measure the time
                var stopwatch2 = new Stopwatch();
                stopwatch2.Start();
                WriteInGroups();
                stopwatch2.Stop();
                Console.WriteLine("WriteInGroups has taken (milliseconds): {0}",
stopwatch2.ElapsedMilliseconds);
            }
        }

        // Write all item values at once
        private static void WriteAllAtOnce()
        {
            var client = new EasyDAClient();

            // Create an array of arguments for all items
            var arguments = new DAItemValueArguments[TotalItems];
            int index = 0;
            for (int iLoop = 0; iLoop < NumberOfGroups; iLoop++)
                for (int iItem = 0; iItem < ItemsInGroup; iItem++)
                    arguments[index++] = new DAItemValueArguments(
                        "OPCLabs.KitServer.2",
                        String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}",
iLoop + 1, iItem + 1),
                        0);

            // Perform the OPC write
            OperationResult[] operationResults =
client.WriteMultipleItemValues(arguments);

            // Count successful results
            int successCount = 0;
            for (int iItem = 0; iItem < TotalItems; iItem++)
            {
                Debug.Assert(operationResults[iItem] != null);
                if (operationResults[iItem].Succeeded)
                    successCount++;
            }

            if (successCount != TotalItems)
                Console.WriteLine("Warning: There were some failures, success count is
{0}", successCount);
        }

        // Write item values in groups
        private static void WriteInGroups()
        {
            var client = new EasyDAClient();
```

```
            int successCount = 0;
            for (int iLoop = 0; iLoop < NumberOfGroups; iLoop++)
            {
                // Create an array of item arguments for items in one group
                var arguments = new DAItemValueArguments[ItemsInGroup];
                for (int iItem = 0; iItem < ItemsInGroup; iItem++)
                    arguments[iItem] = new DAItemValueArguments(
                        "OPCLabs.KitServer.2",
                        String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}",
iLoop + 1, iItem + 1),
                        0);

                // Perform the OPC write
                OperationResult[] operationResults =
client.WriteMultipleItemValues(arguments);

                // Count successful results (totalling to previous value)
                for (int iItem = 0; iItem < ItemsInGroup; iItem++)
                {
                    Debug.Assert(operationResults[iItem] != null);
                    if (operationResults[iItem].Succeeded) successCount++;
                }
            }

            if (successCount != TotalItems)
                Console.WriteLine("Warning: There were some failures, success count is
{0}", successCount);
        }
    }
}
```

VB.NET

```
' This example measures the time needed to write 2000 item values all at once, and in
20 groups by 100 items.
' Note that the writes will currently all fail, as we do not have the appropriate
writeable items available.

Imports System.Threading
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteMultipleItemValues
        Private Const NumberOfGroups As Integer = 100
        Private Const ItemsInGroup As Integer = 20
        Private Const TotalItems As Integer = NumberOfGroups * ItemsInGroup

        ' Main method
        Public Shared Sub TimeMeasurements()
            ' Make the measurements 10 times; note that first time the times might be
longer.
            For i As Integer = 1 To 10
                ' Pause - we do not want the component to use the values it has in
memory
                Thread.Sleep(2 * 1000)
```

```
                '  Write all item values at once, and measure the time
                Dim stopwatch1 = New Stopwatch()
                stopwatch1.Start()
                WriteAllAtOnce()
                stopwatch1.Stop()
                Console.WriteLine("WriteAllAtOnce has taken (milliseconds): {0}",
stopwatch1.ElapsedMilliseconds)

                ' Pause - we do not want the component to use the values it has in
memory
                Thread.Sleep(2 * 1000)

                ' Write item values in groups, and measure the time
                Dim stopwatch2 = New Stopwatch()
                stopwatch2.Start()
                WriteInGroups()
                stopwatch2.Stop()
                Console.WriteLine("WriteInGroups has taken (milliseconds): {0}",
stopwatch2.ElapsedMilliseconds)
            Next i
        End Sub

        '  Write all item values at once
        Private Shared Sub WriteAllAtOnce()
            Dim client = New EasyDAClient()

            ' Create an array of arguments for all items
            Dim arguments = New DAItemValueArguments(TotalItems - 1) {}
            Dim index As Integer = 0
            For iLoop As Integer = 0 To NumberOfGroups - 1
                For iItem As Integer = 0 To ItemsInGroup - 1
                    arguments(index) = New DAItemValueArguments(
                        "OPCLabs.KitServer.2",
                            String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}",
iLoop + 1, iItem + 1),
                            0)
                    index += 1
                Next iItem
            Next iLoop

            ' Perform the OPC write
            Dim operationResults() As OperationResult =
client.WriteMultipleItemValues(arguments)

            ' Count successful results
            Dim successCount As Integer = 0
            For iItem As Integer = 0 To TotalItems - 1
                Debug.Assert(operationResults(iItem) IsNot Nothing)
                If operationResults(iItem).Succeeded Then
                    successCount += 1
                End If
            Next iItem

            If successCount <> TotalItems Then
                Console.WriteLine("Warning: There were some failures, success count is
{0}", successCount)
```

```
            End If
        End Sub

        ' Write item values in groups
        Private Shared Sub WriteInGroups()
            Dim client = New EasyDAClient()

            Dim successCount As Integer = 0
            For iLoop As Integer = 0 To NumberOfGroups - 1
                ' Create an array of item arguments for items in one group
                Dim arguments = New DAItemValueArguments(ItemsInGroup - 1) {}
                For iItem As Integer = 0 To ItemsInGroup - 1
                    arguments(iItem) = New DAItemValueArguments(
                            "OPCLabs.KitServer.2",
                            String.Format("Simulation.Incrementing.Copy_{0}.Phase_{1}",
iLoop + 1, iItem + 1),
                            0)
                Next iItem

                ' Perform the OPC write
                Dim operationResults() As OperationResult =
client.WriteMultipleItemValues(arguments)

                ' Count successful results (totalling to previous value)
                For iItem As Integer = 0 To ItemsInGroup - 1
                    Debug.Assert(operationResults(iItem) IsNot Nothing)
                    If operationResults(iItem).Succeeded Then
                        successCount += 1
                    End If
                Next iItem
            Next iLoop

            If successCount <> TotalItems Then
                Console.WriteLine("Warning: There were some failures, success count is
{0}", successCount)
            End If
        End Sub
    End Class
End Namespace
```

13.2.5.42 Examples - OPC Data Access - Write multiple values, specify requested data types

C#

```
// Shows how to write into multiple OPC items using a single method call, specifying
their requested data types.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.BaseLib.OperationModel;
```

```
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess._EasyDAClient
{
    partial class WriteMultipleItemValues
    {
        public static void RequestedDataType()
        {
            var client = new EasyDAClient();

            Console.WriteLine("Writing multiple item values...");
            OperationResult[] resultArray = client.WriteMultipleItemValues(new[] {
                    new DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I2", 12345)
                        { ItemDescriptor = { RequestedDataType = VarTypes.I2}}, // <--
the requested data type
                    new DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_R4", 234.56)
                        { ItemDescriptor = { RequestedDataType = VarTypes.R4}}  // <--
the requested data type
                });


            for (int i = 0; i < resultArray.Length; i++)
            {
                Debug.Assert(resultArray[i] != null);
                if (resultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                {
                    Debug.Assert(resultArray[i].Exception != null);
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray[i].ErrorMessageBrief);
                }
            }

            Console.WriteLine("Reading multiple item values...");
            ValueResult[] valueResultArray =
client.ReadMultipleItemValues("OPCLabs.KitServer.2",
                new DAItemDescriptor[] { "Simulation.Register_I2",
"Simulation.Register_R4" });

            for (int i = 0; i < valueResultArray.Length; i++)
            {
                Debug.Assert(valueResultArray[i] != null);
                Console.WriteLine("valueResultArray[{0}]: {1}", i,
valueResultArray[i]);
            }
        }
    }
}
```

VB.NET

```
' Shows how to write into multiple OPC items using a single method call, specifying
their requested data types.
```

```
Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess._EasyDAClient
    Partial Friend Class WriteMultipleItemValues
        Public Shared Sub RequestedDataType()
            Dim client = New EasyDAClient()

            Console.WriteLine("Writing multiple item values...")
            Dim resultArray = client.WriteMultipleItemValues(New DAItemValueArguments()
{ _
                New DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_I2", 12345) With _
                {.ItemDescriptor = New DAItemDescriptor() With {.RequestedDataType =
VarTypes.I2}}, _
                New DAItemValueArguments("", "OPCLabs.KitServer.2",
"Simulation.Register_R4", 234.56) With _
                {.ItemDescriptor = New DAItemDescriptor() With {.RequestedDataType =
VarTypes.R4}} _
            })

            For i = 0 To resultArray.Length - 1
                Debug.Assert(resultArray(i) IsNot Nothing)
                If resultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Debug.Assert(resultArray(i).Exception IsNot Nothing)
                    Console.WriteLine("Result {0} *** Failure: {1}", i,
resultArray(i).ErrorMessageBrief)
                End If
            Next i

            Console.WriteLine("Reading multiple item values...")
            Dim valueResultArray = client.ReadMultipleItemValues("OPCLabs.KitServer.2",
                    New DAItemDescriptor() {"Simulation.Register_I2",
"Simulation.Register_R4"})

            For i = 0 To valueResultArray.Length - 1
                Debug.Assert(valueResultArray(i) IsNot Nothing)
                Console.WriteLine("valueResultArray[{0}]: {1}", i, valueResultArray(i))
            Next i
        End Sub
    End Class
End Namespace
```

13.2.5.43 Examples - OPC Data Access - Write multiple values, timestamps and qualities

Object Pascal

```
// This example shows how to write values, timestamps and qualities into 3 items at
once.

class procedure WriteMultipleItems.Main;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
  I: Cardinal;
  ItemVtqArguments1: _DAItemVtqArguments;
  ItemVtqArguments2: _DAItemVtqArguments;
  ItemVtqArguments3: _DAItemVtqArguments;
  Results: OleVariant;
  OperationResult: _OperationResult;

begin
  ItemVtqArguments1 := CoDAItemVtqArguments.Create;
  ItemVtqArguments1.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemVtqArguments1.ItemDescriptor.ItemID := 'Simulation.Register_I4';
  ItemVtqArguments1.Vtq.Value := 23456;
  ItemVtqArguments1.Vtq.TimestampLocal := Now();
  ItemVtqArguments1.Vtq.Quality.NumericalValue := DAQualities_GoodNonSpecific;

  ItemVtqArguments2 := CoDAItemVtqArguments.Create;
  ItemVtqArguments2.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemVtqArguments2.ItemDescriptor.ItemID := 'Simulation.Register_R8';
  ItemVtqArguments2.Vtq.Value := 2.34567890;
  ItemVtqArguments2.Vtq.TimestampLocal := Now();
  ItemVtqArguments2.Vtq.Quality.NumericalValue := DAQualities_GoodNonSpecific;

  ItemVtqArguments3 := CoDAItemVtqArguments.Create;
  ItemVtqArguments3.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';
  ItemVtqArguments3.ItemDescriptor.ItemID := 'Simulation.Register_BSTR';
  ItemVtqArguments3.Vtq.Value := 'ABC';
  ItemVtqArguments3.Vtq.TimestampLocal := Now();
  ItemVtqArguments3.Vtq.Quality.NumericalValue := DAQualities_GoodNonSpecific;

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := ItemVtqArguments1;
  Arguments[1] := ItemVtqArguments2;
  Arguments[2] := ItemVtqArguments3;

  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(
    Client.WriteMultipleItems(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    OperationResult := IInterface(Results[I]) as _OperationResult;
    if OperationResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ' *** Failure: ',
```

```
OperationResult.Exception.GetBaseException.Message);
  end;

end;
```

VBScript

```
Rem This example shows how to write values, timestamps and qualities into 3 items at
once.

Option Explicit

Dim ItemVtqArguments1: Set ItemVtqArguments1 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemVtqArguments")
ItemVtqArguments1.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemVtqArguments1.ItemDescriptor.ItemID = "Simulation.Register_I4"
ItemVtqArguments1.Vtq.Value = 23456
ItemVtqArguments1.Vtq.TimestampLocal = Now()
ItemVtqArguments1.Vtq.Quality.NumericalValue = 192

Dim ItemVtqArguments2: Set ItemVtqArguments2 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemVtqArguments")
ItemVtqArguments2.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemVtqArguments2.ItemDescriptor.ItemID = "Simulation.Register_R8"
ItemVtqArguments2.Vtq.Value = 2.34567890
ItemVtqArguments2.Vtq.TimestampLocal = Now()
ItemVtqArguments2.Vtq.Quality.NumericalValue = 192

Dim ItemVtqArguments3: Set ItemVtqArguments3 =
CreateObject("OpcLabs.EasyOpc.DataAccess.OperationModel.DAItemVtqArguments")
ItemVtqArguments3.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
ItemVtqArguments3.ItemDescriptor.ItemID = "Simulation.Register_BSTR"
ItemVtqArguments3.Vtq.Value = "ABC"
ItemVtqArguments3.Vtq.TimestampLocal = Now()
ItemVtqArguments3.Vtq.Quality.NumericalValue = 192

Dim arguments(2)
Set arguments(0) = ItemVtqArguments1
Set arguments(1) = ItemVtqArguments2
Set arguments(2) = ItemVtqArguments3

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
Dim results: results = Client.WriteMultipleItems(arguments)

Dim i: For i = LBound(results) To UBound(results)
    Dim OperationResult: Set OperationResult = results(i)
    If OperationResult.Succeeded Then
        WScript.Echo "Result " & i & ": success"
    Else
        WScript.Echo "Result " & i & ": " &
OperationResult.Exception.GetBaseException.Message
    End If
Next
```

13.2.5.44 Examples - OPC Data Access - Write single value, timestamp and quality

Object Pascal

```
// This example shows how to write a value, timestamp and quality into a single item.

class procedure WriteItem.Main;
var
  Client: OpcLabs_EasyOpcClassic_TLB._EasyDAClient;
begin
  // Instantiate the client object
  Client := CoEasyDAClient.Create;

  try
    Client.WriteItem('', 'OPCLabs.KitServer.2', 'Simulation.Register_I4', 12345,
EncodeDate(1980, 1, 1), DAQualities_GoodNonSpecific);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

VBScript

```
Rem This example shows how to write a value, timestamp and quality into a single item.

Option Explicit

Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.DataAccess.EasyDAClient")
On Error Resume Next
Client.WriteItem "", "OPCLabs.KitServer.2", "Simulation.Register_I4", 12345,
DateSerial(1980, 1, 1), 192
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

13.2.6 Examples - OPC UA Alarms&Conditions

13.2.6.1 Examples - OPC UA Alarms&Conditions - Acknowledge an event

C#

```
// This example shows how to acknowledge an OPC UA event.
```

```
using System;
using System.Threading;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class Acknowledge
    {
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client objects
            var client = new EasyUAClient();
            IEasyUAAlarmsAndConditionsClient alarmsAndConditionsClient =
client.AsAlarmsAndConditionsClient();

            UANodeId nodeId = null;
            byte[] eventId = null;
            var anEvent = new ManualResetEvent(initialState: false);

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                new UAEventFilterBuilder(
                    UAFilterElements.Equals(
                    UABaseEventObject.Operands.NodeId,
                        new UANodeId(expandedText:
"nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=1:Colours/EastTank?Yellow")),
                    UABaseEventObject.AllFields),
                (sender, eventArgs) =>
                {
                    if (!eventArgs.Succeeded)
                    {
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief);
                        return;
                    }
                    if (eventArgs.EventData != null)
                    {
                        UABaseEventObject baseEventObject = eventArgs.EventData.BaseEvent;
                        Console.WriteLine(baseEventObject);

                        // Make sure we do not catch the event more than once
                        if (anEvent.WaitOne(0))
                            return;

                        nodeId = baseEventObject.NodeId;
                        eventId = baseEventObject.EventId;

                        anEvent.Set();
                    }
                },
                state:null);

            Console.WriteLine("Waiting for an event for 30 seconds...");
            if (!anEvent.WaitOne(30*1000))
            {
                Console.WriteLine("Event not received.");
```

```
                return;
            }

            Console.WriteLine("Acknowledging an event...");
            try
            {
                alarmsAndConditionsClient.Acknowledge(
                    endpointDescriptor,
                    nodeId,
                    eventId,
                    "Acknowledged by an automated example code.");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
            }

            Console.WriteLine("Waiting for 5 seconds...");
            Thread.Sleep(5 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            Thread.Sleep(5 * 1000);
        }



        // Example output:
        //Subscribing...
        //Waiting for an event for 30 seconds...
        //[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
        //Acknowledging an event...
        //Waiting for 5 seconds...
        //[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
        //Unsubscribing...
        //Waiting for 5 seconds...
    }
}
```

Object Pascal

```
// This example shows how to acknowledge an OPC UA event.

type
  THelperMethods11 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_AllFields: _UAAttributeFieldCollection; static;
  end;

type
  TClientEventHandlers11 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
```

```
      const eventArgs: _EasyUAEventNotificationEventArgs);
    private
      FAnEvent: Boolean;
      FEventId: OleVariant;
      FNodeId: _UANodeId;
    published
      property AnEvent: Boolean read FAnEvent write FAnEvent;
      property EventId: OleVariant read FEventId;
      property NodeId: _UANodeId read FNodeId;
  end;

procedure TClientEventHandlers11.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  BaseEventObject: _UABaseEventObject;
begin
  if not eventArgs.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
    Exit;
  end;

  if eventArgs.EventData <> nil then
  begin
    BaseEventObject := eventArgs.EventData.BaseEvent;
      WriteLn(BaseEventObject.ToString);

    // Make sure we do not catch the event more than once
    if FAnEvent then
      Exit;

    FNodeId := BaseEventObject.NodeId;
    TVarData(FEventId).VType := varArray or varVariant;
    TVarData(FEventId).VArray := PVarArray(BaseEventObject.EventId);
    FAnEvent := True;
  end;
end;

class procedure Acknowledge.Main;
var
  AlarmsAndConditionsClient: _EasyUAAlarmsAndConditionsClient;
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers11;
  EndpointDescriptor: string;
  EndTime: Cardinal;
  EventFilter: _UAEventFilter;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  Operand1: _UASimpleAttributeOperand;
  Operand2: _UALiteralOperand;
  NodeDescriptor: _UANodeDescriptor;
  NodeId: _UANodeId;
  ServerNodeId: _UANodeID;
  WhereClause: _UAContentFilterElement;
begin
  // Define which server we will work with.
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Event filter: Events with specific node ID.
  Operand1 := THelperMethods11.UABaseEventObject_Operands_NodeId;
  NodeId := CoUANodeId.Create;
  NodeId.ExpandedText := 'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Yellow';
  Operand2 := CoUALiteralOperand.Create;
  Operand2.Value := NodeId;
```

```
WhereClause := CoUAContentFilterElement.Create;
WhereClause.FilterOperator := UAFilterOperator_Equals;
WhereClause.FilterOperands.Add(Operand1);
WhereClause.FilterOperands.Add(Operand2);

EventFilter := CoUAEventFilter.Create;
EventFilter.SelectClauses := THelperMethods11.UABaseEventObject_AllFields;
EventFilter.WhereClause := WhereClause;

ServerNodeId := CoUANodeId.Create;
ServerNodeId.StandardName := 'Server';

MonitoringParameters := CoUAMonitoringParameters.Create;
MonitoringParameters.EventFilter := EventFilter;
MonitoringParameters.QueueSize := 1000;
MonitoringParameters.SamplingInterval := 1000;

MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;
MonitoredItemArguments.EndpointDescriptor.UrlString := EndpointDescriptor;
MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
MonitoredItemArguments.NodeDescriptor.NodeId := ServerNodeId;

// Instantiate the client object and hook events
Client := TEasyUAClient.Create(nil);
ClientEventHandlers := TClientEventHandlers11.Create;
Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;
AlarmsAndConditionsClient := Client.AsAlarmsAndConditionsClient;
ClientEventHandlers.AnEvent := False;

Arguments := VarArrayCreate([0, 0], varVariant);
Arguments[0] := MonitoredItemArguments;

WriteLn('Subscribing...');
Client.SubscribeMultipleMonitoredItems(Arguments);

WriteLn('Waiting for an event for 30 seconds...');
EndTime := Ticks + 30*1000;
repeat
    if ClientEventHandlers.AnEvent or (EndTime < Ticks) then
      Break;
    PumpSleep(1000);
until False;

if ClientEventHandlers.AnEvent then
begin
  WriteLn('Acknowledging an event...');
  try
    NodeDescriptor := CoUANodeDescriptor.Create;
    NodeDescriptor.NodeId := ClientEventHandlers.NodeId;
    AlarmsAndConditionsClient.Acknowledge(
     MonitoredItemArguments.EndpointDescriptor,
     NodeDescriptor,
     ClientEventHandlers.EventId,
     'Acknowledged by an automated example code.');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
end
else
begin
  WriteLn('Event not received.');
end;
```

```
  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;

// Example output:
// Subscribing...
// Waiting for an event for 30 seconds...
// [EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
// Acknowledging an event...
// Waiting for 5 seconds...
// [EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
// Unsubscribing...
// Waiting for 5 seconds...


class function THelperMethods11.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods11.UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames :=
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods11.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods11.UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
end;

class function THelperMethods11.UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
end;

class function THelperMethods11.UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand;
```

```
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods11.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods11.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;

class function THelperMethods11.UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
end;

class function THelperMethods11.UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
end;

class function THelperMethods11.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods11.UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;

class function THelperMethods11.UABaseEventObject_AllFields: _UAAttributeFieldCollection;
var
  Fields: _UAAttributeFieldCollection;
begin
  Fields := CoUAAttributeFieldCollection.Create;
  Fields.Add(UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventType.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Time.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_LocalTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Message.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Severity.ToUAAttributeField);

  Result := Fields;
end;
```

PHP

```
// This example shows how to acknowledge an OPC UA event.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

const UAFilterOperator_Equals = 1;

class ClientEvents {
    private $AnEvent = false;
    private $EventId;
    private $NodeId;

    function EventNotification($Sender, $E)
```

```
    {
        if (!$E->Succeeded) {
            printf(" *** Failure: %s\n", $E->ErrorMessageBrief);
            return;
        }

        if (!is_null($E->EventData)) {
            $BaseEventObject = $E->EventData->BaseEvent;
            printf("%s\n", $BaseEventObject);

            // Make sure we do not catch the event more than once
           if ($this->AnEvent)
               return;

           $this->NodeId = $BaseEventObject->NodeId;
           $this->EventId = $BaseEventObject->EventId;
           $this->AnEvent = true;
        }
    }

    public function getAnEvent() {
        return $this->AnEvent;
    }

    public function getEventId() {
        return $this->EventId;
    }

    public function getNodeId() {
        return $this->NodeId;
    }
}


// Define which server we will work with.
$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Event filter: Events with specific node ID.
$Operand1 = UABaseEventObject_Operands_NodeId();
$NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId->ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Yellow";
$Operand2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand2->Value = $NodeId;
$WhereClause = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$WhereClause->FilterOperator = UAFilterOperator_Equals;
$WhereClause->FilterOperands->Add($Operand1);
$WhereClause->FilterOperands->Add($Operand2);

$EventFilter = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter->SelectClauses = UABaseEventObject_AllFields();
$EventFilter->WhereClause = $WhereClause;

$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";

$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->EventFilter = $EventFilter;
$MonitoringParameters->QueueSize = 1000;
$MonitoringParameters->SamplingInterval = 1000;

$MonitoredItemArguments = new COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments->AttributeId = UAAttributeId_EventNotifier;
$MonitoredItemArguments->EndpointDescriptor = $EndpointDescriptor;
$MonitoredItemArguments->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments->NodeDescriptor->NodeId = $ServerNodeId;
```

```
// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");
$AlarmsAndConditionsClient = $Client->AsAlarmsAndConditionsClient;

$arguments[0] = $MonitoredItemArguments;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Waiting for an event for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while ((time() < $startTime + 30) and !
($ClientEvents->getAnEvent()));

if ($ClientEvents->getAnEvent()) {
    printf("Acknowledging an event...\n");
    try
    {
        $NodeDescriptor = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
        $NodeDescriptor->NodeId = $ClientEvents->getNodeId();
        $AlarmsAndConditionsClient->Acknowledge(
           $EndpointDescriptor,
           $NodeDescriptor,
           $ClientEvents->getEventId(),
           "Acknowledged by an automated example code.");
    }
    catch (com_exception $e)
    {
          printf("Failure: %s\n", $e->getMessage());
    }
}
else {
    printf("Event not received.\n");
}

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}
```

```
function UABaseEventObject_Operands_EventId() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventId");
}

function UABaseEventObject_Operands_EventType() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventType");
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}

function UABaseEventObject_Operands_ReceiveTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/ReceiveTime");
}

function UABaseEventObject_Operands_LocalTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/LocalTime");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}


function UABaseEventObject_AllFields() {
    $Fields = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
    $Fields->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventType()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_ReceiveTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_LocalTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField);

    return $Fields;
}
```

VB.NET

```
' This example shows how to obtain acknowledge an event.

Imports System
Imports System.Threading
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions.Extensions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
```

```
    Friend Class Acknowledge
        Public Shared Sub Main1()

            ' Instantiate the client object
            Dim client = New EasyUAClient()
            Dim alarmsAndConditionsClient As IEasyUAAlarmsAndConditionsClient =
client.AsAlarmsAndConditionsClient()

            Dim nodeId As UANodeId = Nothing
            Dim eventId As Byte() = Nothing
            Dim anEvent = New ManualResetEvent(initialState:=False)

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent(
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
                UAObjectIds.Server,
                1000,
                New UAEventFilterBuilder(
                    UAFilterElements.Equals(
                        UABaseEventObject.Operands.NodeId,
                        New
UANodeId(expandedText:="nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Colours/EastTank?
Yellow")),
                        UABaseEventObject.AllFields),
                Sub(sender, eventArgs)
                    If Not eventArgs.Succeeded Then
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief)
                        Return
                    End If
                    If eventArgs.EventData IsNot Nothing Then
                        Dim baseEventObject = eventArgs.EventData.BaseEvent
                        Console.WriteLine(baseEventObject)

                        ' Make sure we do not catch the event more than once
                        If anEvent.WaitOne(0) Then
                            Return
                        End If

                        nodeId = baseEventObject.NodeId
                        eventId = baseEventObject.EventId

                        anEvent.Set()
                    End If
                End Sub,
                state:=Nothing)

            Console.WriteLine("Waiting for an event for 30 seconds...")
            If Not anEvent.WaitOne(30 * 1000) Then
                Console.WriteLine("Event not received")
                Return
            End If

            Console.WriteLine("Acknowledging an event...")
            Try
                alarmsAndConditionsClient.Acknowledge(
                    "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
                    nodeId,
                    eventId,
                    "Acknowledged by an automated example code")
            Catch uaException As UAException
                Console.WriteLine("Failure: {0}", uaException.GetBaseException().Message)
            End Try

            Console.WriteLine("Waiting for 5 seconds...")
            Thread.Sleep(5 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()
```

```
            Console.WriteLine("Waiting for 5 seconds...")
            Thread.Sleep(5 * 1000)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to acknowledge an OPC UA event.

Option Explicit

Const UAAttributeId_NodeId = 1
Const UAAttributeId_EventNotifier = 12

Const UAFilterOperator_Equals = 1

' Define which server we will work with.
Dim EndpointDescriptor: Set EndpointDescriptor = CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
EndpointDescriptor.UrlString = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer"

' Instantiate the client objects and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"
Dim AlarmsAndConditionsClient: Set AlarmsAndConditionsClient = Client.AsAlarmsAndConditionsClient

'
Dim NodeId
Dim EventId
Dim anEvent: anEvent = False ' Some tools have event objects, but VBScript doesn't, we will use a boolean
flag instead.

' Prepare arguments
Dim arguments(0)
Set arguments(0) = CreateMonitoredItemArguments

WScript.Echo "Subscribing..."
Client.SubscribeMultipleMonitoredItems arguments

WScript.Echo "Waiting for an event for 30 seconds..."
Dim endTime: endTime = Now() + 30*(1/24/60/60)
While (Not anEvent) And (Now() < endTime)
    WScript.Sleep 1000
WEnd
If Not anEvent Then
    WScript.Echo "Event not received."
    WScript.Quit
End If

WScript.Echo "Acknowledging an event..."
Dim NodeDescriptor: Set NodeDescriptor = CreateObject("OpcLabs.EasyOpc.UA.UANodeDescriptor")
Set NodeDescriptor.NodeId = NodeId
On Error Resume Next
AlarmsAndConditionsClient.Acknowledge EndpointDescriptor, NodeDescriptor, EventId, "Acknowledged by an
automated example code."
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
End If
On Error Goto 0

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
```

```
WScript.Sleep 5 * 1000


Function ObjectTypeIds_BaseEventType
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.StandardName = "BaseEventType"
    Set ObjectTypeIds_BaseEventType = NodeId
End Function

Function UAFilterElements_SimpleAttribute(TypeId, simpleRelativeBrowsePathString)
    Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
    Dim Operand: Set Operand = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Set Operand.TypeId.NodeId = TypeId
    Set Operand.QualifiedNames =
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection
    Set UAFilterElements_SimpleAttribute = Operand
End Function

Function UABaseEventObject_Operands_NodeId
    Dim Operand: Set Operand = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Operand.TypeId.NodeId.StandardName = "BaseEventType"
    Operand.AttributeId = UAAttributeId_NodeId
    Set UABaseEventObject_Operands_NodeId = Operand
End Function

Function UABaseEventObject_Operands_EventId
    Set UABaseEventObject_Operands_EventId =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventId")
End Function

Function UABaseEventObject_Operands_EventType
    Set UABaseEventObject_Operands_EventType =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventType")
End Function

Function UABaseEventObject_Operands_SourceNode
    Set UABaseEventObject_Operands_SourceNode =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceNode")
End Function

Function UABaseEventObject_Operands_SourceName
    Set UABaseEventObject_Operands_SourceName =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceName")
End Function

Function UABaseEventObject_Operands_Time
    Set UABaseEventObject_Operands_Time = UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
"/Time")
End Function

Function UABaseEventObject_Operands_ReceiveTime
    Set UABaseEventObject_Operands_ReceiveTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/ReceiveTime")
End Function

Function UABaseEventObject_Operands_LocalTime
    Set UABaseEventObject_Operands_LocalTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/LocalTime")
End Function

Function UABaseEventObject_Operands_Message
    Set UABaseEventObject_Operands_Message =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Message")
End Function

Function UABaseEventObject_Operands_Severity
```

```
    Set UABaseEventObject_Operands_Severity =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Severity")
End Function

Function UABaseEventObject_AllFields
    Dim Fields: Set Fields = CreateObject("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection")

    Fields.Add UABaseEventObject_Operands_NodeId.ToUAAttributeField

    Fields.Add UABaseEventObject_Operands_EventId.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_EventType.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceNode.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceName.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Time.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_LocalTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Message.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Severity.ToUAAttributeField

    Set UABaseEventObject_AllFields = Fields
End Function

Function CreateMonitoredItemArguments
    ' Event filter: Events with specific node ID.
    Dim Operand1: Set Operand1 = UABaseEventObject_Operands_NodeId
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Yellow"
    Dim Operand2: Set Operand2 = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand")
    Set Operand2.Value = NodeId
    Dim WhereClause: Set WhereClause =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement")
    WhereClause.FilterOperator = UAFilterOperator_Equals
    WhereClause.FilterOperands.Add Operand1
    WhereClause.FilterOperands.Add Operand2

    Dim EventFilter: Set EventFilter = CreateObject("OpcLabs.EasyOpc.UA.UAEventFilter")
    Set EventFilter.SelectClauses = UABaseEventObject_AllFields
    Set EventFilter.WhereClause = WhereClause

    Dim ServerNodeId: Set ServerNodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    ServerNodeId.StandardName = "Server"

    Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
    Set MonitoringParameters.EventFilter = EventFilter
    MonitoringParameters.QueueSize = 1000
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments: Set MonitoredItemArguments =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
    MonitoredItemArguments.AttributeId = UAAttributeId_EventNotifier
    MonitoredItemArguments.EndpointDescriptor = EndpointDescriptor
    Set MonitoredItemArguments.MonitoringParameters = MonitoringParameters
    Set MonitoredItemArguments.NodeDescriptor.NodeId = ServerNodeId

    Set CreateMonitoredItemArguments = MonitoredItemArguments
End Function

Sub Client_EventNotification(Sender, EventArgs)
    If Not EventArgs.Succeeded Then
        WScript.Echo "*** Failure: " & EventArgs.ErrorMessageBrief
        Exit Sub
    End If

    If Not (EventArgs.EventData Is Nothing) Then
        Dim BaseEventObject: Set BaseEventObject = EventArgs.EventData.BaseEvent
        WScript.Echo BaseEventObject
```

```
        ' Make sure we do not catch the event more than once
        If anEvent Then
            Exit Sub
        End If

        Set NodeId = BaseEventObject.NodeId
        EventId = BaseEventObject.EventId

        anEvent = True
    End If
End Sub


' Example output:
'Subscribing...
'Waiting for an event for 30 seconds...
'[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
'Acknowledging an event...
'Waiting for 5 seconds...
'[EastTank] 100! "The alarm was acknoweledged." @11/9/2019 9:56:23 AM
'Unsubscribing...
'Waiting for 5 seconds...
```

13.2.6.2 Examples - OPC UA Alarms&Conditions - Browsing for event sources

C#

```
// This example shows how to browse objects under the "Objects" node and display event
sources.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class BrowseEventSources
    {
        public static void Overload2()
        {
            // Start browsing from the "Objects" node
            try
            {
                BrowseFrom(UAObjectIds.ObjectsFolder);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
```

```
        }

        private static void BrowseFrom(UANodeDescriptor nodeDescriptor)
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine("Parent node: {0}", nodeDescriptor);

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain event sources
            UANodeElementCollection eventSourceNodeElementCollection =
client.BrowseEventSources(
                endpointDescriptor, nodeDescriptor);

            // Display event sources
            if (eventSourceNodeElementCollection.Count != 0)
            {
                Console.WriteLine();
                Console.WriteLine("Event sources:");
                foreach (UANodeElement eventSourceNodeElement in
eventSourceNodeElementCollection)
                    Console.WriteLine(eventSourceNodeElement);
            }

            // Obtain objects
            UANodeElementCollection objectNodeElementCollection = client.BrowseObjects(
                endpointDescriptor, nodeDescriptor);

            // Recurse
            foreach (UANodeElement objectNodeElement in objectNodeElementCollection)
                BrowseFrom(objectNodeElement);
        }

        // Example output (truncated):
        //
        //
        //Parent node: ObjectsFolder
        //
        //
        //Parent node: Server
        //
        //Event sources:
        //Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Green (Object)
        //Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Yellow (Object)
        //
        //
        //Parent node: Server_ServerCapabilities
        //...
    }
}
```

Object Pascal

```
// This example shows how to browse objects under the "Objects" node and display event
sources.

class procedure BrowseEventSources.Main;
var
  ObjectNodeId: _UANodeId;
begin
  ObjectNodeId := CoUANodeId.Create;
  ObjectNodeId.StandardName := 'Objects';
  try
    BrowseFrom(ObjectNodeId);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
end;

class procedure BrowseEventSources.BrowseFrom(NodeId: _UANodeId);
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  EventSourceNodeElement: _UANodeElement;
  EventSourceNodeElementEnumerator: IEnumVariant;
  EventSourceNodeElements: _UANodeElementCollection;
  ObjectNodeElement: _UANodeElement;
  ObjectNodeElementEnumerator: IEnumVariant;
  ObjectNodeElements: _UANodeElementCollection;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  WriteLn;
  WriteLn;
  WriteLn('Parent node: ', NodeId.ToString);

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain event sources
  EventSourceNodeElements := Client.BrowseEventSources(EndpointDescriptor,
NodeId.ExpandedText);

  // Display event sources
  if EventSourceNodeElements.Count <> 0 then
  begin
    WriteLn;
    WriteLn('Event sources:');
    EventSourceNodeElementEnumerator := EventSourceNodeElements.GetEnumerator;
    while (EventSourceNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
```

```
    begin
      EventSourceNodeElement := IUnknown(Element) as _UANodeElement;
      WriteLn(EventSourceNodeElement.ToString);
    end;
  end;

  // Obtain objects
  ObjectNodeElements := Client.BrowseObjects(EndpointDescriptor, NodeId.ExpandedText);

  // Recurse
  ObjectNodeElementEnumerator := ObjectNodeElements.GetEnumerator;
  while (ObjectNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    ObjectNodeElement := IUnknown(Element) as _UANodeElement;
    BrowseFrom(ObjectNodeElement.NodeId);
  end;
end;
```

PHP

```
// This example shows how to browse objects under the "Objects" node and display event
sources.


// Start browsing from the "Objects" node
$ObjectNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ObjectNodeId->StandardName = "Objects";
try
{
    BrowseFrom($ObjectNodeId);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

function BrowseFrom($NodeId) {
    $EndpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer";

    printf("\n");
    printf("\n");
    printf("Parent node: %s\n", $NodeId);

    // Instantiate the client object
    $Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

    // Obtain event sources
    $EventSourceNodeElements = $Client->BrowseEventSources($EndpointDescriptor,
$NodeId->ExpandedText);

    // Display event sources
    if ($EventSourceNodeElements->Count != 0) {
        printf("\n");
        printf("Event sources:\n");
        foreach ($EventSourceNodeElements as $EventSourceNodeElement)
```

```
        {
            printf("%s\n", $EventSourceNodeElement);
        }
     }

    // Obtain objects
    $ObjectNodeElements = $Client->BrowseObjects($EndpointDescriptor, $NodeId-
>ExpandedText);

    // Recurse
    foreach ($ObjectNodeElements as $ObjectNodeElement)
    {
        BrowseFrom($ObjectNodeElement->NodeId);
    }
}

// Example output (truncated):
//
//
//Parent node: ObjectsFolder
//
//
//Parent node: Server
//
//Event sources:
//Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
//Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
//
//
//Parent node: Server_ServerCapabilities
//...
```

VB.NET

```
' This example shows how to browse objects under the "Objects" node and display event
sources.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class BrowseEventSources
        Public Shared Sub Overload2()

            ' Start browsing from the "Objects" node
            Try
                BrowseFrom(UAObjectIds.ObjectsFolder)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try
        End Sub
```

```
        Private Shared Sub BrowseFrom(nodeDescriptor As UANodeDescriptor)

            Console.WriteLine()
            Console.WriteLine()
            Console.WriteLine("Parent node: {0}", nodeDescriptor)

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain notifiers
            Dim eventSourceNodeElementCollection As UANodeElementCollection =
client.BrowseEventSources( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Display notifiers
            Console.WriteLine()
            Console.WriteLine("Event sources:")
            For Each eventSourceNodeElement As UANodeElement In
eventSourceNodeElementCollection
                Console.WriteLine(eventSourceNodeElement)
            Next eventSourceNodeElement

            ' Obtain objects
            Dim objectNodeElementCollection = client.BrowseObjects( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Recurse
            For Each objectNodeElement As UANodeElement In objectNodeElementCollection
                BrowseFrom(objectNodeElement)
            Next objectNodeElement
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to browse objects under the "Objects" node and display event
sources.

Option Explicit

' Start browsing from the "Objects" node
Dim ObjectsNodeId: Set ObjectsNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ObjectsNodeId.StandardName = "Objects"
On Error Resume Next
BrowseFrom ObjectsNodeId
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

```
Sub BrowseFrom(NodeId)
    Dim endpointDescriptor
    endpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"

    WScript.Echo
    WScript.Echo
    WScript.Echo "Parent node: " & NodeId

    ' Instantiate the client object
    Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

    ' Obtain event sources
    Dim EventSourceNodeElementCollection: Set EventSourceNodeElementCollection =
Client.BrowseEventSources( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Display event sources
    If EventSourceNodeElementCollection.Count <> 0 Then
        WScript.Echo
        WScript.Echo "Event sources:"
        Dim EventSourceNodeElement: For Each EventSourceNodeElement In
EventSourceNodeElementCollection
            WScript.Echo EventSourceNodeElement
        Next
    End If

    ' Obtain objects
    Dim ObjectNodeElementCollection: Set ObjectNodeElementCollection =
Client.BrowseObjects( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Recurse
    Dim ObjectNodeElement: For Each ObjectNodeElement In ObjectNodeElementCollection
        BrowseFrom ObjectNodeElement.NodeId
    Next
End Sub



' Example output (truncated):
'
'
'Parent node: ObjectsFolder
'
'
'Parent node: Server
'
'Event sources:
'Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
'Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
'
```

```
'
'Parent node: Server_ServerCapabilities
'...
```

13.2.6.3 Examples - OPC UA Alarms&Conditions - Browsing for notifiers

C#

```
// This example shows how to browse objects under the "Objects" node and display
notifiers.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class BrowseNotifiers
    {
        public static void Overload2()
        {
            // Start browsing from the "Objects" node
            try
            {
                BrowseFrom(UAObjectIds.ObjectsFolder);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }

        private static void BrowseFrom(UANodeDescriptor nodeDescriptor)
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine("Parent node: {0}", nodeDescriptor);

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain notifiers
            UANodeElementCollection notifierNodeElementCollection =
client.BrowseNotifiers(
                endpointDescriptor, nodeDescriptor);
```

```
            // Display notifiers
            if (notifierNodeElementCollection.Count != 0)
            {
                Console.WriteLine();
                Console.WriteLine("Notifiers:");
                foreach (UANodeElement notifierNodeElement in
notifierNodeElementCollection)
                    Console.WriteLine(notifierNodeElement);
            }

            // Obtain objects
            UANodeElementCollection objectNodeElementCollection = client.BrowseObjects(
                endpointDescriptor, nodeDescriptor);

            // Recurse
            foreach (UANodeElement objectNodeElement in objectNodeElementCollection)
                BrowseFrom(objectNodeElement);
        }

        // Example output (truncated):
        //
        //
        //Parent node: ObjectsFolder
        //
        //
        //Parent node: Server
        //
        //Notifiers:
        //Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Green (Object)
        //Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=0:/Yellow (Object)
        //
        //
        //Parent node: Server_ServerCapabilities
        //...
    }
}
```

Object Pascal

```
// This example shows how to browse objects under the "Objects" node and display
notifiers.

class procedure BrowseNotifiers.Main;
var
  ObjectNodeId: _UANodeId;
begin
  ObjectNodeId := CoUANodeId.Create;
  ObjectNodeId.StandardName := 'Objects';
  try
    BrowseFrom(ObjectNodeId);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
```

```
    end;
  end;
end;

class procedure BrowseNotifiers.BrowseFrom(NodeId: _UANodeId);
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NotifierNodeElement: _UANodeElement;
  NotifierNodeElementEnumerator: IEnumVariant;
  NotifierNodeElements: _UANodeElementCollection;
  ObjectNodeElement: _UANodeElement;
  ObjectNodeElementEnumerator: IEnumVariant;
  ObjectNodeElements: _UANodeElementCollection;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  WriteLn;
  WriteLn;
  WriteLn('Parent node: ', NodeId.ToString);

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain notifiers
  NotifierNodeElements := Client.BrowseNotifiers(EndpointDescriptor,
NodeId.ExpandedText);

  // Display notifires
  if NotifierNodeElements.Count <> 0 then
  begin
    WriteLn;
    WriteLn('Notifiers:');
    NotifierNodeElementEnumerator := NotifierNodeElements.GetEnumerator;
    while (NotifierNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
    begin
      NotifierNodeElement := IUnknown(Element) as _UANodeElement;
      WriteLn(NotifierNodeElement.ToString);
    end;
  end;

  // Obtain objects
  ObjectNodeElements := Client.BrowseObjects(EndpointDescriptor, NodeId.ExpandedText);

  // Recurse
  ObjectNodeElementEnumerator := ObjectNodeElements.GetEnumerator;
  while (ObjectNodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    ObjectNodeElement := IUnknown(Element) as _UANodeElement;
    BrowseFrom(ObjectNodeElement.NodeId);
  end;
end;

// Example output (truncated):
```

```
//
//
//Parent node: ObjectsFolder
//
//
//Parent node: Server
//
//Notifiers:
//Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
//Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
//
//
//Parent node: Server_ServerCapabilities
//...
```

PHP

```
// This example shows how to browse objects under the "Objects" node and display
notifiers.


// Start browsing from the "Objects" node
$ObjectNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ObjectNodeId->StandardName = "Objects";
try
{
    BrowseFrom($ObjectNodeId);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

function BrowseFrom($NodeId) {
    $EndpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer";

    printf("\n");
    printf("\n");
    printf("Parent node: %s\n", $NodeId);

    // Instantiate the client object
    $Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

    // Obtain notifiers
    $NotifierNodeElements = $Client->BrowseNotifiers($EndpointDescriptor, $NodeId-
>ExpandedText);

    // Display notifires
    if ($NotifierNodeElements->Count != 0) {
        printf("\n");
        printf("Notifiers:\n");
        foreach ($NotifierNodeElements as $NotifierNodeElement)
```

```
        {
            printf("%s\n", $NotifierNodeElement);
        }
    }

    // Obtain objects
    $ObjectNodeElements = $Client->BrowseObjects($EndpointDescriptor, $NodeId-
>ExpandedText);

    // Recurse
    foreach ($ObjectNodeElements as $ObjectNodeElement)
    {
        BrowseFrom($ObjectNodeElement->NodeId);
    }
}

// Example output (truncated):
//
//
//Parent node: ObjectsFolder
//
//
//Parent node: Server
//
//Notifiers:
//Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
//Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
//
//
//Parent node: Server_ServerCapabilities
//...
```

VB.NET

```
' This example shows how to browse objects under the "Objects" node and display
notifiers.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class BrowseNotifiers
        Public Shared Sub Overload2()

            ' Start browsing from the "Objects" node
            Try
                BrowseFrom(UAObjectIds.ObjectsFolder)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try
        End Sub
```

```
        Private Shared Sub BrowseFrom(nodeDescriptor As UANodeDescriptor)

            Console.WriteLine()
            Console.WriteLine()
            Console.WriteLine("Parent node: {0}", nodeDescriptor)

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain notifiers
            Dim notifierNodeElementCollection As UANodeElementCollection =
client.BrowseNotifiers( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Display notifiers
            Console.WriteLine()
            Console.WriteLine("Notifiers:")
            For Each notifierNodeElement As UANodeElement In
notifierNodeElementCollection
                Console.WriteLine(notifierNodeElement)
            Next notifierNodeElement

            ' Obtain objects
            Dim objectNodeElementCollection = client.BrowseObjects( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                nodeDescriptor)

            ' Recurse
            For Each objectNodeElement As UANodeElement In objectNodeElementCollection
                BrowseFrom(objectNodeElement)
            Next objectNodeElement
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to browse objects under the "Objects" node and display
notifiers.

Option Explicit

' Start browsing from the "Objects" node
Dim ObjectsNodeId: Set ObjectsNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ObjectsNodeId.StandardName = "Objects"
On Error Resume Next
BrowseFrom ObjectsNodeId
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

```
Sub BrowseFrom(NodeId)
    Dim endpointDescriptor
    endpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"

    WScript.Echo
    WScript.Echo
    WScript.Echo "Parent node: " & NodeId

    ' Instantiate the client object
    Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

    ' Obtain notifiers
    Dim NotifierNodeElementCollection: Set NotifierNodeElementCollection =
Client.BrowseNotifiers( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Display notifiers
    If NotifierNodeElementCollection.Count <> 0 Then
        WScript.Echo
        WScript.Echo "Notifiers:"
        Dim NotifierNodeElement: For Each NotifierNodeElement In
NotifierNodeElementCollection
            WScript.Echo NotifierNodeElement
        Next
    End If

    ' Obtain objects
    Dim ObjectNodeElementCollection: Set ObjectNodeElementCollection =
Client.BrowseObjects( _
        endpointDescriptor, NodeId.ExpandedText)

    ' Recurse
    Dim ObjectNodeElement: For Each ObjectNodeElement In ObjectNodeElementCollection
        BrowseFrom ObjectNodeElement.NodeId
    Next
End Sub



' Example output (truncated):
'
'
'Parent node: ObjectsFolder
'
'
'Parent node: Server
'
'Notifiers:
'Green -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Green
(Object)
'Yellow -> nsu=http://opcfoundation.org/Quickstarts/AlarmCondition ;ns=2;s=0:/Yellow
(Object)
'
```

```
'
'Parent node: Server_ServerCapabilities
'...
```

13.2.6.4 Examples - OPC UA Alarms&Conditions - Callback using a lambda

C#

```
// This example shows how to subscribe to event notifications and display each incoming
event
// using a callback method that is provided as lambda expression.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;

namespace UADocExamples.AlarmsAndConditions
{
    partial class SubscribeEvent
    {
        public static void CallbackLambda()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object
            var client = new EasyUAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the event
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                (sender, eventArgs) => Console.WriteLine(eventArgs));
            // Remark: Production code would check e.Exception before accessing
e.EventData.

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 2 seconds...");
            System.Threading.Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to event notifications and display each incoming
event
' using a callback method that is provided as lambda expression.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard

Namespace UADocExamples.AlarmsAndConditions
    Partial Friend Class SubscribeEvent
        Public Shared Sub CallbackLambda()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the event
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                UAObjectIds.Server, _
                1000, _
                Sub(sender, eventArgs) Console.WriteLine(eventArgs))
            ' Remark: Production code would check e.Exception before accessing
e.EventData.

            Console.WriteLine("Processing event notifications for 10 seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

13.2.6.5 Examples - OPC UA Alarms&Conditions - Event pull

C#

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class PullEventNotification
    {
        public static void Main1()
```

```
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyUAClient { PullEventNotificationQueueCapacity = 1000
};

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            int endTick = Environment.TickCount + 30 * 1000;
            do
            {
                EasyUAEventNotificationEventArgs eventArgs =
client.PullEventNotification(2 * 1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);
        }

        // Example output (truncated):
        //Subscribing...
        //Processing event notifications for 30 seconds...
        //[] Success
        //[] Success; Refresh; RefreshInitiated
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was
activated" @9/10/2019 8:08:23 PM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity
has increased." @9/10/2019 8:09:07 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity
has increased." @9/10/2019 8:10:09 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity
has increased." @11/9/2019 10:29:42 AM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:11 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:10:19 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
```

```
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity
has increased." @11/9/2019 10:29:42 AM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity
has increased." @9/10/2019 8:08:58 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity
has increased." @9/10/2019 8:09:48 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity
has increased." @11/9/2019 10:29:42 AM
        //[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:02 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:59 PM
        //[] Success; Refresh; RefreshComplete
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:43 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:43 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:44 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:44 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:45 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:45 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:46 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:46 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:47 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:47 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:48 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:48 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:49 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:49 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:50 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:50 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:51 AM
```

```
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:51 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:52 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:52 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:53 AM
        //[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 10:29:53 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:53 AM
        //[] Success; (10 field results) [WestTank] 500! "The alarm severity has
increased." @11/9/2019 10:29:53 AM
        //[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 10:29:53 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:54 AM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
10:29:54 AM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
10:29:55 AM
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

class procedure PullEventNotification.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUAEventNotificationEventArgs;
begin
  // Instantiate the client object and hook events
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullEventNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(
    'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer',
    'nsu=http://opcfoundation.org/UA/;i=2253',  // UAObjectIds.Server
    1000);

  WriteLn('Processing event notifications for 30 seconds...');
  EndTick := Ticks + 30*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullEventNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;
```

```
  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
```

```
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:43
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:43 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:44
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:44 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:45
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:45 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:46
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:46 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:47
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:47 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:48
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:48 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:49
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:49 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:50
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:50 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:51
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:51 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:52
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:52 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:53
AM
//[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:53 AM
//[] Success; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:54
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:54 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:55
AM
//...
```

PHP

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.
```

```
// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$Client->PullEventNotificationQueueCapacity = 1000;

printf("Subscribing...\n");
$Client->SubscribeEvent(
    "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
    "nsu=http://opcfoundation.org/UA/;i=2253",  // UAObjectIds_Server
    1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time();
do {
    $EventArgs = $Client->PullEventNotification(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        printf("%s\n", $EventArgs);
    }
} while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
```

```
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:43
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:43 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:44
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:44 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:45
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:45 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:46
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:46 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:47
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:47 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:48
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:48 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:49
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:49 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:50
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:50 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:51
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:51 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:52
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:52 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:53
AM
//[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
```

```
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:53 AM
//[] Success; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:54
AM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:54 AM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:55
AM
//...
```

VB.NET

```
' This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class PullEventNotification
        Public Shared Sub Main1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            client.PullEventNotificationQueueCapacity = 1000

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent(
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
                UAObjectIds.Server,
                1000)

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Dim endTick As Integer = Environment.TickCount + 30 * 1000
            Do
                Dim eventArgs As EasyUAEventNotificationEventArgs =
client.PullEventNotification(2 * 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
                End If
            Loop While Environment.TickCount < endTick
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to subscribe to event notifications, pull events, and display
each incoming event.

class procedure PullEventNotification.Main;
```

```
var
  Client: EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUAEventNotificationEventArgs;
begin
  // Instantiate the client object and hook events
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullEventNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(
    'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer',
    'nsu=http://opcfoundation.org/UA/;i=2253',  // UAObjectIds.Server
    1000);

  WriteLn('Processing event notifications for 30 seconds...');
  EndTick := GetTickCount + 60*1000;
  while GetTickCount < EndTick do
  begin
    EventArgs := Client.PullEventNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;
```

VBScript

```
Rem This example shows how to subscribe to event notifications, pull events, and
display each incoming event.

Option Explicit

Const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullEventNotificationQueueCapacity = 1000

WScript.Echo "Subscribing..."
Client.SubscribeEvent "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", UAObjectIds_Server, 1000

WScript.Echo "Processing event notifications for 30 seconds..."
Dim endTime: endTime = Now() + 30*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullEventNotification(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
```

```
    End If
Loop While Now() < endTime


' Example output (truncated):
'Subscribing...
'Processing event notifications for 30 seconds...
'[] Success
'[] Success; Refresh; RefreshInitiated
'[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:13 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was acknoweledged."
@11/9/2019 9:56:23 AM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:17 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:12 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:04 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/9/2019 10:29:42 AM
'[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
```

```
increased." @9/10/2019 8:09:59 PM
'[] Success; Refresh; RefreshComplete
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:43 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:43 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:44 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:44 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:45 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:45 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:46 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:46 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:47 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:47 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:48 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:48 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:49 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:49 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:50 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:50 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:51 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:51 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:52 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:52 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:53 AM
'[] Success; (10 field results) [NorthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:53 AM
'[] Success; (10 field results) [WestTank] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
'[] Success; (10 field results) [SouthMotor] 500! "The alarm severity has increased."
@11/9/2019 10:29:53 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:54 AM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 10:29:54 AM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 10:29:55 AM
'...
```

13.2.6.6 Examples - OPC UA Alarms&Conditions - Retrieve base event properties

C#

```
// This example shows how to display specific fields of incoming events that are
derived from base event type.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
```

```
    class BaseEvent
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender,
EasyUAEventNotificationEventArgs e)
        {
            Console.WriteLine();

            // Display the event
            if (e.EventData == null)
            {
                Console.WriteLine(e);
                return;
            }
            UABaseEventObject baseEventObject = e.EventData.BaseEvent;
            Console.WriteLine("Source name: {0}", baseEventObject.SourceName);
            Console.WriteLine("Message: {0}", baseEventObject.Message);
            Console.WriteLine("Severity: {0}", baseEventObject.Severity);
        }
    }
}
```

Object Pascal

```
// This example shows how to display specific fields of incoming events that are
derived from base event type.

type
  TClientEventHandlers12 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers12.Client_EventNotification(
```

```
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  BaseEventObject: _UABaseEventObject;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;

  BaseEventObject := eventArgs.EventData.BaseEvent;
  WriteLn('Source name: ', BaseEventObject.SourceName);
  WriteLn('Message: ', BaseEventObject.Message);
  WriteLn('Severity: ', BaseEventObject.Severity);
end;

class procedure BaseEvent.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers12;
  EndpointDescriptor: string;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers12.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(EndpointDescriptor, UAObjectIds_Server, 1000);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to display specific fields of incoming events that are
```

```
derived from base event type.


class ClientEvents {
    function EventNotification($Sender, $E)
    {
        printf("\n");

        // Display the event
        if (is_null($E->EventData)) {
            printf("%s\n", $E);
            return;
        }

        $BaseEventObject = $E->EventData->BaseEvent;
        printf("Source name: %s\n", $BaseEventObject->SourceName);
        printf("Message: %s\n", $BaseEventObject->Message);
        printf("Severity: %s\n", $BaseEventObject->Severity);
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeEvent($EndpointDescriptor, UAObjectIds_Server, 1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

VB.NET

```
' This example shows how to display specific fields of incoming events that are derived
from base event type.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class BaseEvent
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
```

```
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                UAObjectIds.Server, _
                1000)

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As
EasyUAEventNotificationEventArgs)
            Console.WriteLine()

            ' Display the event
            If e.EventData Is Nothing Then
                Console.WriteLine(e)
                Exit Sub
            End If
            Dim baseEventObject = e.EventData.BaseEvent
            Console.WriteLine("Source name: {0}", baseEventObject.SourceName)
            Console.WriteLine("Message: {0}", baseEventObject.Message)
            Console.WriteLine("Severity: {0}", baseEventObject.Severity)
        End Sub
    End Class
End Namespace
```

13.2.6.7 Examples - OPC UA Alarms&Conditions - Retrieve fields from event data

C#

```
// This example shows how to display all fields of incoming events, or extract specific fields.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class FieldResults
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
```

```
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            Console.WriteLine();

            // Display the event
            if (e.EventData == null)
            {
                Console.WriteLine(e);
                return;
            }
            Console.WriteLine("All fields:");
            foreach (KeyValuePair<UAAttributeField, ValueResult> pair in e.EventData.FieldResults)
            {
                UAAttributeField attributeField = pair.Key;
                ValueResult valueResult = pair.Value;
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult);
            }

            // Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}",
                e.EventData.FieldResults[UABaseEventObject.Operands.SourceName]);

            // Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}",
                e.EventData.FieldResults[UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")]);
        }



        // Example output (truncated):
        //Subscribing...
        //Processing event notifications for 30 seconds...
        //
        //[] Success
        //
        //[] Success; Refresh; RefreshInitiated
        //
        //All fields:
        //  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
        //  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
        //  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
        //  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
        //  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
        //  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
        //  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
        //Source name: Success; EastTank {System.String}
        //Message: Success; The dialog was activated {System.String}
        //
        //All fields:
        //  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
        //  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType
{OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
        //  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
        //  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
```

```
        //  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
        //  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
        //  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
        //  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
        //Source name: Success; EastTank {System.String}
        //Message: Success; The alarm was acknoweledged. {System.String}
        //
        //...
    }
}
```

Object Pascal

```
// This example shows how to display all fields of incoming events, or extract specific fields.

type
  THelperMethods15 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString: string):
_UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
  end;

type
  TClientEventHandlers15 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers15.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  AttributeField: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  EntryEnumerator: IEnumVARIANT;
  ValueResult: OleVariant;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;

  WriteLn('All fields:');

  EntryEnumerator := eventArgs.EventData.FieldResults.GetEnumerator;
  while (EntryEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    AttributeField := IUnknown(Element.Key) as _UAAttributeField;
    ValueResult := IUnknown(Element.Value) as _ValueResult;
    WriteLn('  ', AttributeField.ToString, ' -> ', ValueResult.ToString);
  end;

  // Extracting specific fields using an event type ID and a simple relative path
  WriteLn('Source name: ',
eventArgs.EventData.FieldResults.Item[THelperMethods15.UABaseEventObject_Operands_SourceName.ToUAAttributeField].ToString);

  WriteLn('Message: ',
eventArgs.EventData.FieldResults.Item[THelperMethods15.UABaseEventObject_Operands_Message.ToUAAttributeField].ToString);
end;

class procedure FieldResults.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers15;
  EndpointDescriptor: string;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';
```

```
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers15.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(EndpointDescriptor, UAObjectIds_Server, 1000);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//
//[] Success
//
///[] Success; Refresh; RefreshInitiated
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The dialog was activated {System.String}
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The alarm was acknoweledged. {System.String}
//
//...

class function THelperMethods15.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods15.UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
```

```
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames := BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods15.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods15.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;
```

PHP

```
// This example shows how to display all fields of incoming events, or extract specific fields.


class ClientEvents {
    function EventNotification($Sender, $E)
    {
        printf("\n");

        // Display the event
        if (is_null($E->EventData)) {
            printf("%s\n", $E);
            return;
        }

        printf("All fields:\n");

        foreach ($E->EventData->FieldResults as $Pair)
        {
            $AttributeField = $Pair->Key;
            $ValueResult = $Pair->Value;
            printf("  %s -> %s\n", $AttributeField, $ValueResult);
        }

        // Extracting specific fields using an event type ID and a simple relative path
        printf("Source name: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_SourceName()-
>ToUAAttributeField()));
        printf("Message: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_Message()-
>ToUAAttributeField()));
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeEvent($EndpointDescriptor, UAObjectIds_Server, 1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
```

```
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//
//[] Success
//
//[] Success; Refresh; RefreshInitiated
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The dialog was activated {System.String}
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The alarm was acknoweledged. {System.String}
//
//...
```

VB.NET

```
' This example shows how to display all fields of incoming events, or extract specific fields.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class FieldResults
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
```

```
                UAObjectIds.Server, _
                1000)

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As EasyUAEventNotificationEventArgs)
            Console.WriteLine()

            ' Display the event
            If e.EventData Is Nothing Then
                Console.WriteLine(e)
                Exit Sub
            End If
            Console.WriteLine("All fields:")
            For Each pair In e.EventData.FieldResults
                Dim attributeField = pair.Key
                Dim valueResult = pair.Value
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult)
            Next pair
            ' Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}", _
                e.EventData.FieldResults(UABaseEventObject.Operands.SourceName))
            ' Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}", _
                e.EventData.FieldResults(UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")))
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to display all fields of incoming events, or extract specific fields.

Option Explicit

Const uaObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253"

Dim endpointDescriptor
endpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Client.SubscribeEvent endpointDescriptor, uaObjectIds_Server, 1000

WScript.Echo "Processing event notifications for 30 seconds..."
WScript.Sleep 30*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5*1000



Function UAFilterElements_SimpleAttribute(TypeId, simpleRelativeBrowsePathString)
    Dim BrowsePathParser: Set BrowsePathParser = CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
    Dim QualifiedNames: Set QualifiedNames =
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNAmeCollection

    Dim SimpleAttributeOperand: Set SimpleAttributeOperand =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Set SimpleAttributeOperand.TypeId.NodeId = TypeId
    Set SimpleAttributeOperand.QualifiedNames = QualifiedNames

    Set UAFilterElements_SimpleAttribute = SimpleAttributeOperand
End Function
```

```
Function ObjectTypeIds_BaseEventType
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.StandardName = "BaseEventType"
    Set ObjectTypeIds_BaseEventType = NodeId
End Function

Function UABaseEventObject_Operands_Message
    Set UABaseEventObject_Operands_Message = UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Message")
End Function

Function UABaseEventObject_Operands_SourceName
    Set UABaseEventObject_Operands_SourceName = UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
"/SourceName")
End Function

Sub Client_EventNotification(Sender, e)
    WScript.Echo

    ' Display the event
    If e.EventData Is Nothing Then
        WScript.Echo e
        Exit Sub
    End If
    WScript.Echo "All fields:"
    Dim Pair: For Each Pair In e.EventData.FieldResults
        Dim AttributeField: Set AttributeField = Pair.Key
        Dim ValueResult: Set ValueResult = Pair.Value
        WScript.Echo "  " & AttributeField & " -> " & ValueResult
    Next

    ' Extracting specific fields using an event type ID and a simple relative path
    WScript.Echo "Source name: " &
e.EventData.FieldResults.Item(UABaseEventObject_Operands_SourceName.ToUAAttributeField)
    WScript.Echo "Message: " & e.EventData.FieldResults.Item(UABaseEventObject_Operands_Message.ToUAAttributeField)
End Sub



' Example output (truncated):
'Subscribing...
'Processing event notifications for 30 seconds...
'
'[] Success
'
'[] Success; Refresh; RefreshInitiated
'
'All fields:
'  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
'  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
'  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
'  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
'  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
'  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
'  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
'Source name: Success; EastTank {System.String}
'Message: Success; The dialog was activated {System.String}
'
'All fields:
'  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
'  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
'  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
'  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
'  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
'  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
'  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
'  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
'Source name: Success; EastTank {System.String}
'Message: Success; The alarm was acknoweledged. {System.String}
'
```

```
'...
```

13.2.6.8 Examples - OPC UA Alarms&Conditions - Specify Select clauses in an event filter

C#

```
// This example shows how to select fields for event notifications.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class SelectClauses
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                new UAAttributeFieldCollection
                {
                    // Select specific fields using standard operand symbols
                    UABaseEventObject.Operands.NodeId,
                    UABaseEventObject.Operands.SourceNode,
                    UABaseEventObject.Operands.SourceName,
                    UABaseEventObject.Operands.Time,

                    // Select specific fields using an event type ID and a simple relative path
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message"),
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Severity"),
                });

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            Console.WriteLine();

            // Display the event
            if (e.EventData == null)
            {
                Console.WriteLine(e);
                return;
            }
            Console.WriteLine("All fields:");
            foreach (KeyValuePair<UAAttributeField, ValueResult> pair in e.EventData.FieldResults)
            {
```

```
                UAAttributeField attributeField = pair.Key;
                ValueResult valueResult = pair.Value;
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult);
            }
            // Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}",
                e.EventData.FieldResults[UABaseEventObject.Operands.SourceName]);
            // Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}",
                e.EventData.FieldResults[UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")]);
        }
    }
}
```

Object Pascal

```
// This example shows how to select fields for event notifications.

type
  THelperMethods17 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString: string):
_UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
  end;

type
  TClientEventHandlers17 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers17.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
var
  AttributeField: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  EntryEnumerator: IEnumVARIANT;
  ValueResult: OleVariant;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;

  WriteLn('All fields:');

  EntryEnumerator := eventArgs.EventData.FieldResults.GetEnumerator;
  while (EntryEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    AttributeField := IUnknown(Element.Key) as _UAAttributeField;
    ValueResult := IUnknown(Element.Value) as _ValueResult;
    WriteLn(' ', AttributeField.ToString, ' -> ', ValueResult.ToString);
  end;

  // Extracting specific fields using an event type ID and a simple relative path
  WriteLn('Source name: ',
eventArgs.EventData.FieldResults.Item[THelperMethods17.UABaseEventObject_Operands_SourceName.ToUAAttributeField].ToString);

  WriteLn('Message: ',
eventArgs.EventData.FieldResults.Item[THelperMethods17.UABaseEventObject_Operands_Message.ToUAAttributeField].ToString);
end;

class procedure SelectClauses.Main;
const
```

```
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers17;
  EndpointDescriptor: string;
  EventFilter: _UAEventFilter;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  SelectClauses: UAAttributeFieldCollection;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers17.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');

  SelectClauses := CoUAAttributeFieldCollection.Create;
  // Select specific fields using an event type ID and a simple relative path
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_Time.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_Message.ToUAAttributeField);
  SelectClauses.Add(THelperMethods17.UABaseEventObject_Operands_Severity.ToUAAttributeField);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := SelectClauses;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;


class function THelperMethods17.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods17.UAFilterElements_SimpleAttribute(TypeId: _UANodeId; simpleRelativeBrowsePathString:
string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
```

```
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames := BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods17.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods17.UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods17.UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods17.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;

class function THelperMethods17.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods17.UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;
```

PHP

```
// This example shows how to select fields for event notifications.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

class ClientEvents {
    function EventNotification($Sender, $E)
    {
        printf("\n");

        // Display the event
        if (is_null($E->EventData)) {
            printf("%s\n", $E);
            return;
        }

        printf("All fields:\n");

        foreach ($E->EventData->FieldResults as $Pair)
        {
            $AttributeField = $Pair->Key;
            $ValueResult = $Pair->Value;
            printf("  %s -> %s\n", $AttributeField, $ValueResult);
        }

        // Extracting specific fields using an event type ID and a simple relative path
        printf("Source name: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_SourceName()-
>ToUAAttributeField()));
        printf("Message: %s\n", $E->EventData->FieldResults->Item(UABaseEventObject_Operands_Message()-
>ToUAAttributeField()));
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";
```

```
// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");

$SelectClauses = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
// Select specific fields using an event type ID and a simple relative path
$SelectClauses->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField());
$SelectClauses->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField());

$EventFilter = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter->SelectClauses = $SelectClauses;

$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->EventFilter = $EventFilter;
$MonitoringParameters->QueueSize = 1000;
$MonitoringParameters->SamplingInterval = 1000;

$MonitoredItemArguments = new COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments->NodeDescriptor->NodeId->StandardName = "Server";
$MonitoredItemArguments->MonitoringParameters = $MonitoringParameters;
// $MonitoredItemArguments->SubscriptionParameters->PublishingInterval = 0;
$MonitoredItemArguments->AttributeId = UAAttributeId_EventNotifier;

$Arguments[0] = $MonitoredItemArguments;

$Client->SubscribeMultipleMonitoredItems($Arguments);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}
```

```
function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}


// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//
//[] Success
//
//[] Success; Refresh; RefreshInitiated
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?OnlineState {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {95, 68, 22, 205, 114, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; DialogConditionType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 9/10/2019 8:08:23 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The dialog was activated {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 100 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The dialog was activated {System.String}
//
//All fields:
//  NodeId="BaseEventType", NodeId -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank?Red {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/EventId -> Success; [16] {124, 156, 219, 54, 120, ...} {System.Byte[]}
//  NodeId="BaseEventType"/EventType -> Success; ExclusiveDeviationAlarmType {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceNode -> Success; nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Colours/EastTank {OpcLabs.EasyOpc.UA.AddressSpace.UANodeId}
//  NodeId="BaseEventType"/SourceName -> Success; EastTank {System.String}
//  NodeId="BaseEventType"/Time -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/ReceiveTime -> Success; 10/14/2019 4:00:13 PM {System.DateTime}
//  NodeId="BaseEventType"/LocalTime -> Success; 00:00, DST {OpcLabs.EasyOpc.UA.UATimeZoneData}
//  NodeId="BaseEventType"/Message -> Success; The alarm was acknoweledged. {System.String}
//  NodeId="BaseEventType"/Severity -> Success; 500 {System.Int32}
//Source name: Success; EastTank {System.String}
//Message: Success; The alarm was acknoweledged. {System.String}
//
//...
```

VB.NET

```
' This example shows how to select fields for event notifications.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class SelectClauses
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            ' Select specific fields using standard operand symbols
            ' Select specific fields using an event type ID and a simple relative path
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000, _
                New UAAttributeFieldCollection() From
                {
```

```
                    UABaseEventObject.Operands.NodeId,
                    UABaseEventObject.Operands.SourceNode,
                    UABaseEventObject.Operands.SourceName,
                    UABaseEventObject.Operands.Time,
 _
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message"),
                    UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Severity")
                })

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As EasyUAEventNotificationEventArgs)
            Console.WriteLine()

            ' Display the event
            If e.EventData Is Nothing Then
                Console.WriteLine(e)
                Exit Sub
            End If
            For Each pair In e.EventData.FieldResults
                Dim attributeField = pair.Key
                Dim valueResult = pair.Value
                Console.WriteLine("  {0} -> {1}", attributeField, valueResult)
            Next pair
            ' Extracting a specific field using a standard operand symbol
            Console.WriteLine("Source name: {0}", _
                e.EventData.FieldResults(UABaseEventObject.Operands.SourceName))
            ' Extracting a specific field using an event type ID and a simple relative path
            Console.WriteLine("Message: {0}", _
                e.EventData.FieldResults(UAFilterElements.SimpleAttribute(UAObjectTypeIds.BaseEventType, "/Message")))
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to select fields for event notifications.

type
  TClientEventHandlers4 = class
    procedure OnEventNotification(
      Sender: TObject;
      sender0: OleVariant;
      eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers4.OnEventNotification(
  Sender: TObject;
  sender0: OleVariant;
  eventArgs: _EasyUAEventNotificationEventArgs);

  function ToUAAttributeField(Operand: UASimpleAttributeOperand): UAAttributeField;
  var
    AttributeField: UAAttributeField;
  begin
    AttributeField := CoUAAttributeField.Create;
    AttributeField.Operand := Operand;
    ToUAAttributeField := AttributeField;
  end;

  function UAFilterElements_SimpleAttribute(TypeId: UANodeId; SimpleRelativeBrowsePathString: string):
UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
    BrowsePathParser: UABrowsePathParser;
  begin
    BrowsePathParser := CoUABrowsePathParser.Create;
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId := TypeId;
    Operand.QualifiedNames := BrowsePathParser.ParseRelative(SimpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
    UAFilterElements_SimpleAttribute := Operand;
```

```
  end;

  function ObjectTypeIds_BaseEventType: UANodeId;
  var
    NodeId: UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    ObjectTypeIds_BaseEventType := NodeId;
  end;

var
  Count: Cardinal;
  Element: OleVariant;
  EntryEnumerator: IEnumVariant;
  Entry: DictionaryEntry2;
  AttributeField: UAAttributeField;
  AValueResult: ValueResult;
begin
  WriteLn;

  // Display the event
  if eventArgs.EventData = nil then
  begin
    WriteLn(eventArgs.ToString);
    Exit;
  end;
  WriteLn('All fields:');
  EntryEnumerator := eventArgs.EventData.FieldResults.GetEnumerator;
  while EntryEnumerator.Next(1, Element, Count) = S_OK do
  begin
    Entry := IUnknown(Element) as DictionaryEntry2;
    AttributeField := IUnknown(Entry.key) as UAAttributeField;
    AValueResult := IUnknown(Entry.Value) as ValueResult;
    WriteLn('  ', AttributeField.ToString, ' -> ', AValueResult.ToString);
  end;
  // Extracting a specific field using an event type ID and a simple relative path
  WriteLn('Source name: ',
    eventArgs.EventData.FieldResults[ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/SourceName'))].ToString);
  WriteLn('Message: ',
    eventArgs.EventData.FieldResults[ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType,
'/Message'))].ToString);
end;

class procedure SelectClauses.Main;

  function ToUAAttributeField(Operand: UASimpleAttributeOperand): UAAttributeField;
  var
    AttributeField: UAAttributeField;
  begin
    AttributeField := CoUAAttributeField.Create;
    AttributeField.Operand := Operand;
    ToUAAttributeField := AttributeField;
  end;

  function ObjectTypeIds_BaseEventType: UANodeId;
  var
    NodeId: UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    ObjectTypeIds_BaseEventType := NodeId;
  end;

  function UABaseEventObject_Operands_NodeId: UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
  begin
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId.StandardName := 'BaseEventType';
    Operand.AttributeId := UAAttributeId_NodeId;
    UABaseEventObject_Operands_NodeId := Operand;
  end;

  function UAFilterElements_SimpleAttribute(TypeId: UANodeId; SimpleRelativeBrowsePathString: string):
UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
```

```
    BrowsePathParser: UABrowsePathParser;
  begin
    BrowsePathParser := CoUABrowsePathParser.Create;
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId := TypeId;
    Operand.QualifiedNames := BrowsePathParser.ParseRelative(SimpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
    UAFilterElements_SimpleAttribute := Operand;
  end;

var
  Arguments: OleVariant;
  EvsClient: TEvsEasyUAClient;
  Client: EasyUAClient;
  ClientEventHandlers: TClientEventHandlers4;
  Handle: Cardinal;
  HandleArray: OleVariant;
  MonitoredItemArguments: EasyUAMonitoredItemArguments;
  MonitoringParameters: UAMonitoringParameters;
  EventFilter: UAEventFilter;
  SelectClauses: UAAttributeFieldCollection;
begin
  // Instantiate the client object and hook events
  EvsClient := TEvsEasyUAClient.Create(nil);
  Client := EvsClient.ComServer;
  ClientEventHandlers := TClientEventHandlers4.Create;
  EvsClient.OnEventNotification := @ClientEventHandlers.OnEventNotification;

  WriteLn('Subscribing...');

  SelectClauses := CoUAAttributeFieldCollection.Create;
  // Select specific fields using an event type ID and a simple relative path
  SelectClauses.Add(ToUAAttributeField(UABaseEventObject_Operands_NodeId));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message')));
  SelectClauses.Add(ToUAAttributeField(UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity')));

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := SelectClauses;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(PSafeArray(TVarData(Arguments).VArray)));

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Done...');
end;
```

13.2.6.9 Examples - OPC UA Alarms&Conditions - Specify Where clause of an event filter

C#

```
// This example shows how to specify criteria for event notifications.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class WhereClause
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(
                endpointDescriptor,
                UAObjectIds.Server,
                1000,
                new UAEventFilterBuilder(
                    // Either the severity is >= 500, or the event comes from a specified source
node
                    UAFilterElements.Or(
                        UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity,
500),
                        UAFilterElements.Equals(
                            UABaseEventObject.Operands.SourceNode,
                            new
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor"))),

                    UABaseEventObject.AllFields));

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs e)
        {
            // Display the event
            Console.WriteLine(e);
```

```
        }
    }
}
```

Object Pascal

```
// This example shows how to specify criteria for event notifications.

type
  THelperMethods10 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_AllFields: _UAAttributeFieldCollection; static;
  end;

type
  TClientEventHandlers10 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers10.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
  WriteLn(eventArgs.ToString);
end;

class procedure WhereClause.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers10;
  Element1, Element2: _UAContentFilterElement;
  EndpointDescriptor: string;
  EventFilter: _UAEventFilter;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  Operand1, Operand3: _UASimpleAttributeOperand;
  Operand2, Operand4: _UALiteralOperand;
  SourceNodeId: _UANodeId;
  WhereClause: _UAContentFilterElement;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer';
```

```
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers10.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');

  WhereClause := CoUAContentFilterElement.Create;

  // Either the severity is >= 500, or the event comes from a specified source node
  Operand1 := THelperMethods10.UABaseEventObject_Operands_Severity;
  Operand2 := CoUALiteralOperand.Create;
  Operand2.Value := 500;
  Element1 := CoUAContentFilterElement.Create;
  Element1.FilterOperator := UAFilterOperator_GreaterThanOrEqual;
  Element1.FilterOperands.Add(Operand1);
  Element1.FilterOperands.Add(Operand2);
  Operand3 := THelperMethods10.UABaseEventObject_Operands_SourceNode;
  SourceNodeId := CoUANodeId.Create;
  SourceNodeId.ExpandedText :=
'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor';
  Operand4 := CoUALiteralOperand.Create;
  Operand4.Value := SourceNodeId;
  Element2 := CoUAContentFilterElement.Create;
  Element2.FilterOperator := UAFilterOperator_Equals;
  Element2.FilterOperands.Add(Operand3);
  Element2.FilterOperands.Add(Operand4);
  WhereClause.FilterOperator := UAFilterOperator_Or;
  WhereClause.FilterOperands.Add(Element1);
  WhereClause.FilterOperands.Add(Element2);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := THelperMethods10.UABaseEventObject_AllFields;
  EventFilter.WhereClause := WhereClause;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
```

```
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;


class function THelperMethods10.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods10.UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames :=
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods10.UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
end;

class function THelperMethods10.UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
end;

class function THelperMethods10.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods10.UABaseEventObject_Operands_SourceNode:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods10.UABaseEventObject_Operands_SourceName:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods10.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;
```

```
class function THelperMethods10.UABaseEventObject_Operands_ReceiveTime:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
end;

class function THelperMethods10.UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
end;

class function THelperMethods10.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods10.UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;

class function THelperMethods10.UABaseEventObject_AllFields: _UAAttributeFieldCollection;
var
  Fields: _UAAttributeFieldCollection;
begin
  Fields := CoUAAttributeFieldCollection.Create;
  Fields.Add(UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventType.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Time.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_LocalTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Message.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Severity.ToUAAttributeField);

  Result := Fields;
end;
```

PHP

```
// This example shows how to specify criteria for event notifications.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

const UAFilterOperator_Equals = 1;
const UAFilterOperator_GreaterThanOrEqual = 5;
const UAFilterOperator_Or = 12;

const UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';

class ClientEvents {
    function EventNotification($Sender, $E)
    {
        // Display the event
        printf("%s\n", $E);
    }
}
```

```
$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

$WhereClause = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");

// Either the severity is >= 500, or the event comes from a specified source node
$Operand1 = UABaseEventObject_Operands_Severity();
$Operand2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand2->Value = 500;
$Element1 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$Element1->FilterOperator = UAFilterOperator_GreaterThanOrEqual;
$Element1->FilterOperands->Add($Operand1);
$Element1->FilterOperands->Add($Operand2);
$Operand3 = UABaseEventObject_Operands_SourceNode();
$SourceNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$SourceNodeId->ExpandedText =
"nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor";
$Operand4 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand4->Value = $SourceNodeId;
$Element2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$Element2->FilterOperator = UAFilterOperator_Equals;
$Element2->FilterOperands->Add($Operand3);
$Element2->FilterOperands->Add($Operand4);
$WhereClause->FilterOperator = UAFilterOperator_Or;
$WhereClause->FilterOperands->Add($Element1);
$WhereClause->FilterOperands->Add($Element2);

$EventFilter = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter->SelectClauses = UABaseEventObject_AllFields();
$EventFilter->WhereClause = $WhereClause;

$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoringParameters->EventFilter = $EventFilter;
$MonitoringParameters->QueueSize = 1000;

$MonitoredItemArguments = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments->NodeDescriptor->NodeId->StandardName = "Server";
$MonitoredItemArguments->MonitoringParameters = $MonitoringParameters;
/$MonitoredItemArguments->SubscriptionParameters->PublishingInterval = 0;
$MonitoredItemArguments->AttributeId = UAAttributeId_EventNotifier;

$arguments[0] = $MonitoredItemArguments;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Processing monitored item changed events for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

```
function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser->ParseRelative($simpleRelativeBrowsePathString)-
>ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_EventId() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventId");
}

function UABaseEventObject_Operands_EventType() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventType");
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}

function UABaseEventObject_Operands_ReceiveTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/ReceiveTime");
}

function UABaseEventObject_Operands_LocalTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/LocalTime");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}

function UABaseEventObject_AllFields() {
    $Fields = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
    $Fields->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventType()->ToUAAttributeField);
```

```
    $Fields->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_ReceiveTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_LocalTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField);

    return $Fields;
}
```

VB.NET

```
' This example shows how to specify criteria for event notifications.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class WhereClause
        Public Shared Sub Main1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            ' Either the severity is >= 500, or the event comes from a specified source node
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer", _
                UAObjectIds.Server, _
                1000, _
                New UAEventFilterBuilder( _
                    UAFilterElements.Or( _
                        UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity,
500),
                        UAFilterElements.Equals( _
                            UABaseEventObject.Operands.SourceNode,
                            New
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor"))),

                    UABaseEventObject.AllFields))

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As
EasyUAEventNotificationEventArgs)
            ' Display the event
            Console.WriteLine(e)
        End Sub
    End Class
```

```
End Namespace
```

Free Pascal

```
// This example shows how to specify criteria for event notifications.

type
  TClientEventHandlers3 = class
    procedure OnEventNotification(
      Sender: TObject;
      sender0: OleVariant;
      eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers3.OnEventNotification(
  Sender: TObject;
  sender0: OleVariant;
  eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
  WriteLn(eventArgs.ToString);
end;

class procedure WhereClause.Main;

  function ToUAAttributeField(Operand: UASimpleAttributeOperand): UAAttributeField;
  var
    AttributeField: UAAttributeField;
  begin
    AttributeField := CoUAAttributeField.Create;
    AttributeField.Operand := Operand;
    ToUAAttributeField := AttributeField;
  end;

  function ObjectTypeIds_BaseEventType: UANodeId;
  var
    NodeId: UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    ObjectTypeIds_BaseEventType := NodeId;
  end;

  function UAFilterElements_SimpleAttribute(TypeId: UANodeId; SimpleRelativeBrowsePathString:
string): UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
    BrowsePathParser: UABrowsePathParser;
  begin
    BrowsePathParser := CoUABrowsePathParser.Create;
    Operand := CoUASimpleAttributeOperand.Create;
    Operand.TypeId.NodeId := TypeId;
    Operand.QualifiedNames :=
BrowsePathParser.ParseRelative(SimpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
    UAFilterElements_SimpleAttribute := Operand;
  end;

  function UABaseEventObject_Operands_NodeId: UASimpleAttributeOperand;
  var
    Operand: UASimpleAttributeOperand;
  begin
    Operand := CoUASimpleAttributeOperand.Create;
```

```
    Operand.TypeId.NodeId.StandardName := 'BaseEventType';
    Operand.AttributeId := UAAttributeId_NodeId;
    UABaseEventObject_Operands_NodeId := Operand;
  end;

  function UABaseEventObject_Operands_EventId: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_EventId :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
  end;

  function UABaseEventObject_Operands_EventType: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_EventType :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
  end;

  function UABaseEventObject_Operands_SourceNode: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_SourceNode :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
  end;

  function UABaseEventObject_Operands_SourceName: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_SourceName :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
  end;

  function UABaseEventObject_Operands_Time: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_Time :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
  end;

  function UABaseEventObject_Operands_ReceiveTime: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_ReceiveTime :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
  end;

  function UABaseEventObject_Operands_LocalTime: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_LocalTime :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
  end;

  function UABaseEventObject_Operands_Message: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_Message :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
  end;

  function UABaseEventObject_Operands_Severity: UASimpleAttributeOperand;
  begin
    UABaseEventObject_Operands_Severity :=
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
  end;

  function UABaseEventObject_AllFields: UAAttributeFieldCollection;
  var
    Fields: UAAttributeFieldCollection;
  begin
```

```
    Fields := CoUAAttributeFieldCollection.Create;

    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_NodeId));

    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_EventId));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_EventType));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_SourceNode));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_SourceName));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_Time));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_ReceiveTime));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_LocalTime));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_Message));
    Fields.Add(ToUAAttributeField(UABaseEventObject_Operands_Severity));

    UABaseEventObject_AllFields := Fields;
  end;

var
  Arguments: OleVariant;
  EvsClient: TEvsEasyUAClient;
  Client: EasyUAClient;
  ClientEventHandlers: TClientEventHandlers3;
  Handle: Cardinal;
  HandleArray: OleVariant;
  MonitoredItemArguments: EasyUAMonitoredItemArguments;
  MonitoringParameters: UAMonitoringParameters;
  EventFilter: UAEventFilter;
  WhereClause: UAContentFilterElement;
  Operand1: UASimpleAttributeOperand;
  Operand2: UALiteralOperand;
  Operand3: UASimpleAttributeOperand;
  Operand4: UALiteralOperand;
  SourceNodeId: UANodeId;
  Element1, Element2: UAContentFilterElement;
  BaseEventType: UANodeId;
begin
  // Instantiate the client object and hook events
  EvsClient := TEvsEasyUAClient.Create(nil);
  Client := EvsClient.ComServer;
  ClientEventHandlers := TClientEventHandlers3.Create;
  EvsClient.OnEventNotification := @ClientEventHandlers.OnEventNotification;

  WriteLn('Subscribing...');

  WhereClause := CoUAContentFilterElement.Create;
  BaseEventType := CoUaNodeId.Create;
  BaseEventType.StandardName := 'BaseEventType';

  // Either the severity is >= 500, or the event comes from a specified source node
  Operand1 := UABaseEventObject_Operands_Severity;
  Operand2 := CoUALiteralOperand.Create;
  Operand2.Value := 500;
  Element1 := CoUAContentFilterElement.Create;
  Element1.FilterOperator := UAFilterOperator_GreaterThanOrEqual;
  Element1.FilterOperands.Add(Operand1);
  Element1.FilterOperands.Add(Operand2);
  Operand3 := UABaseEventObject_Operands_SourceNode;
  SourceNodeId := CoUANodeId.Create;
  SourceNodeId.ExpandedText :=
'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor';
  Operand4 := CoUALiteralOperand.Create;
  Operand4.Value := SourceNodeId;
  Element2 := CoUAContentFilterElement.Create;
```

```
  Element2.FilterOperator := UAFilterOperator_Equals;
  Element2.FilterOperands.Add(Operand3);
  Element2.FilterOperands.Add(Operand4);
  WhereClause.FilterOperator := UAFilterOperator_Or;
  WhereClause.FilterOperands.Add(Element1);
  WhereClause.FilterOperands.Add(Element2);

  EventFilter := CoUAEventFilter.Create;
  EventFilter.SelectClauses := UABaseEventObject_AllFields;
  EventFilter.WhereClause := WhereClause;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoringParameters.EventFilter := EventFilter;
  MonitoringParameters.QueueSize := 1000;

  MonitoredItemArguments := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';
  MonitoredItemArguments.NodeDescriptor.NodeId.StandardName := 'Server';
  MonitoredItemArguments.MonitoringParameters := MonitoringParameters;
  //MonitoredItemArguments.SubscriptionParameters.PublishingInterval := 0;
  MonitoredItemArguments.AttributeId := UAAttributeId_EventNotifier;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoredItemArguments;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(PSafeArray(TVarData(Arguments).VArray)));

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Done...');
end;
```

13.2.6.10 Examples - OPC UA Alarms&Conditions - Subscribe to a single event

C#

```
// This example shows how to subscribe to event notifications and display each incoming
event.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples.AlarmsAndConditions
{
    partial class SubscribeEvent
    {
        public static void Overload1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeEvent(endpointDescriptor, UAObjectIds.Server, 1000);

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender,
EasyUAEventNotificationEventArgs e)
        {
            // Display the event
            Console.WriteLine(e);
        }



        // Example output (truncated):
        //Subscribing...
        //Processing event notifications for 30 seconds...
        //[] Success
        //[] Success; Refresh; RefreshInitiated
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was
activated" @9/10/2019 8:08:23 PM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
        //[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity
has increased." @9/10/2019 8:09:07 PM
        //[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity
has increased." @9/10/2019 8:10:09 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
```

```
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity
has increased." @11/9/2019 2:56:37 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:11 PM
        //[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:10:19 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
        //[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity
has increased." @11/9/2019 2:56:37 PM
        //[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity
has increased." @9/10/2019 8:08:58 PM
        //[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity
has increased." @9/10/2019 8:09:48 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity
has increased." @11/9/2019 2:56:37 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:02 PM
        //[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity
has increased." @9/10/2019 8:09:59 PM
        //[] Success; Refresh; RefreshComplete
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:41 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:41 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:42 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:42 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:43 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:43 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:44 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:44 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:45 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:45 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:46 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:46 PM
```

```
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:47 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:47 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:48 PM
        //[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has
increased." @11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:48 PM
        //[] Success; (10 field results) [WestTank] 700! "The alarm severity has
increased." @11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has
increased." @11/9/2019 2:56:48 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:49 PM
        //[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019
2:56:49 PM
        //[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019
2:56:50 PM
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to event notifications and display each incoming
event.

type
  TClientEventHandlers18 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers18.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
  if eventArgs.Succeeded then
    WriteLn(eventArgs.ToString)
  else
    WriteLn(Format('*** Failure: %s', [eventArgs.ErrorMessageBrief]));
end;

class procedure SubscribeEvent.Main;
const
  UAObjectIds_Server = 'nsu=http://opcfoundation.org/UA/;i=2253';
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers18;
  EndpointDescriptor: string;
begin
```

```
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers18.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  WriteLn('Subscribing...');
  Client.SubscribeEvent(EndpointDescriptor, UAObjectIds_Server, 1000);

  WriteLn('Processing event notifications for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);

end;

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
```

```
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50 PM
//...
```

PHP

```
// This example shows how to subscribe to event notifications and display each incoming
event.
```

```
class ClientEvents {
    function EventNotification($Sender, $E)
    {
        // Display the event
        if ($E->Succeeded) {
            printf("%s\n", $E);
        }
        else {
            printf(" *** Failure: %s\n", $E->ErrorMessageBrief);
        }
    }
}

const UAObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253";

$EndpointDescriptor = "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeEvent($EndpointDescriptor, UAObjectIds_Server, 1000);

printf("Processing event notifications for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);

// Example output (truncated):
//Subscribing...
//Processing event notifications for 30 seconds...
//[] Success
//[] Success; Refresh; RefreshInitiated
//[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was
acknoweledged." @11/9/2019 9:56:23 AM
//[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
//[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
```

```
acknoweledged." @10/14/2019 4:00:16 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
//[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:12 PM
//[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was
acknoweledged." @10/14/2019 4:00:04 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
//[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[] Success; Refresh; RefreshComplete
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
//[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
//[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50 PM
```

```
//...
```

VB.NET

```
' This example shows how to subscribe to event notifications and display each incoming
event.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Friend Class SubscribeEvent
        Public Shared Sub Overload1()
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeEvent( _
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer",
_
                UAObjectIds.Server, _
                1000)

            Console.WriteLine("Processing event notifications for 10 seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As
EasyUAEventNotificationEventArgs)
            ' Display the event
            Console.WriteLine(e)
        End Sub
    End Class
End Namespace
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to event notifications and display each
incoming event.

' The client object, with events
'Public WithEvents Client1 As EasyUAClient

Private Sub SubscribeEvent_Main_Command_Click()
    OutputText = ""

    Set Client1 = New EasyUAClient
```

```
    OutputText = OutputText & "Subscribing..." & vbCrLf
    Call Client1.SubscribeEvent("opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer",
"nsu=http://opcfoundation.org/UA/;i=2253", 1000)

    OutputText = OutputText & "Processing event notifications for 30 seconds..." &
vbCrLf
    Pause 30000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Client1.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client1 = Nothing
End Sub

Private Sub Client1_EventNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUAEventNotificationEventArgs)
    ' Display the event
    If EventArgs.Succeeded Then
        OutputText = OutputText & EventArgs & vbCrLf
    Else
        OutputText = OutputText & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to event notifications and display each
incoming event.

Option Explicit

Const uaObjectIds_Server = "nsu=http://opcfoundation.org/UA/;i=2253"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Client.SubscribeEvent "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", uaObjectIds_Server, 1000

WScript.Echo "Processing event notifications for 30 seconds..."
WScript.Sleep 30*1000



Sub Client_EventNotification(Sender, e)
    ' Display the event
    WScript.Echo e
End Sub
```

```
' Example output (truncated):
'Subscribing...
'Processing event notifications for 30 seconds...
'[] Success
'[] Success; Refresh; RefreshInitiated
'[] Success; Refresh; (10 field results) [EastTank] 100! "The dialog was activated"
@9/10/2019 8:08:23 PM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:13 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm was acknoweledged."
@11/9/2019 9:56:23 AM
'[] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was acknoweledged."
@10/14/2019 4:00:17 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:07 PM
'[] Success; Refresh; (10 field results) [EastTank] 100! "The alarm severity has
increased." @9/10/2019 8:10:09 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:11 PM
'[] Success; Refresh; (10 field results) [NorthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:10:19 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:12 PM
'[] Success; Refresh; (10 field results) [WestTank] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
'[] Success; Refresh; (10 field results) [WestTank] 300! "The alarm was acknoweledged."
@10/14/2019 4:00:04 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:08:58 PM
'[] Success; Refresh; (10 field results) [WestTank] 100! "The alarm severity has
increased." @9/10/2019 8:09:48 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was activated"
@9/10/2019 8:08:25 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm severity has
increased." @11/9/2019 2:56:37 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
'[] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
'[] Success; Refresh; RefreshComplete
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:41 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:41 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:42 PM
```

```
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:42 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:43 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:43 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:44 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:44 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:45 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:45 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:46 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:46 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:47 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:47 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:48 PM
'[] Success; (10 field results) [NorthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:48 PM
'[] Success; (10 field results) [WestTank] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
'[] Success; (10 field results) [SouthMotor] 700! "The alarm severity has increased."
@11/9/2019 2:56:48 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:49 PM
'[] Success; (10 field results) [Internal] 500! "Events Raised" @11/9/2019 2:56:49 PM
'[] Success; (10 field results) [Internal] 500! "Raising Events" @11/9/2019 2:56:50 PM
'...
```

13.2.6.11 Examples - OPC UA Alarms&Conditions - Subscribe to multiple events

C#

```
// This example shows how to subscribe to multiple events.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.AlarmsAndConditions;
using OpcLabs.EasyOpc.UA.Filtering;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.AlarmsAndConditions
{
    class SubscribeMultipleMonitoredItems
    {
        public static void Events()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.EventNotification += client_EventNotification;

            Console.WriteLine("Subscribing...");
```

```
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments("firstState",
                            endpointDescriptor,
                        UAObjectIds.Server,
                        new UAMonitoringParameters(1000, new UAEventFilterBuilder(

UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500),
                            UABaseEventObject.AllFields)))
                        { AttributeId = UAAttributeId.EventNotifier },
                    new EasyUAMonitoredItemArguments("secondState",
                            endpointDescriptor,
                        UAObjectIds.Server,
                        new UAMonitoringParameters(2000, new UAEventFilterBuilder(
                            UAFilterElements.Equals(
                                UABaseEventObject.Operands.SourceNode,
                                new
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor")),

                            UABaseEventObject.AllFields)))
                        { AttributeId = UAAttributeId.EventNotifier },
                });

            Console.WriteLine("Processing event notifications for 30 seconds...");
            System.Threading.Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_EventNotification(object sender, EasyUAEventNotificationEventArgs
e)
        {
            // Display the event
            Console.WriteLine(e);
        }



        // Example output (truncated):
        //Subscribing...
        //Processing monitored item changed events for 30 seconds...
        //[firstState] Success
        //[secondState] Success
        //[firstState] Success; Refresh; RefreshInitiated
        //[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
        //[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
        //[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
        //[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
        //[firstState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[firstState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[firstState] Success; Refresh; RefreshComplete
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
```

```
7:48:08 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:08 PM
        //[secondState] Success; Refresh; RefreshInitiated
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm is
active." @11/8/2019 7:48:07 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm
severity has increased." @9/10/2019 8:09:02 PM
        //[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm
severity has increased." @9/10/2019 8:09:59 PM
        //[secondState] Success; Refresh; RefreshComplete
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:09 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:09 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:10 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:10 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:11 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:11 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:12 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:12 PM
        //[firstState] Success; (10 field results) [EastTank] 500! "The alarm severity has
increased." @11/8/2019 7:48:13 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:13 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:13 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:14 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:14 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:15 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:15 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:16 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:16 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:17 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:17 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019
7:48:18 PM
        //[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019
7:48:18 PM
        //[secondState] Success; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/8/2019 7:48:18 PM
        //...
```

```
    }
}
```

Object Pascal

```
// This example shows how to subscribe to multiple events.

type
  THelperMethods19 = class
    class function ObjectTypeIds_BaseEventType: _UANodeId; static;
    class function UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_EventType: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceNode: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_SourceName: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Time: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_ReceiveTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_LocalTime: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Message: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_Operands_Severity: _UASimpleAttributeOperand; static;
    class function UABaseEventObject_AllFields: _UAAttributeFieldCollection; static;
  end;

type
  TClientEventHandlers19 = class
    procedure Client_EventNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAEventNotificationEventArgs);
  end;

procedure TClientEventHandlers19.Client_EventNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAEventNotificationEventArgs);
begin
  // Display the event
    WriteLn(eventArgs.ToString);
end;


class procedure SubscribeMultipleMonitoredItems.Events;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers19;
  EndpointDescriptor: string;
  EventFilter1, EventFilter2: _UAEventFilter;
  MonitoredItemArguments1, MonitoredItemArguments2: _EasyUAMonitoredItemArguments;
  MonitoringParameters1, MonitoringParameters2: _UAMonitoringParameters;
  Operand11, Operand21: _UASimpleAttributeOperand;
  Operand12, Operand22: _UALiteralOperand;
  ServerNodeId1, ServerNodeId2: _UANodeId;
  SourceNodeId: _UANodeId;
  WhereClause1, WhereClause2: _UAContentFilterElement;
begin
  EndpointDescriptor := 'opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer';

  // set MonitoredItemArguments1
```

```
  // Event filter: The severity is >= 500.
  Operand11 := THelperMethods19.UABaseEventObject_Operands_Severity;
  Operand12 := CoUALiteralOperand.Create;
  Operand12.Value := 500;
  WhereClause1 := CoUAContentFilterElement.Create;
  WhereClause1.FilterOperator := UAFilterOperator_GreaterThanOrEqual;
  WhereClause1.FilterOperands.Add(Operand11);
  WhereClause1.FilterOperands.Add(Operand12);

  EventFilter1 := CoUAEventFilter.Create;
  EventFilter1.SelectClauses := THelperMethods19.UABaseEventObject_AllFields;
  EventFilter1.WhereClause := WhereClause1;

  ServerNodeId1 := CoUANodeId.Create;
  ServerNodeId1.StandardName := 'Server';

  MonitoringParameters1 := CoUAMonitoringParameters.Create;
  MonitoringParameters1.EventFilter := EventFilter1;
  MonitoringParameters1.QueueSize := 1000;
  MonitoringParameters1.SamplingInterval := 1000;

  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.AttributeId := UAAttributeId_EventNotifier;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters1;
  MonitoredItemArguments1.NodeDescriptor.NodeId := ServerNodeId1;
  MonitoredItemArguments1.State := 'firstState';

  // set MonitoredItemArguments2
  // Event filter: The event comes from a specified source node.
  Operand21 := THelperMethods19.UABaseEventObject_Operands_SourceNode;
  SourceNodeId := CoUANodeId.Create;
  SourceNodeId.ExpandedText := 'nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Metals/SouthMotor';
  Operand22 := CoUALiteralOperand.Create;
  Operand22.Value := SourceNodeId;
  WhereClause2 := CoUAContentFilterElement.Create;
  WhereClause2.FilterOperator := UAFilterOperator_Equals;
  WhereClause2.FilterOperands.Add(Operand21);
  WhereClause2.FilterOperands.Add(Operand22);

  EventFilter2 := CoUAEventFilter.Create;
  EventFilter2.SelectClauses := THelperMethods19.UABaseEventObject_AllFields;
  EventFilter2.WhereClause := WhereClause2;

  ServerNodeId2 := CoUANodeId.Create;
  ServerNodeId2.StandardName := 'Server';

  MonitoringParameters2 := CoUAMonitoringParameters.Create;
  MonitoringParameters2.EventFilter := EventFilter2;
  MonitoringParameters2.QueueSize := 1000;
  MonitoringParameters2.SamplingInterval := 2000;

  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.AttributeId := UAAttributeId_EventNotifier;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters2;
  MonitoredItemArguments2.NodeDescriptor.NodeId := ServerNodeId2;
  MonitoredItemArguments2.State := 'secondState';


  // Instantiate the client object and hook events
```

```
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers19.Create;
  Client.OnEventNotification := ClientEventHandlers.Client_EventNotification;

  Arguments := VarArrayCreate([0, 1], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;

  WriteLn('Subscribing...');
  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing monitored item changed events for 30 seconds...');
  PumpSleep(30*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;

class function THelperMethods19.ObjectTypeIds_BaseEventType: _UANodeId;
  var NodeId: _UANodeId;
  begin
    NodeId := CoUANodeId.Create;
    NodeId.StandardName := 'BaseEventType';
    Result := NodeId;
  end;

class function THelperMethods19.UAFilterElements_SimpleAttribute(TypeId: _UANodeId;
simpleRelativeBrowsePathString: string): _UASimpleAttributeOperand;
var
  BrowsePathParser: _UABrowsePathParser;
  Operand: _UASimpleAttributeOperand;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId := TypeId;
  Operand.QualifiedNames :=
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection;
  Result := Operand;
end;

class function THelperMethods19.UABaseEventObject_Operands_NodeId: _UASimpleAttributeOperand;
var
  Operand: _UASimpleAttributeOperand;
begin
  Operand := CoUASimpleAttributeOperand.Create;
  Operand.TypeId.NodeId.StandardName := 'BaseEventType';
  Operand.AttributeId := UAAttributeId_NodeId;
  Result := Operand;
end;

class function THelperMethods19.UABaseEventObject_Operands_EventId: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventId');
end;
```

```
class function THelperMethods19.UABaseEventObject_Operands_EventType:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/EventType');
end;

class function THelperMethods19.UABaseEventObject_Operands_SourceNode:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceNode');
end;

class function THelperMethods19.UABaseEventObject_Operands_SourceName:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/SourceName');
end;

class function THelperMethods19.UABaseEventObject_Operands_Time: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Time');
end;

class function THelperMethods19.UABaseEventObject_Operands_ReceiveTime:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/ReceiveTime');
end;

class function THelperMethods19.UABaseEventObject_Operands_LocalTime:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/LocalTime');
end;

class function THelperMethods19.UABaseEventObject_Operands_Message: _UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Message');
end;

class function THelperMethods19.UABaseEventObject_Operands_Severity:
_UASimpleAttributeOperand;
begin
  Result := UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, '/Severity');
end;

class function THelperMethods19.UABaseEventObject_AllFields: _UAAttributeFieldCollection;
var
  Fields: _UAAttributeFieldCollection;
begin
  Fields := CoUAAttributeFieldCollection.Create;
  Fields.Add(UABaseEventObject_Operands_NodeId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventId.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_EventType.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceNode.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_SourceName.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Time.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_LocalTime.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Message.ToUAAttributeField);
  Fields.Add(UABaseEventObject_Operands_Severity.ToUAAttributeField);
```

```
  Result := Fields;
end;
```

PHP

```
// This example shows how to subscribe to multiple events.

const UAAttributeId_NodeId = 1;
const UAAttributeId_EventNotifier = 12;

const UAFilterOperator_Equals = 1;
const UAFilterOperator_GreaterThanOrEqual = 5;

class ClientEvents {
    function EventNotification($Sender, $E)
    {
        // Display the event
        printf("%s\n", $E);
    }
}

$EndpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer";

// set MonitoredItemArguments1
// Event filter: The severity is >= 500.
$Operand11 = UABaseEventObject_Operands_Severity();
$Operand12 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand12->Value = 500;
$WhereClause1 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
$WhereClause1->FilterOperator = UAFilterOperator_GreaterThanOrEqual;
$WhereClause1->FilterOperands->Add($Operand11);
$WhereClause1->FilterOperands->Add($Operand12);

$EventFilter1 = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter1->SelectClauses = UABaseEventObject_AllFields();
$EventFilter1->WhereClause = $WhereClause1;

$ServerNodeId1 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId1->StandardName = "Server";

$MonitoringParameters1 = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters1->EventFilter = $EventFilter1;
$MonitoringParameters1->QueueSize = 1000;
$MonitoringParameters1->SamplingInterval = 1000;

$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->AttributeId = UAAttributeId_EventNotifier;
$MonitoredItemArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters1;
$MonitoredItemArguments1->NodeDescriptor->NodeId = $ServerNodeId1;
$MonitoredItemArguments1->State = "firstState";

// set MonitoredItemArguments2
// Event filter: The event comes from a specified source node.
$Operand21 = UABaseEventObject_Operands_SourceNode();
$SourceNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$SourceNodeId->ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Metals/SouthMotor";
$Operand22 = new COM("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand");
$Operand22->Value = $SourceNodeId;
$WhereClause2 = new COM("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement");
```

```
$WhereClause2->FilterOperator = UAFilterOperator_Equals;
$WhereClause2->FilterOperands->Add($Operand21);
$WhereClause2->FilterOperands->Add($Operand22);

$EventFilter2 = new COM("OpcLabs.EasyOpc.UA.UAEventFilter");
$EventFilter2->SelectClauses = UABaseEventObject_AllFields();
$EventFilter2->WhereClause = $WhereClause2;

$ServerNodeId2 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId2->StandardName = "Server";

$MonitoringParameters2 = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters2->EventFilter = $EventFilter2;
$MonitoringParameters2->QueueSize = 1000;
$MonitoringParameters2->SamplingInterval = 2000;

$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->AttributeId = UAAttributeId_EventNotifier;
$MonitoredItemArguments2->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters2;
$MonitoredItemArguments2->NodeDescriptor->NodeId = $ServerNodeId2;
$MonitoredItemArguments2->State = "secondState";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Processing monitored item changed events for 30 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 30);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);


function ObjectTypeIds_BaseEventType() {
    $NodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
    $NodeId->StandardName = "BaseEventType";
    return $NodeId;
}

function UAFilterElements_SimpleAttribute($TypeId, $simpleRelativeBrowsePathString) {
  $BrowsePathParser = new COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");
  $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
  $Operand->TypeId->NodeId = $TypeId;
  $Operand->QualifiedNames = $BrowsePathParser-
>ParseRelative($simpleRelativeBrowsePathString)->ToUAQualifiedNameCollection;
  return $Operand;
}

function UABaseEventObject_Operands_NodeId() {
    $Operand = new COM("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand");
```

```
    $Operand->TypeId->NodeId->StandardName = "BaseEventType";
    $Operand->AttributeId = UAAttributeId_NodeId;
    return $Operand;
}

function UABaseEventObject_Operands_EventId() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventId");
}

function UABaseEventObject_Operands_EventType() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/EventType");
}

function UABaseEventObject_Operands_SourceNode() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceNode");
}

function UABaseEventObject_Operands_SourceName() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/SourceName");
}

function UABaseEventObject_Operands_Time() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Time");
}

function UABaseEventObject_Operands_ReceiveTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/ReceiveTime");
}

function UABaseEventObject_Operands_LocalTime() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/LocalTime");
}

function UABaseEventObject_Operands_Message() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Message");
}

function UABaseEventObject_Operands_Severity() {
  return UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType(), "/Severity");
}


function UABaseEventObject_AllFields() {
    $Fields = new COM("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection");
    $Fields->Add(UABaseEventObject_Operands_NodeId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventId()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_EventType()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceNode()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_SourceName()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Time()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_ReceiveTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_LocalTime()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Message()->ToUAAttributeField);
    $Fields->Add(UABaseEventObject_Operands_Severity()->ToUAAttributeField);

    return $Fields;
}

// Example output (truncated):
//Subscribing...
//Processing monitored item changed events for 30 seconds...
//[firstState] Success
```

```
//[secondState] Success
//[firstState] Success; Refresh; RefreshInitiated
//[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
//[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
//[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
//[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
//[firstState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[firstState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[firstState] Success; Refresh; RefreshComplete
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:08
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:08
PM
//[secondState] Success; Refresh; RefreshInitiated
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm is active."
@11/8/2019 7:48:07 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
//[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
//[secondState] Success; Refresh; RefreshComplete
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:09
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:09
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:10
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:10
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:11
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:11
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:12
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:12
PM
//[firstState] Success; (10 field results) [EastTank] 500! "The alarm severity has increased."
@11/8/2019 7:48:13 PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:13
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:13
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:14
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:14
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:15
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:15
```

```
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:16
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:16
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:17
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:17
PM
//[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:18
PM
//[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:18
PM
//[secondState] Success; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/8/2019 7:48:18 PM
//...
```

VB.NET

```
' This example shows how to subscribe to multiple events.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.AlarmsAndConditions
Imports OpcLabs.EasyOpc.UA.Filtering
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.AlarmsAndConditions
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub Events()

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.EventNotification, AddressOf client_EventNotification

            Console.WriteLine("Subscribing...")

            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments("firstState", _
                        "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", _
                        UAObjectIds.Server, _
                        New UAMonitoringParameters(1000, New UAEventFilterBuilder( _

UAFilterElements.GreaterThanOrEqual(UABaseEventObject.Operands.Severity, 500), _
                            UABaseEventObject.AllFields))) _
                        With {.AttributeId = UAAttributeId.EventNotifier}, _
                    New EasyUAMonitoredItemArguments("secondState", _
                        "opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer", _
                        UAObjectIds.Server, _
                        New UAMonitoringParameters(2000, New UAEventFilterBuilder( _
                            UAFilterElements.Equals( _
                                UABaseEventObject.Operands.SourceNode, _
                                New
UANodeId("nsu=http://opcfoundation.org/Quickstarts/AlarmCondition;ns=2;s=1:Metals/SouthMotor")),
_
                            UABaseEventObject.AllFields))) _
                        With {.AttributeId = UAAttributeId.EventNotifier} _
```

```
                } _
            )

            Console.WriteLine("Processing event notifications for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_EventNotification(ByVal sender As Object, ByVal e As
EasyUAEventNotificationEventArgs)
            ' Display the event
            Console.WriteLine(e)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to subscribe to multiple events.

Option Explicit

Const UAAttributeId_NodeId = 1
Const UAAttributeId_EventNotifier = 12

Const UAFilterOperator_Equals = 1
Const UAFilterOperator_GreaterThanOrEqual = 5

Dim endpointDescriptor
endpointDescriptor = "opc.tcp://opcua.demo-this.com:62544/Quickstarts/AlarmConditionServer"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

Dim arguments(1)
Set arguments(0) = CreateMonitoredItemArguments1
Set arguments(1) = CreateMonitoredItemArguments2

WScript.Echo "Subscribing..."
Client.SubscribeMultipleMonitoredItems arguments

WScript.Echo "Processing monitored item changed events for 30 seconds..."
WScript.Sleep 30 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Function ObjectTypeIds_BaseEventType
    Dim NodeId: Set NodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    NodeId.StandardName = "BaseEventType"
    Set ObjectTypeIds_BaseEventType = NodeId
```

```
End Function

Function UAFilterElements_SimpleAttribute(TypeId, simpleRelativeBrowsePathString)
    Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
    Dim Operand: Set Operand =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Set Operand.TypeId.NodeId = TypeId
    Set Operand.QualifiedNames =
BrowsePathParser.ParseRelative(simpleRelativeBrowsePathString).ToUAQualifiedNameCollection
    Set UAFilterElements_SimpleAttribute = Operand
End Function

Function UABaseEventObject_Operands_NodeId
    Dim Operand: Set Operand =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UASimpleAttributeOperand")
    Operand.TypeId.NodeId.StandardName = "BaseEventType"
    Operand.AttributeId = UAAttributeId_NodeId
    Set UABaseEventObject_Operands_NodeId = Operand
End Function

Function UABaseEventObject_Operands_EventId
    Set UABaseEventObject_Operands_EventId =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventId")
End Function

Function UABaseEventObject_Operands_EventType
    Set UABaseEventObject_Operands_EventType =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/EventType")
End Function

Function UABaseEventObject_Operands_SourceNode
    Set UABaseEventObject_Operands_SourceNode =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceNode")
End Function

Function UABaseEventObject_Operands_SourceName
    Set UABaseEventObject_Operands_SourceName =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/SourceName")
End Function

Function UABaseEventObject_Operands_Time
    Set UABaseEventObject_Operands_Time =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Time")
End Function

Function UABaseEventObject_Operands_ReceiveTime
    Set UABaseEventObject_Operands_ReceiveTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/ReceiveTime")
End Function

Function UABaseEventObject_Operands_LocalTime
    Set UABaseEventObject_Operands_LocalTime =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/LocalTime")
End Function

Function UABaseEventObject_Operands_Message
    Set UABaseEventObject_Operands_Message =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Message")
End Function

Function UABaseEventObject_Operands_Severity
```

```
    Set UABaseEventObject_Operands_Severity =
UAFilterElements_SimpleAttribute(ObjectTypeIds_BaseEventType, "/Severity")
End Function

Function UABaseEventObject_AllFields
    Dim Fields: Set Fields = CreateObject("OpcLabs.EasyOpc.UA.UAAttributeFieldCollection")

    Fields.Add UABaseEventObject_Operands_NodeId.ToUAAttributeField

    Fields.Add UABaseEventObject_Operands_EventId.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_EventType.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceNode.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_SourceName.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Time.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_ReceiveTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_LocalTime.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Message.ToUAAttributeField
    Fields.Add UABaseEventObject_Operands_Severity.ToUAAttributeField

    Set UABaseEventObject_AllFields = Fields
End Function

Function CreateMonitoredItemArguments1
    ' Event filter: The severity is >= 500.
    Dim Operand1: Set Operand1 = UABaseEventObject_Operands_Severity
    Dim Operand2: Set Operand2 = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand")
    Operand2.Value = 500
    Dim WhereClause: Set WhereClause =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement")
    WhereClause.FilterOperator = UAFilterOperator_GreaterThanOrEqual
    WhereClause.FilterOperands.Add Operand1
    WhereClause.FilterOperands.Add Operand2

    Dim EventFilter: Set EventFilter = CreateObject("OpcLabs.EasyOpc.UA.UAEventFilter")
    Set EventFilter.SelectClauses = UABaseEventObject_AllFields
    Set EventFilter.WhereClause = WhereClause

    Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    ServerNodeId.StandardName = "Server"

    Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
    Set MonitoringParameters.EventFilter = EventFilter
    MonitoringParameters.QueueSize = 1000
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments: Set MonitoredItemArguments =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
    MonitoredItemArguments.AttributeId = UAAttributeId_EventNotifier
    MonitoredItemArguments.EndpointDescriptor.UrlString = endpointDescriptor
    Set MonitoredItemArguments.MonitoringParameters = MonitoringParameters
    Set MonitoredItemArguments.NodeDescriptor.NodeId = ServerNodeId
    MonitoredItemArguments.State = "firstState"

    Set CreateMonitoredItemArguments1 = MonitoredItemArguments
End Function

Function CreateMonitoredItemArguments2
    ' Event filter: The event comes from a specified source node.
    Dim Operand1: Set Operand1 = UABaseEventObject_Operands_SourceNode
    Dim SourceNodeId: Set SourceNodeId =
```

```
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    SourceNodeId.ExpandedText = "nsu=http://opcfoundation.org/Quickstarts/AlarmCondition
;ns=2;s=1:Metals/SouthMotor"
    Dim Operand2: Set Operand2 = CreateObject("OpcLabs.EasyOpc.UA.Filtering.UALiteralOperand")
    Set Operand2.Value = SourceNodeId
    Dim WhereClause: Set WhereClause =
CreateObject("OpcLabs.EasyOpc.UA.Filtering.UAContentFilterElement")
    WhereClause.FilterOperator = UAFilterOperator_Equals
    WhereClause.FilterOperands.Add Operand1
    WhereClause.FilterOperands.Add Operand2

    Dim EventFilter: Set EventFilter = CreateObject("OpcLabs.EasyOpc.UA.UAEventFilter")
    Set EventFilter.SelectClauses = UABaseEventObject_AllFields
    Set EventFilter.WhereClause = WhereClause

    Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
    ServerNodeId.StandardName = "Server"

    Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
    Set MonitoringParameters.EventFilter = EventFilter
    MonitoringParameters.QueueSize = 1000
    MonitoringParameters.SamplingInterval = 2000

    Dim MonitoredItemArguments: Set MonitoredItemArguments =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
    MonitoredItemArguments.AttributeId = UAAttributeId_EventNotifier
    MonitoredItemArguments.EndpointDescriptor.UrlString = endpointDescriptor
    Set MonitoredItemArguments.MonitoringParameters = MonitoringParameters
    Set MonitoredItemArguments.NodeDescriptor.NodeId = ServerNodeId
    MonitoredItemArguments.State = "secondState"

    Set CreateMonitoredItemArguments2 = MonitoredItemArguments
End Function

Sub Client_EventNotification(Sender, e)
    ' Display the event
    WScript.Echo e
End Sub



' Example output (truncated):
'Subscribing...
'Processing monitored item changed events for 30 seconds...
'[firstState] Success
'[secondState] Success
'[firstState] Success; Refresh; RefreshInitiated
'[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:13 PM
'[firstState] Success; Refresh; (10 field results) [EastTank] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:17 PM
'[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:02 PM
'[firstState] Success; Refresh; (10 field results) [NorthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:16 PM
'[firstState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[firstState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[firstState] Success; Refresh; RefreshComplete
```

```
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:08
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:08
PM
'[secondState] Success; Refresh; RefreshInitiated
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The dialog was
activated" @9/10/2019 8:08:25 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm is active."
@11/8/2019 7:48:07 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 700! "The alarm was
acknoweledged." @10/14/2019 4:00:21 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 500! "The alarm was
acknoweledged." @10/14/2019 4:00:03 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:02 PM
'[secondState] Success; Refresh; (10 field results) [SouthMotor] 100! "The alarm severity has
increased." @9/10/2019 8:09:59 PM
'[secondState] Success; Refresh; RefreshComplete
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:09
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:09
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:10
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:10
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:11
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:11
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:12
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:12
PM
'[firstState] Success; (10 field results) [EastTank] 500! "The alarm severity has increased."
@11/8/2019 7:48:13 PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:13
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:13
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:14
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:14
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:15
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:15
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:16
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:16
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:17
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:17
PM
'[firstState] Success; (10 field results) [Internal] 500! "Raising Events" @11/8/2019 7:48:18
PM
'[firstState] Success; (10 field results) [Internal] 500! "Events Raised" @11/8/2019 7:48:18
PM
'[secondState] Success; (10 field results) [SouthMotor] 300! "The alarm severity has
increased." @11/8/2019 7:48:18 PM
```

```
'...
```

13.2.7 Examples - OPC UA Application

13.2.7.1 Examples - OPC UA Application - Find all our registrations in GDS

C#

```
// Shows how to find all application's registrations in the GDS.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class FindGdsRegistrations
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Find all application's registrations in the GDS.
            IReadOnlyDictionary<UANodeId, UAApplicationElement>
applicationElementDictionary;
            try
            {
                applicationElementDictionary =
clientApplication.FindGdsRegistrations(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
```

```
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (KeyValuePair<UANodeId, UAApplicationElement> pair in
applicationElementDictionary)
            {
                Console.WriteLine();
                Console.WriteLine("Application ID: {0}", pair.Key);

                UAApplicationElement applicationElement = pair.Value;
                Console.WriteLine("Application element: {0}", applicationElement);
                Console.WriteLine("  Application URI string: {0}",
applicationElement.ApplicationUriString);
                Console.WriteLine("  Discovery URI strings: {0}",
applicationElement.DiscoveryUriStrings);
                Console.WriteLine("  Product URI string: {0}",
applicationElement.ProductUriString);
            }
        }
    }
}
```

Object Pascal

```
// Shows how to find all application's registrations in the GDS.

class procedure FindGdsRegistrations.Main;
var
  ApplicationElement: _UAApplicationElement;
  ApplicationElementDictionary: _UANodeIdUAApplicationElementReadOnlyDictionary;
  ApplicationId: _UANodeId;
  Count: Cardinal;
  Element: OleVariant;
  Enumerator: IEnumVARIANT;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);
```

```
  // Find all application's registrations in the GDS.
  try
    ApplicationElementDictionary :=
ClientApplication.FindGdsRegistrations(GdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  Enumerator := ApplicationElementDictionary.GetEnumerator;
  while (Enumerator.Next(1, Element, Count) = S_OK) do
  begin
    WriteLn;
    ApplicationId := IUnknown(Element.Key) as _UANodeId;
    WriteLn('Application ID: ', ApplicationId.ToString);

    ApplicationElement := IUnknown(Element.Value) as _UAApplicationElement;
    WriteLn('  Application URI string: ', ApplicationElement.ApplicationUriString);
    WriteLn('  Discovery URI string: ', ApplicationElement.DiscoveryUriString);
    WriteLn('  Product URI string: ', ApplicationElement.ProductUriString);
  end;
end;
```

VB.NET

```
' Shows how to find all application's registrations in the GDS.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class FindGdsRegistrations
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Find all application's registrations in the GDS.
            Dim applicationElementDictionary As IReadOnlyDictionary(Of UANodeId,
UAApplicationElement)
```

```
            Try
                applicationElementDictionary =
clientApplication.FindGdsRegistrations(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each pair In applicationElementDictionary
                Console.WriteLine()
                Console.WriteLine("Application ID: {0}", pair.Key)

                Dim applicationElement = pair.Value
                Console.WriteLine("Application element: {0}", applicationElement)
                Console.WriteLine("  Application URI string: {0}",
applicationElement.ApplicationUriString)
                Console.WriteLine("  Discovery URI strings: {0}",
applicationElement.DiscoveryUriStrings)
                Console.WriteLine("  Product URI string: {0}",
applicationElement.ProductUriString)
            Next pair
        End Sub
    End Class
End Namespace
```

13.2.7.2 Examples - OPC UA Application - Get certificate subject name

C#

```
// Shows how to get the subject name of application certificates.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class GetCertificateSubjectName
    {
        public static void Main1()
        {
            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Get the subject name of application certificates.
            string certificateSubjectName = clientApplication.GetCertificateSubjectName();

            // Display results
            Console.WriteLine("Certificate subject name: {0}", certificateSubjectName);
        }
    }
```

```
}
```

Object Pascal

```
// Shows how to get the subject name of application certificates.

class procedure GetCertificateSubjectName.Main;
var
  CertificateSubjectName: string;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
begin
  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Get the subject name of application certificates.
  CertificateSubjectName := ClientApplication.GetCertificateSubjectName();

  // Display results
  WriteLn('Certificate subject name: ', CertificateSubjectName);
end;
```

VB.NET

```
' Shows how to get the subject name of application certificates.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class GetCertificateSubjectName
        Public Shared Sub Main1()
            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Get the subject name of application certificates.
            Dim certificateSubjectName = clientApplication.GetCertificateSubjectName()

            ' Display results
            Console.WriteLine("Certificate subject name: {0}", certificateSubjectName)
        End Sub
    End Class
End Namespace
```

13.2.7.3 Examples - OPC UA Application - Get registration information

C#

```
// Shows how to get the OPC UA registration information for this application.
```

```
using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Discovery;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class GetApplicationElement
    {
        public static void Main1()
        {
            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Get the OPC UA registration information for this application.
            UAApplicationElement applicationElement =
clientApplication.GetApplicationElement();

            // Display results
            Console.WriteLine("Application element:");
            Console.WriteLine("  Application name: {0}",
applicationElement.ApplicationName);
            Console.WriteLine("  Application type: {0}",
applicationElement.ApplicationType);
            Console.WriteLine("  Application URI string: {0}",
applicationElement.ApplicationUriString);
            Console.WriteLine("  Discovery URI strings: {0}",
applicationElement.DiscoveryUriStrings);
            Console.WriteLine("  Product URI string: {0}",
applicationElement.ProductUriString);
        }
    }
}
```

Object Pascal

```
// Shows how to get the OPC UA registration information for this application.

class procedure GetApplicationElement.Main;
var
  ApplicationElement: _UAApplicationElement;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
begin
  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Get the OPC UA registration information for this application.
  ApplicationElement := ClientApplication.GetApplicationElement;

  // Display results
  WriteLn('Application element:');
```

```
  WriteLn('  Application name: ', ApplicationElement.ApplicationName);
  WriteLn('  Application type: ', ApplicationElement.ApplicationType);
  WriteLn('  Application URI string: ', ApplicationElement.ApplicationUriString);
  WriteLn('  Discovery URI string: ', ApplicationElement.DiscoveryUriString);
  WriteLn('  Product URI string: ', ApplicationElement.ProductUriString);
end;
```

VB.NET

```
' Shows how to get the OPC UA registration information for this application.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class GetApplicationElement
        Public Shared Sub Main1()
            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Get the OPC UA registration information for this application.
            Dim applicationElement = clientApplication.GetApplicationElement()

            ' Display results
            Console.WriteLine("Application element:")
            Console.WriteLine("  Application name: {0}",
applicationElement.ApplicationName)
            Console.WriteLine("  Application type: {0}",
applicationElement.ApplicationType)
            Console.WriteLine("  Application URI string: {0}",
applicationElement.ApplicationUriString)
            Console.WriteLine("  Discovery URI strings: {0}",
applicationElement.DiscoveryUriStrings)
            Console.WriteLine("  Product URI string: {0}",
applicationElement.ProductUriString)
        End Sub
    End Class
End Namespace
```

13.2.7.4 Examples - OPC UA Application - Obtain new certificate from GDS

C#

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.Security.Cryptography.PkiCertificates;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Application.Extensions;
using OpcLabs.EasyOpc.UA.Extensions;
```

```
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    partial class ObtainNewCertificate
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.
            PkiCertificate certificate;
            try
            {
                certificate =
clientApplication.ObtainNewCertificate(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Certificate: {0}", certificate);
        }
    }
}
```

Object Pascal

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
// and store it for subsequent usage.

class procedure ObtainNewCertificate.Main;
var
  ApplicationElement: _UAApplicationElement;
  Certificate: _X509Certificate;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  Parameters: _UAObtainNewCertificateParameters;
begin
  // Define which GDS we will work with.
```

```
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  ApplicationElement := ClientApplication.GetApplicationElement;
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Obtain a new application certificate from the certificate manager (GDS), and store it
for subsequent usage.
  Parameters := CoUAObtainNewCertificateParameters.Create;

  try
    Certificate := ClientApplication.ObtainNewCertificate(GdsEndpointDescriptor,
Parameters);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Certificate: ', (Certificate as _PKICertificate).ToString);
end;
```

PHP

```
// Shows how to obtain a new application certificate from the certificate manager (GDS),
// and store it for subsequent usage.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
$ApplicationElement = $ClientApplication->GetApplicationElement;
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

// Obtain a new application certificate from the certificate manager (GDS), and store it
```

```
for subsequent usage.
$Parameters = new COM("OpcLabs.EasyOpc.UA.Application.UAObtainNewCertificateParameters");

try
{
    $Certificate = $ClientApplication->ObtainNewCertificate($GdsEndpointDescriptor,
$Parameters);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("Certificate: %s\n", $Certificate);
```

VB.NET

```
' Shows how to obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.Security.Cryptography.PkiCertificates
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Application.Extensions
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Partial Friend Class ObtainNewCertificate
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Obtain a new application certificate from the certificate manager (GDS), and
store it for subsequent usage.
            Dim certificate As PkiCertificate
            Try
                certificate = clientApplication.ObtainNewCertificate(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
```

```
            Console.WriteLine("Certificate: {0}", certificate)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage.

Option Explicit

' Define which GDS we will work with.
Dim GdsEndpointDescriptor: Set GdsEndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
GdsEndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName = "appadmin"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password = "demo"

' Obtain the client application service.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ClientApplication: Set ClientApplication =
Client.GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA")

' Display which application we are about to work with.
Dim ApplicationElement: Set ApplicationElement = ClientApplication.GetApplicationElement
WScript.Echo "Application URI string: " &
ClientApplication.GetApplicationElement.ApplicationUriString

Rem Obtain a new application certificate from the certificate manager (GDS), and store it
for subsequent usage.
Dim Parameters: Set Parameters =
CreateObject("OpcLabs.EasyOpc.UA.Application.UAObtainNewCertificateParameters")
On Error Resume Next
Dim Certificate: Set Certificate =
ClientApplication.ObtainNewCertificate(GdsEndpointDescriptor, Parameters)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Certificate: " & Certificate
```

13.2.7.5 Examples - OPC UA Application - Obtain new certificate from GDS, report progress

C#

```
// Shows how to obtain a new application certificate from the certificate manager
(GDS), and store it for subsequent usage,
// with progress reporting.
```

```
using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.Security.Cryptography.PkiCertificates;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Application.Extensions;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    partial class ObtainNewCertificate
    {
        public static void Progress()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage.
            PkiCertificate certificate;
            try
            {
                certificate =
clientApplication.ObtainNewCertificate(gdsEndpointDescriptor,
                    new Progress<string>(s => Console.WriteLine("Progress: {0}", s)));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Certificate: {0}", certificate);
        }
    }
}
```

VB.NET

```
' Shows how to obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage,
```

```
' with progress reporting.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.Security.Cryptography.PkiCertificates
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Application.Extensions
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Partial Friend Class ObtainNewCertificate
        Public Shared Sub Progress()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Obtain a new application certificate from the certificate manager (GDS),
and store it for subsequent usage.
            Dim certificate As PkiCertificate
            Try
                certificate =
clientApplication.ObtainNewCertificate(gdsEndpointDescriptor,
                    New Progress(Of String)(Sub(s) Console.WriteLine("Progress: {0}",
s)))
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Certificate: {0}", certificate)
        End Sub
    End Class
End Namespace
```

13.2.7.6 Examples - OPC UA Application - Refresh trust lists from GDS

C#

```
// Shows how to refresh own certificate stores using current trust lists for the
```

```
application from the certificate manager.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class RefreshTrustLists
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Refresh own certificate stores using current trust lists for the application
from the certificate manager.
            UATrustListMasks refreshedTrustLists;
            try
            {
                refreshedTrustLists =
clientApplication.RefreshTrustLists(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Refreshed trust lists: {0}", refreshedTrustLists);
        }
    }
}
```

Object Pascal

```
// Shows how to refresh own certificate stores using current trust lists
// for the application from the certificate manager.

class procedure RefreshTrustLists.Main;
var
```

```
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  RefreshedTrustLists: UATrustListMasks;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Refresh own certificate stores using current trust lists for the application from the
certificate manager.
  try
    RefreshedTrustLists := clientApplication.RefreshTrustLists(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('Refreshed trust lists: ', RefreshedTrustLists);
end;
```

PHP

```
// Shows how to refresh own certificate stores using current trust lists
// for the application from the certificate manager.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);
```

```
// Refresh own certificate stores using current trust lists for the application from the
certificate manager.
try
{
    $RefreshedTrustLists = $ClientApplication->RefreshTrustLists($GdsEndpointDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("Refreshed trust lists: %s\n", $RefreshedTrustLists);
```

VB.NET

```
' Shows how to refresh own certificate stores using current trust lists for the application
from the certificate manager.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class RefreshTrustLists
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Refresh own certificate stores using current trust lists for the application
from the certificate manager.
            Dim refreshedTrustLists As UATrustListMasks
            Try
                refreshedTrustLists =
clientApplication.RefreshTrustLists(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Refreshed trust lists: {0}", refreshedTrustLists)
```

```
        End Sub
    End Class
End Namespace
```

13.2.7.7 Examples - OPC UA Application - Register to GDS

C#

```
// Shows how to create an application registration in the GDS, assigning it a new
application ID.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class RegisterToGds
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Create an application registration in the GDS, assigning it a new
application ID.
            UANodeId applicationId;
            try
            {
                applicationId = clientApplication.RegisterToGds(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Application ID: {0}", applicationId);
        }
```

```
    }
}
```

Object Pascal

```
// Shows how to create an application registration in the GDS, assigning it a new
application ID.

class procedure RegisterToGds.Main;
var
  ApplicationId: _UANodeId;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Create an application registration in the GDS, assigning it a new application ID.
  try
    ApplicationId := ClientApplication.RegisterToGds(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Application ID: ', ApplicationId.ToString);
end;
```

PHP

```
// Shows how to create an application registration in the GDS, assigning it a new
application ID.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
```

```
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

// Create an application registration in the GDS, assigning it a new application ID.
try
{
    $ApplicationId = $ClientApplication->RegisterToGds($GdsEndpointDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("Application ID: %s\n", $ApplicationId);
```

VB.NET

```
' Shows how to create an application registration in the GDS, assigning it a new
application ID.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class RegisterToGds
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Create an application registration in the GDS, assigning it a new application
ID.
            Dim applicationId As UANodeId
            Try
                applicationId = clientApplication.RegisterToGds(gdsEndpointDescriptor)
            Catch uaException As UAException
```

```
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Application ID: {0}", applicationId)
        End Sub
    End Class
End Namespace
```

13.2.7.8 Examples - OPC UA Application - Unregister from GDS

C#

```
// Shows how to delete an application registration from the GDS.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class UnregisterFromGds
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Delete an application registration from the GDS.
            try
            {
                clientApplication.UnregisterFromGds(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
        }
    }
```

```
}
```

Object Pascal

```
// Shows how to delete an application registration from the GDS.

class procedure UnregisterFromGds.Main;
var
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Delete an application registration from the GDS.
  try
    ClientApplication.UnregisterFromGds(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;
end;
```

PHP

```
// Shows how to delete an application registration from the GDS.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
```

```
>ApplicationUriString);

  // Delete an application registration from the GDS.
try
{
    $ClientApplication->UnregisterFromGds($GdsEndpointDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}
```

VB.NET

```
' Shows how to delete an application registration from the GDS.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class UnregisterFromGds
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Delete an application registration from the GDS.
            Try
                clientApplication.UnregisterFromGds(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

13.2.7.9 Examples - OPC UA Application - Update GDS registration

C#

```
// Shows how to update an application registration in the GDS, keeping its application ID
if possible.

using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Application._IEasyUAClientApplication
{
    class UpdateGdsRegistration
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Obtain the client application service.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();

            // Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString);

            // Update an application registration in the GDS, keeping its application ID if
possible.
            UANodeId applicationId;
            try
            {
                applicationId =
clientApplication.UpdateGdsRegistration(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Application ID: {0}", applicationId);
        }
    }
}
```

Object Pascal

```
// Shows how to update an application registration in the GDS, keeping its application ID
```

```
if possible.

class procedure UpdateGdsRegistration.Main;
var
  ApplicationId: _UANodeId;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Obtain the client application service.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Display which application we are about to work with.
  WriteLn('Application URI string: ',
ClientApplication.GetApplicationElement.ApplicationUriString);

  // Update an application registration in the GDS, keeping its application ID if possible.
  try
    ApplicationId := ClientApplication.UpdateGdsRegistration(gdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Application ID: ', ApplicationId.ToString);
end;
```

PHP

```
// Shows how to update an application registration in the GDS, keeping its application ID
if possible.

// Define which GDS we will work with.
$GdsEndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$GdsEndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->UserName = "appadmin";
$GdsEndpointDescriptor->UserIdentity->UserNameTokenInfo->Password = "demo";

// Obtain the client application service.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientApplication = $Client-
>GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA");

// Display which application we are about to work with.
```

```
printf("Application URI string: %s\n", $ClientApplication->GetApplicationElement-
>ApplicationUriString);

// Update an application registration in the GDS, keeping its application ID if possible.
try
{
    $ApplicationId = $ClientApplication->UpdateGdsRegistration($GdsEndpointDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("Application ID: %s\n", $ApplicationId);
```

VB.NET

```
' Shows how to update an application registration in the GDS, keeping its application ID if
possible.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Application._IEasyUAClientApplication
    Friend Class UpdateGdsRegistration
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Obtain the client application service.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Display which application we are about to work with.
            Console.WriteLine("Application URI string: {0}",
                clientApplication.GetApplicationElement().ApplicationUriString)

            ' Update an application registration in the GDS, keeping its application ID if
possible.
            Dim applicationId As UANodeId
            Try
                applicationId =
clientApplication.UpdateGdsRegistration(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
```

```
            Console.WriteLine("Application ID: {0}", applicationId)
        End Sub
    End Class
End Namespace
```

13.2.8 Examples - OPC UA Complex Data

13.2.8.1 Examples - OPC UA Complex Data - Create a data type

C#

```
StructuredDataType structuredDataType = new StructuredDataType(
        "ControllerConfiguration",

"urn:UnifiedAutomation:CppDemoServer:BuildingAutomation:ControllerConfiguration")
    {
        new DataField("Name", UAOpcBinaryStandardDataTypes.CharArray),
        new DataField("DeviceAddress", UAOpcBinaryStandardDataTypes.UInt32),
        new DataField("TemperatureSetpoint", UAOpcBinaryStandardDataTypes.Double)
    };
```

Notice the use of pre-defined primitive types from the UAOpcBinaryStandardDataTypes Class. It is common to encounter references to standard data types from OPC Binary data type system, and the OPC UA Complex Data extension gives you this class so that you can refer to them easily.

13.2.8.2 Examples - OPC UA Complex Data - Create generic data

The example below shows how different kinds of generic data can be created.

C#

```
// This example shows different ways of constructing generic data.

using System;
using System.Collections;
using OpcLabs.BaseLib.DataTypeModel;

namespace UADocExamples.ComplexData._GenericData
{
    class _Construction
    {
        public static void Main1()
        {
            // Create enumeration data with value of 1.
            var enumerationData = new EnumerationData(1);
```

```
            Console.WriteLine(enumerationData);

            // Create opaque data from an array of 2 bytes, specifying its size as 15
bits.
            var opaqueData1 = new OpaqueData(new byte[] {0xAA, 0x55}, sizeInBits:15);
            Console.WriteLine(opaqueData1);

            // Create opaque data from a bit array.
            var opaqueData2 = new OpaqueData(new BitArray(new[] { false, true, false,
true, false }));
            Console.WriteLine(opaqueData2);

            // Create primitive data with System.Double value of 180.0.
            var primitiveData1 = new PrimitiveData(180.0d);
            Console.WriteLine(primitiveData1);

            // Create primitive data with System.String value.
            var primitiveData2 = new PrimitiveData("Temperature is too high!");
            Console.WriteLine(primitiveData2);

            // Create sequence data with two elements, using collection initializer
syntax.
            var sequenceData1 = new SequenceData
            {
                opaqueData1,
                opaqueData2
            };
            Console.WriteLine(sequenceData1);

            // Create the same sequence data, using the Add method.
            var sequenceData2 = new SequenceData();
            sequenceData2.Elements.Add(opaqueData1);
            sequenceData2.Elements.Add(opaqueData2);
            Console.WriteLine(sequenceData2);

            // Create the same sequence data, using an array (an enumerable) of its
elements.
            var sequenceData3 = new SequenceData(
                new GenericDataCollection(new[] {opaqueData1, opaqueData2}));
            Console.WriteLine(sequenceData3);

            // Create structured data with two members, using collection initializer
syntax.
            var structuredData1 = new StructuredData
            {
                {"Message", primitiveData2},
                {"Status", enumerationData}
            };
            Console.WriteLine(structuredData1);

            // Create the same structured data using the Add method.
            var structuredData2 = new StructuredData();
            structuredData2.Add("Message", primitiveData2);
            structuredData2.Add("Status", enumerationData);
            Console.WriteLine(structuredData2);

            // Create union data.
```

```
            var unionData1 = new UnionData("DoubleField", primitiveData1);
            Console.WriteLine(unionData1);
        }
    }
}
```

Object Pascal

```
// This example shows different ways of constructing generic data.

class procedure _Construction.Main;
var
  ByteArray1, ByteArray2: OleVariant;
  EnumerationData: _EnumerationData;
  OpaqueData1, OpaqueData2: _OpaqueData;
  PrimitiveData1, PrimitiveData2: _PrimitiveData;
  SequenceData1: _SequenceData;
  StructuredData1: _StructuredData;
begin
  // Create enumeration data with value of 1.
  EnumerationData := CoEnumerationData.Create;
  EnumerationData.Value := 1;
  WriteLn(EnumerationData.ToString);

  // Create opaque data from an array of 2 bytes, specifying its size as 15 bits.
  OpaqueData1 := CoOpaqueData.Create;
  ByteArray1 := VarArrayCreate([0, 1], varVariant);
  ByteArray1[0] := $AA;
  ByteArray1[1] := $55;
  OpaqueData1.SetByteArrayValue(ByteArray1, 15);
  WriteLn(OpaqueData1.ToString);
  // Create opaque data from an array of 1 bytes, specifying its size as 5 bits.
  OpaqueData2 := CoOpaqueData.Create;
  ByteArray2 := VarArrayCreate([0, 0], varVariant);
  ByteArray2[0] := $A;
  OpaqueData2.SetByteArrayValue(ByteArray2, 5);
  WriteLn(OpaqueData2.ToString);

  // Create primitive data with System.Double value of 180.0.
  PrimitiveData1 := CoPrimitiveData.Create;
  PrimitiveData1.Value := 180.0;
  WriteLn(PrimitiveData1.ToString);

  // Create primitive data with System.String value.
  PrimitiveData2 := CoPrimitiveData.Create;
  PrimitiveData2.Value := 'Temperature is too high!';
  WriteLn(PrimitiveData2.ToString);

  // Create sequence data with two elements, using the Add method.
  SequenceData1 := CoSequenceData.Create;
  SequenceData1.Elements.Add(OpaqueData1);
  SequenceData1.Elements.Add(OpaqueData2);
  WriteLn(SequenceData1.ToString);

  // Create structured data with two members, using the Add method.
  StructuredData1 := CoStructuredData.Create;
  StructuredData1.Add('Message', PrimitiveData2);
```

```
  StructuredData1.Add('Status', EnumerationData);
  WriteLn(StructuredData1.ToString);
end;
```

PHP

```
// This example shows different ways of constructing generic data.

// Create enumeration data with value of 1.
$EnumerationData = new COM("OpcLabs.BaseLib.DataTypeModel.EnumerationData");
$EnumerationData->Value = 1;
printf("%s\n", $EnumerationData);

// Create opaque data from an array of 2 bytes, specifying its size as 15 bits.
$OpaqueData1 = new COM("OpcLabs.BaseLib.DataTypeModel.OpaqueData");
$ByteArray1[0] = 0xAA;
$ByteArray1[1] = 0x55;
$OpaqueData1->SetByteArrayValue($ByteArray1, 15);
printf("%s\n", $OpaqueData1);

// Create opaque data from a bit array.
$OpaqueData2 = new COM("OpcLabs.BaseLib.DataTypeModel.OpaqueData");
$BitArray = $OpaqueData2->Value;
$BitArray->Length = 5;
$BitArray->Set(0, false);
$BitArray->Set(1, true);
$BitArray->Set(2, false);
$BitArray->Set(3, true);
$BitArray->Set(4, false);
printf("%s\n", $OpaqueData2);

// Create primitive data with System.Double value of 180.0.
$PrimitiveData1 = new COM("OpcLabs.BaseLib.DataTypeModel.PrimitiveData");
$PrimitiveData1->Value = 180.0;
printf("%s\n", $PrimitiveData1);

// Create primitive data with System.String value.
$PrimitiveData2 = new COM("OpcLabs.BaseLib.DataTypeModel.PrimitiveData");
$PrimitiveData2->Value = "Temperature is too high!";
printf("%s\n", $PrimitiveData2);

// Create sequence data with two elements, using the Add method.
$SequenceData1 = new COM("OpcLabs.BaseLib.DataTypeModel.SequenceData");
$SequenceData1->Elements->Add($OpaqueData1);
$SequenceData1->Elements->Add($OpaqueData2);
printf("%s\n", $SequenceData1);

// Create structured data with two members, using the Add method.
$StructuredData1 = new COM("OpcLabs.BaseLib.DataTypeModel.StructuredData");
$StructuredData1->Add("Message", $PrimitiveData2);
$StructuredData1->Add("Status", $EnumerationData);
printf("%s\n", $StructuredData1);
```

VB.NET

```
' This example shows different ways of constructing generic data.
```

```
Imports System
Imports System.Collections
Imports OpcLabs.BaseLib.DataTypeModel

Namespace UADocExamples.ComplexData._GenericData

    Friend Class _Construction

        Public Shared Sub Main1()
            ' Create enumeration data with value of 1.
            Dim enumerationData = New EnumerationData(1)
            Console.WriteLine(enumerationData)

            ' Create opaque data from an array of 2 bytes, specifying its size as 15
bits.
            Dim opaqueData1 = New OpaqueData(New Byte() {170, 85}, sizeInBits:=15)
            Console.WriteLine(opaqueData1)

            ' Create opaque data from a bit array.
            Dim opaqueData2 = New OpaqueData(New BitArray(New Boolean() {False, True,
False, True, False}))
            Console.WriteLine(opaqueData2)

            ' Create primitive data with System.Double value of 180.0.
            Dim primitiveData1 = New PrimitiveData(180)
            Console.WriteLine(primitiveData1)

            ' Create primitive data with System.String value.
            Dim primitiveData2 = New PrimitiveData("Temperature is too high!")
            Console.WriteLine(primitiveData2)

            ' Create sequence data with two elements, using collection initializer
syntax.
            Dim sequenceData1 = New SequenceData() From {opaqueData1, opaqueData2}
            Console.WriteLine(sequenceData1)

            ' Create the same sequence data, using the Add method.
            Dim sequenceData2 = New SequenceData
            sequenceData2.Elements.Add(opaqueData1)
            sequenceData2.Elements.Add(opaqueData2)
            Console.WriteLine(sequenceData2)

            ' Create the same sequence data, using an array (an enumerable) of its
elements.
            Dim sequenceData3 = New SequenceData(New GenericDataCollection(New
OpaqueData() {opaqueData1, opaqueData2}))
            Console.WriteLine(sequenceData3)

            ' Create structured data with two members, using collection initializer
syntax.
            Dim structuredData1 = New StructuredData() From { _
                {"Message", primitiveData2}, _
                {"Status", enumerationData}}
            Console.WriteLine(structuredData1)

            ' Create the same structured data using the Add method.
            Dim structuredData2 = New StructuredData()
```

```
            structuredData2.Add("Message", primitiveData2)
            structuredData2.Add("Status", enumerationData)
            Console.WriteLine(structuredData2)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows different ways of constructing generic data.

Option Explicit

' Create enumeration data with value of 1.
Dim EnumerationData: Set EnumerationData =
CreateObject("OpcLabs.BaseLib.DataTypeModel.EnumerationData")
EnumerationData.Value = 1
WScript.Echo EnumerationData

' Create opaque data from an array of 2 bytes, specifying its size as 15 bits.
Dim OpaqueData1: Set OpaqueData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.OpaqueData")
OpaqueData1.SetByteArrayValue Array(&HAA, &H55), 15
WScript.Echo OpaqueData1

' Create opaque data from a bit array.
Dim OpaqueData2: Set OpaqueData2 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.OpaqueData")
Dim BitArray: Set BitArray = OpaqueData2.Value
BitArray.Length = 5
BitArray(0) = False
BitArray(1) = True
BitArray(2) = False
BitArray(3) = True
BitArray(4) = False
WScript.Echo OpaqueData2

' Create primitive data with System.Double value of 180.0.
Dim PrimitiveData1: Set PrimitiveData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.PrimitiveData")
PrimitiveData1.Value = 180.0
WScript.Echo PrimitiveData1

' Create primitive data with System.String value.
Dim PrimitiveData2: Set PrimitiveData2 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.PrimitiveData")
PrimitiveData2.Value = "Temperature is too high!"
WScript.Echo PrimitiveData2

' Create sequence data with two elements, using the Add method.
Dim SequenceData1: Set SequenceData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.SequenceData")
SequenceData1.Elements.Add OpaqueData1
SequenceData1.Elements.Add OpaqueData2
WScript.Echo SequenceData1

' Create structured data with two members, using the Add method.
```

```
Dim StructuredData1: Set StructuredData1 =
CreateObject("OpcLabs.BaseLib.DataTypeModel.StructuredData")
StructuredData1.Add "Message", PrimitiveData2
StructuredData1.Add "Status", EnumerationData
WScript.Echo StructuredData1
```

13.2.8.3 Examples - OPC UA Complex Data - Disable and Enable

C#

```
// Shows how to disable and enable the OPC UA Complex Data plug-in.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._PluginSetup
{
    class Enabled
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue


            // We will explicitly disable the Complex Data plug-in, and read a node
which returns complex data. We will
            // receive an object of type UAExtensionObject, which contains the encoded
data in its binary form. In this
            // form, the data cannot be easily further processed by your application.
            //
            // Disabling the Complex Data plug-in may be useful e.g. for licensing
reasons (when the product edition you
            // have does not support the Complex Data plug-in, and you want to avoid
the associated error), or for
            // performance reasons (if you do not need the internal content of the
value, for example if your code just
            // needs to take the value read, and write it elsewhere).

            var client1 = new EasyUAClient();
            client1.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
false;
```

```
            object value1;
            try
            {
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine(value1);


            // Now we will read the same value, but with the Complex Data plug-in
enabled. This time we will receive an
            // object of type UAGenericObject, which contains the data in the decoded
form, accessible for further
            // processing by your application.
            //
            // Note that it is not necessary to explicitly enable the Complex Data
plug-in like this, because it is enabled
            // by default.

            var client2 = new EasyUAClient();
            client2.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
true;

            object value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor);
            Console.WriteLine(value2);


            // Example output:
            //
            // Binary Byte[1373]; {nsu=http://test.org/UA/Data/;i=11437}
            // (ScalarValueDataType) structured

            // On the first line, the type and length of the encoded data is shown, and
the node ID is the encoding ID.
            // On the 2nd line, the kind (structured) and the name of the complex data
type (ScalarValueDataType) is shown.
        }
    }
}
```

Object Pascal

```
// Shows how to disable and enable the OPC UA Complex Data plug-in.

class procedure Enabled.Main;
var
  Client1, Client2: _EasyUAClient;
  EndpointDescriptor: string;
  NodeDescriptor: string;
  Value1, Value2: OleVariant;
begin
  // Define which server and node we will work with.
```

```
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // We will explicitly disable the Complex Data plug-in, and read a node which returns
complex data. We will
  // receive an object of type UAExtensionObject, which contains the encoded data in
its binary form. In this
  // form, the data cannot be easily further processed by your application.
  //
  // Disabling the Complex Data plug-in may be useful e.g. for licensing reasons (when
the product edition you
  // have does not support the Complex Data plug-in, and you want to avoid the
associated error), or for
  // performance reasons (if you do not need the internal content of the value, for
example if your code just
  // needs to take the value read, and write it elsewhere).
  Client1 := CoEasyUAClient.Create;
  Client1.InstanceParameters.PluginSetups.FindName('UAComplexData').Enabled := false;

  try
    Value1 := Client1.ReadValue(EndpointDescriptor, NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
  WriteLn(Value1);

  // Now we will read the same value, but with the Complex Data plug-in enabled. This
time we will receive an
  // object of type UAGenericObject, which contains the data in the decoded form,
accessible for further
  // processing by your application.
  //
  // Note that it is not necessary to explicitly enable the Complex Data plug-in like
this, because it is enabled
  // by default.

  Client2 := CoEasyUAClient.Create;
  Client2.InstanceParameters.PluginSetups.FindName('UAComplexData').Enabled := true;

  Value2 := Client2.ReadValue(EndpointDescriptor, NodeDescriptor);
  WriteLn(Value2);

  // Example output:
  //
  // Binary Byte[1373]; {nsu=http://test.org/UA/Data/;i=11437}
  // (ScalarValueDataType) structured

  // On the first line, the type and length of the encoded data is shown, and the node
ID is the encoding ID.
```

```
  // On the 2nd line, the kind (structured) and the name of the complex data type
(ScalarValueDataType) is shown.

end;
```

VB.NET

```
' Shows how to disable and enable the OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._PluginSetup

    Friend Class Enabled

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue


            ' We will explicitly disable the Complex Data plug-in, and read a node
which returns complex data. We will
            ' receive an object of type UAExtensionObject, which contains the encoded
data in its binary form. In this
            ' form, the data cannot be easily further processed by your application.
            '
            ' Disabling the Complex Data plug-in may be useful e.g. for licensing
reasons (when the product edition you
            ' have does not support the Complex Data plug-in, and you want to avoid the
associated error), or for
            ' performance reasons (if you do not need the internal content of the
value, for example if your code just
            ' needs to take the value read, and write it elsewhere).

            Dim client1 = New EasyUAClient
            client1.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
False

            Dim value1 As Object
            Try
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
```

```
            End Try
            Console.WriteLine(value1)


            ' Now we will read the same value, but with the Complex Data plug-in
enabled. This time we will receive an
            ' object of type UAGenericObject, which contains the data in the decoded
form, accessible for further
            ' processing by your application.
            '
            ' Note that it is not necessary to explicitly enable the Complex Data plug-
in like this, because it is enabled
            ' by default.

            Dim client2 = New EasyUAClient
            client2.InstanceParameters.PluginSetups.FindName("UAComplexData").Enabled =
True

            Dim value2 As Object
            Try
                value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value2)


            ' Example output:
            '
            ' Binary Byte[1373]; {nsu=http://test.org/UA/Data/;i=11437}
            ' (ScalarValueDataType) structured

            ' On the first line, the type and length of the encoded data is shown, and
the node ID is the encoding ID.
            ' On the 2nd line, the kind (structured) and the name of the complex data
type (ScalarValueDataType) is shown.
        End Sub
    End Class
End Namespace
```

13.2.8.4 Examples - OPC UA Complex Data - Obtain content of data type dictionary

C#

```
// Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
// content of the data type dictionary.

using System;
using System.Text;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.OperationModel.Generic;
```

```
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.DataTypeModel;
using OpcLabs.EasyOpc.UA.DataTypeModel.Extensions;
using OpcLabs.EasyOpc.UA.InformationModel;
using OpcLabs.EasyOpc.UA.Plugins.ComplexData;

namespace UADocExamples.ComplexData._IUADataTypeDictionaryProvider
{
    class ResolveDataTypeDescriptorFromDataTypeEncodingId
    {
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Obtain the data type ID.
            //
            // In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
            // attribute, or easier, by calling the extension method ReadDataType on the IEasyUAClient interface.
The sample
            // server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type,
            // and the actual values are then sub-types of this base data type. This abstract data type does not
have any
            // encodings associated with it and it is therefore not possible to extract its description from the
server.
            // We therefore use a hard-coded data type ID for one of the sub-types in this example.
            //
            // The code to obtain the data type ID for given node would normally look like this:
            //    UANodeId dataTypeId = client.ReadDataType(
            //        endpointUriString,
            //        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            //
            UANodeId dataTypeId = "nsu=http://test.org/UA/Data/;i=9440";    // ScalarValueDataType

            // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
            // operations.
            IEasyUAClientComplexData complexData = client.GetService<IEasyUAClientComplexData>();

            // Get the data type model provider. Provides methods to access data types in OPC UA model.
            IUADataTypeModelProvider dataTypeModelProvider = complexData.DataTypeModelProvider;

            // Resolve the data type ID from our data type ID, for encoding name "Default Binary".
            ValueResult<UAModelNodeDescriptor> encodingIdResult =
dataTypeModelProvider.ResolveEncodingIdFromDataTypeId(
                new UAModelNodeDescriptor(endpointDescriptor, dataTypeId),
                UABrowseNames.DefaultBinary);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!encodingIdResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", encodingIdResult.ErrorMessageBrief);
                return;
            }
            UAModelNodeDescriptor encodingId = encodingIdResult.Value;

            // Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA
model.
            IUADataTypeDictionaryProvider dataTypeDictionaryProvider = complexData.DataTypeDictionaryProvider;

            // Resolve the data type descriptor from the encoding ID.
            ValueResult<UADataTypeDescriptor> dataTypeDescriptorResult =
                dataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(encodingId);
```

```
            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!dataTypeDescriptorResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", dataTypeDescriptorResult.ErrorMessageBrief);
                return;
            }
            UADataTypeDescriptor dataTypeDescriptor = dataTypeDescriptorResult.Value;

            // The data type descriptor contains two pieces of information:
            // The data type dictionary ID: This determines the dictionary where the data type is defined.
            Console.WriteLine(dataTypeDescriptor.DataTypeDictionaryId);
            // And the data type description: It is a "pointer" into the data type dictionary itself, selecting a
specific
            // type definition inside the data type dictionary. The format of it depends on the data type system;
            // in our case, it is a string that is the name of one of the type elements in the XML document of
the data type
            // dictionary.
            Console.WriteLine(dataTypeDescriptor.DataTypeDescription);

            // Obtain the actual content of the data type dictionary.
            ValueResult<byte[]> dataTypeDictionaryResult =

dataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(dataTypeDescriptor.DataTypeDictionaryId);

            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            if (!dataTypeDictionaryResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", dataTypeDictionaryResult.ErrorMessageBrief);
                return;
            }
            byte[] dataTypeDictionary = dataTypeDictionaryResult.Value;

            // The data type dictionary returned is an array of bytes; its syntax and semantics depends on the
data type
            // system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
            string text = Encoding.UTF8.GetString(dataTypeDictionary);
            Console.WriteLine();
            Console.WriteLine(text);


            // Example output (truncated):
            //
            //http://opcua.demo-this.com:51211/UA/SampleServer;
NodeId="nsu=http://test.org/UA/Data/;ns=2;i=11422"
            //ScalarValueDataType
            //
            //<opc:TypeDictionary
            //  xmlns:opc="http://opcfoundation.org/BinarySchema/"
            //  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            //  xmlns:ua="http://opcfoundation.org/UA/"
            //  xmlns:tns="http://test.org/UA/Data/"
            //  DefaultByteOrder="LittleEndian"
            //  TargetNamespace="http://test.org/UA/Data/"
            //>
            //  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version
1.1.334.0 of the OPC UA deliverables. -->
            //  <opc:Import Namespace="http://opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
            //
            //  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
            //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
            //    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
            //    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
            //    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
            //    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
            //    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
            //    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
            //    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
            //    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
            //    <opc:Field Name="FloatValue" TypeName="opc:Float" />
            //    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
            //    <opc:Field Name="StringValue" TypeName="opc:String" />
```

```
            //    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
            //    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
            //    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
            //    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
            //    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
            //    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
            //    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
            //    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
            //    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
            //    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
            //    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
            //    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
            //    <opc:Field Name="Number" TypeName="ua:Variant" />
            //    <opc:Field Name="Integer" TypeName="ua:Variant" />
            //    <opc:Field Name="UInteger" TypeName="ua:Variant" />
            //  </opc:StructuredType>
            //
            //  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
            //    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
            //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
            //    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
        }
    }
}
```

Object Pascal

```
// Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
// content of the data type dictionary.

class procedure ResolveDataTypeDescriptorFromDataTypeEncodingId.Main;
var
  Client: _EasyUAClient;
  ComplexData: _EasyUAClientComplexData;
  DataTypeDictionary: OleVariant;
  DataTypeDictionaryResult: _ValueResult;
  DataTypeDescriptor: _UADataTypeDescriptor;
  DataTypeDescriptorResult: _ValueResult;
  DataTypeId: string;
  DataTypeModelProvider: _UADataTypeModelProvider;
  DataTypeDictionaryProvider: _UADataTypeDictionaryProvider;
  EncodingId: _UAModelNodeDescriptor;
  EncodingIdResult: _ValueResult;
  EncodingName: _UAQualifiedName;
  EndpointDescriptor: string;
  I: Cardinal;
  ModelNodeDescriptor: _UAModelNodeDescriptor;
  Text: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
    //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
    //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain the data type ID.
  //
  // In many cases, you would be able to obtain the data type ID of a particular node by reading its DataType
  // attribute.
  // The sample server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type, and
  // the actual values are then sub-types of this base data type. This abstract data type does not have any
encodings
  // associated with it and it is therefore not possible to extract its description from the server. We therefore
use
  // a hard-coded data type ID for one of the sub-types in this example.
  DataTypeId := 'nsu=http://test.org/UA/Data/;i=9440';    // ScalarValueDataType

  // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
```

```
  // operations.
  ComplexData :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData,
OpcLabs.EasyOpcUA')) as _EasyUAClientComplexData;

  // Get the data type model provider. Provides methods to access data types in OPC UA model.
  DataTypeModelProvider := ComplexData.DataTypeModelProvider;

  // Resolve the data type ID from our data type ID, for encoding name "Default Binary".
  ModelNodeDescriptor := CoUAModelNodeDescriptor.Create;
  ModelNodeDescriptor.EndpointDescriptor.UrlString := EndpointDescriptor;
  ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText := DataTypeId;
  EncodingName := CoUAQualifiedName.Create;
  EncodingName.StandardName := 'DefaultBinary';
  EncodingIdResult := DataTypeModelProvider.ResolveEncodingIdFromDataTypeId(ModelNodeDescriptor, EncodingName);
  // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
  if not EncodingIdResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [EncodingIdResult.ErrorMessageBrief]));
    Exit;
  end;
  EncodingId := IUnknown(EncodingIdResult.Value) as _UAModelNodeDescriptor;

  // Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA model.
  DataTypeDictionaryProvider := ComplexData.DataTypeDictionaryProvider;

  // Resolve the data type descriptor from the encoding ID.
  DataTypeDescriptorResult :=
DataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(EncodingId);
  // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
  if not DataTypeDescriptorResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [DataTypeDescriptorResult.ErrorMessageBrief]));
    Exit;
  end;
  DataTypeDescriptor := IUnknown(DataTypeDescriptorResult.Value) as _UADataTypeDescriptor;

  // The data type descriptor contains two pieces of information:
  // The data type dictionary ID: This determines the dictionary where the data type is defined.
  WriteLn(DataTypeDescriptor.DataTypeDictionaryId.ToString);
  // And the data type description: It is a "pointer" into the data type dictionary itself, selecting a specific
  // type definition inside the data type dictionary. The format of it depends on the data type system;
  // in our case, it is a string that is the name of one of the type elements in the XML document of the data
type
  // dictionary.
  WriteLn(DataTypeDescriptor.DataTypeDescription);

  // Obtain the actual content of the data type dictionary.
  DataTypeDictionaryResult :=

DataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(DataTypeDescriptor.DataTypeDictionaryId);

  // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
  if not DataTypeDictionaryResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [DataTypeDictionaryResult.ErrorMessageBrief]));
    Exit;
  end;
  DataTypeDictionary := DataTypeDictionaryResult.Value;

  // The data type dictionary returned is an array of bytes; its syntax and semantics depends on the data type
  // system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
  for I := VarArrayLowBound(DataTypeDictionary, 1) to VarArrayHighBound(DataTypeDictionary, 1) do
  begin
      Text := Text + Chr(Byte(DataTypeDictionary[I]));
  end;

  WriteLn;
  WriteLn(Text);

  // Example output (truncated):
  //
  //http://opcua.demo-this.com:51211/UA/SampleServer; NodeId="nsu=http://test.org/UA/Data/;ns=2;i=11422"
```

```
  //ScalarValueDataType
  //
  //<opc:TypeDictionary
  //  xmlns:opc="http://opcfoundation.org/BinarySchema/"
  //  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  //  xmlns:ua="http://opcfoundation.org/UA/"
  //  xmlns:tns="http://test.org/UA/Data/"
  //  DefaultByteOrder="LittleEndian"
  //  TargetNamespace="http://test.org/UA/Data/"
  //>
  //  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version 1.1.334.0
of the OPC UA deliverables. -->
  //  <opc:Import Namespace="http://opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
  //
  //  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
  //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
  //    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
  //    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
  //    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
  //    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
  //    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
  //    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
  //    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
  //    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
  //    <opc:Field Name="FloatValue" TypeName="opc:Float" />
  //    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
  //    <opc:Field Name="StringValue" TypeName="opc:String" />
  //    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
  //    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
  //    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
  //    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
  //    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
  //    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
  //    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
  //    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
  //    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
  //    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
  //    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
  //    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
  //    <opc:Field Name="Number" TypeName="ua:Variant" />
  //    <opc:Field Name="Integer" TypeName="ua:Variant" />
  //    <opc:Field Name="UInteger" TypeName="ua:Variant" />
  //  </opc:StructuredType>
  //
  //  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
  //    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
  //    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
  //    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />

end;
```

VB.NET

```
' Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
' content of the data type dictionary.

Imports System
Imports System.Text
Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.OperationModel.Generic
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.DataTypeModel
Imports OpcLabs.EasyOpc.UA.DataTypeModel.Extensions
Imports OpcLabs.EasyOpc.UA.InformationModel
Imports OpcLabs.EasyOpc.UA.Plugins.ComplexData

Namespace UADocExamples.ComplexData._IUADataTypeDictionaryProvider

    Friend Class ResolveDataTypeDescriptorFromDataTypeEncodingId
```

```
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Obtain the data type ID.
            '
            ' In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
            ' attribute, or easier, by calling the extension method ReadDataType on the IEasyUAClient interface.
The sample
            ' server, however, shows a more advanced approach in which the data type ID refers to an abstract
data type,
            ' and the actual values are then sub-types of this base data type. This abstract data type does not
have any
            ' encodings associated with it and it is therefore not possible to extract its description from the
server.
            ' We therefore use a hard-coded data type ID for one of the sub-types in this example.
            '
            ' The code to obtain the data type ID for given node would normally look like this:
            '    UANodeId dataTypeId = client.ReadDataType(
            '        endpointUriString,
            '        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            '
            Dim dataTypeId As UANodeId = "nsu=http://test.org/UA/Data/;i=9440"  ' ScalarValueDataType

            ' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
            ' operations.
            Dim complexData As IEasyUAClientComplexData = client.GetService(Of IEasyUAClientComplexData)()

            ' Get the data type model provider. Provides methods to access data types in OPC UA model.
            Dim dataTypeModelProvider As IUADataTypeModelProvider = complexData.DataTypeModelProvider

            ' Resolve the data type ID from our data type ID, for encoding name "Default Binary".
            Dim encodingIdResult As ValueResult(Of UAModelNodeDescriptor) =
dataTypeModelProvider.ResolveEncodingIdFromDataTypeId(
                New UAModelNodeDescriptor(endpointDescriptor, dataTypeId),
                UABrowseNames.DefaultBinary)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not encodingIdResult.Succeeded Then
                Console.WriteLine("*** Failure: {0}", encodingIdResult.ErrorMessageBrief)
                Exit Sub
            End If
            Dim encodingId As UAModelNodeDescriptor = encodingIdResult.Value

            ' Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA
model.
            Dim dataTypeDictionaryProvider As IUADataTypeDictionaryProvider =
complexData.DataTypeDictionaryProvider

            ' Resolve the data type descriptor from the encoding ID.
            Dim dataTypeDescriptorResult As ValueResult(Of UADataTypeDescriptor) =
                    dataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(encodingId)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not dataTypeDescriptorResult.Succeeded Then
                Console.WriteLine("*** Failure: {0}", dataTypeDescriptorResult.ErrorMessageBrief)
                Exit Sub
            End If
            Dim dataTypeDescriptor As UADataTypeDescriptor = dataTypeDescriptorResult.Value

            ' The data type descriptor contains two pieces of information:
            ' The data type dictionary ID: This determines the dictionary where the data type is defined.
            Console.WriteLine(dataTypeDescriptor.DataTypeDictionaryId)
            ' And the data type description: It is a "pointer" into the data type dictionary itself, selecting a
```

```
specific
            ' type definition inside the data type dictionary. The format of it depends on the data type system;
            ' in our case, it is a string that is the name of one of the type elements in the XML document of the
data type
            ' dictionary.
            Console.WriteLine(dataTypeDescriptor.DataTypeDescription)

            ' Obtain the actual content of the data type dictionary.
            Dim dataTypeDictionaryResult As ValueResult(Of Byte()) =

dataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(dataTypeDescriptor.DataTypeDictionaryId)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be
thrown.
            If Not dataTypeDictionaryResult.Succeeded Then
                Console.WriteLine("*** Failure: {0}", dataTypeDictionaryResult.ErrorMessageBrief)
                Exit Sub
            End If
            Dim dataTypeDictionary() As Byte = dataTypeDictionaryResult.Value

            ' The data type dictionary returned is an array of bytes; its syntax and semantics depends on the
data type
            ' system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
            Dim text As String = Encoding.UTF8.GetString(dataTypeDictionary)
            Console.WriteLine()
            Console.WriteLine(text)


            ' Example output (truncated):
            '
            'http://opcua.demo-this.com:51211/UA/SampleServer; NodeId="nsu=http://test.org/UA/Data/;ns=2;i=11422"
            'ScalarValueDataType
            '
            '<opc:TypeDictionary
            '  xmlns:opc="http://opcfoundation.org/BinarySchema/"
            '  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            '  xmlns:ua="http://opcfoundation.org/UA/"
            '  xmlns:tns="http://test.org/UA/Data/"
            '  DefaultByteOrder="LittleEndian"
            '  TargetNamespace="http://test.org/UA/Data/"
            '>
            '  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version
1.1.334.0 of the OPC UA deliverables. -->
            '  <opc:Import Namespace="http://opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
            '
            '  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
            '    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
            '    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
            '    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
            '    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
            '    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
            '    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
            '    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
            '    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
            '    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
            '    <opc:Field Name="FloatValue" TypeName="opc:Float" />
            '    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
            '    <opc:Field Name="StringValue" TypeName="opc:String" />
            '    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
            '    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
            '    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
            '    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
            '    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
            '    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
            '    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
            '    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
            '    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
            '    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
            '    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
            '    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
            '    <opc:Field Name="Number" TypeName="ua:Variant" />
            '    <opc:Field Name="Integer" TypeName="ua:Variant" />
            '    <opc:Field Name="UInteger" TypeName="ua:Variant" />
            '  </opc:StructuredType>
```

```
            '
            '  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
            '    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
            '    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
            '    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to obtain data type description object for complex data node with OPC UA Complex Data plug-in, and
the actual
Rem content of the data type dictionary.

Option Explicit

' Define which server we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain the data type ID.
'
' In many cases, you would be able to obtain the data type ID of a particular node by reading its DataType
' attribute.
' The sample server, however, shows a more advanced approach in which the data type ID refers to an abstract data
type, and
' the actual values are then sub-types of this base data type. This abstract data type does not have any
encodings
' associated with it and it is therefore not possible to extract its description from the server. We therefore
use
' a hard-coded data type ID for one of the sub-types in this example.
Dim dataTypeId: dataTypeId = "nsu=http://test.org/UA/Data/;i=9440"    ' ScalarValueDataType

' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex data
' operations.
Dim ComplexData: Set ComplexData = _
    Client.GetServiceByName("OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData, OpcLabs.EasyOpcUA")

' Get the data type model provider. Provides methods to access data types in OPC UA model.
Dim DataTypeModelProvider: Set DataTypeModelProvider = ComplexData.DataTypeModelProvider

' Resolve the data type ID from our data type ID, for encoding name "Default Binary".
Dim ModelNodeDescriptor: Set ModelNodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.InformationModel.UAModelNodeDescriptor")
ModelNodeDescriptor.EndpointDescriptor.UrlString = endpointDescriptor
ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText = dataTypeId
Dim EncodingName: Set EncodingName = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UAQualifiedName")
EncodingName.StandardName = "DefaultBinary"
Dim EncodingIdResult: Set EncodingIdResult = _
    DataTypeModelProvider.ResolveEncodingIdFromDataTypeId(ModelNodeDescriptor, EncodingName)
' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
If Not EncodingIdResult.Succeeded Then
    WScript.Echo "*** Failure: "  & EncodingIdResult.Exception.GetBaseException().Message
    WScript.Quit
End If
Dim EncodingId: Set EncodingId = EncodingIdResult.Value

' Get the data type dictionary provider. Provides methods to access data type dictionaries in OPC UA model.
Dim DataTypeDictionaryProvider: Set DataTypeDictionaryProvider = ComplexData.DataTypeDictionaryProvider

' Resolve the data type descriptor from the encoding ID.
Dim DataTypeDescriptorResult: Set DataTypeDescriptorResult = _
    DataTypeDictionaryProvider.ResolveDataTypeDescriptorFromDataTypeEncodingId(EncodingId)
' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
If Not DataTypeDescriptorResult.Succeeded Then
    WScript.Echo "*** Failure: "  & DataTypeDescriptorResult.Exception.GetBaseException().Message
    WScript.Quit
```

```
End If
Dim DataTypeDescriptor: Set DataTypeDescriptor = DataTypeDescriptorResult.Value

' The data type descriptor contains two pieces of information:
' The data type dictionary ID: This determines the dictionary where the data type is defined.
WScript.Echo DataTypeDescriptor.DataTypeDictionaryId
' And the data type description: It is a "pointer" into the data type dictionary itself, selecting a specific
' type definition inside the data type dictionary. The format of it depends on the data type system;
' in our case, it is a string that is the name of one of the type elements in the XML document of the data type
' dictionary.
WScript.Echo DataTypeDescriptor.DataTypeDescription

' Obtain the actual content of the data type dictionary.
Dim DataTypeDictionaryResult: Set DataTypeDictionaryResult = _

DataTypeDictionaryProvider.GetDataTypeDictionaryFromDataTypeDictionaryId(DataTypeDescriptor.DataTypeDictionaryId)
' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want exception be thrown.
If Not DataTypeDictionaryResult.Succeeded Then
    WScript.Echo "*** Failure: "  & DataTypeDictionaryResult.Exception.GetBaseException().Message
    WScript.Quit
End If
Dim DataTypeDictionary: DataTypeDictionary = DataTypeDictionaryResult.Value

' The data type dictionary returned is an array of bytes; its syntax and semantics depends on the data type
' system. In our case, we know that the data type dictionary is actually a string encoded in UTF-8.
Dim text
Dim i: For i = LBound(DataTypeDictionary) To UBound(DataTypeDictionary)
    text = text & Chr(DataTypeDictionary(i))
Next
WScript.Echo
WScript.Echo text


' Example output (truncated):
'
'http://opcua.demo-this.com:51211/UA/SampleServer; NodeId="nsu=http:'test.org/UA/Data/;ns=2;i=11422"
'ScalarValueDataType
'
'<opc:TypeDictionary
'  xmlns:opc="http:'opcfoundation.org/BinarySchema/"
'  xmlns:xsi="http:'www.w3.org/2001/XMLSchema-instance"
'  xmlns:ua="http:'opcfoundation.org/UA/"
'  xmlns:tns="http:'test.org/UA/Data/"
'  DefaultByteOrder="LittleEndian"
'  TargetNamespace="http:'test.org/UA/Data/"
'>
'  <!-- This File was generated on 2013-01-22 and supports the specifications supported by version 1.1.334.0 of
the OPC UA deliverables. -->
'  <opc:Import Namespace="http:'opcfoundation.org/UA/" Location="Opc.Ua.BinarySchema.bsd"/>
'
'  <opc:StructuredType Name="ScalarValueDataType" BaseType="ua:ExtensionObject">
'    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" />
'    <opc:Field Name="SByteValue" TypeName="opc:SByte" />
'    <opc:Field Name="ByteValue" TypeName="opc:Byte" />
'    <opc:Field Name="Int16Value" TypeName="opc:Int16" />
'    <opc:Field Name="UInt16Value" TypeName="opc:UInt16" />
'    <opc:Field Name="Int32Value" TypeName="opc:Int32" />
'    <opc:Field Name="UInt32Value" TypeName="opc:UInt32" />
'    <opc:Field Name="Int64Value" TypeName="opc:Int64" />
'    <opc:Field Name="UInt64Value" TypeName="opc:UInt64" />
'    <opc:Field Name="FloatValue" TypeName="opc:Float" />
'    <opc:Field Name="DoubleValue" TypeName="opc:Double" />
'    <opc:Field Name="StringValue" TypeName="opc:String" />
'    <opc:Field Name="DateTimeValue" TypeName="opc:DateTime" />
'    <opc:Field Name="GuidValue" TypeName="opc:Guid" />
'    <opc:Field Name="ByteStringValue" TypeName="opc:ByteString" />
'    <opc:Field Name="XmlElementValue" TypeName="ua:XmlElement" />
'    <opc:Field Name="NodeIdValue" TypeName="ua:NodeId" />
'    <opc:Field Name="ExpandedNodeIdValue" TypeName="ua:ExpandedNodeId" />
'    <opc:Field Name="QualifiedNameValue" TypeName="ua:QualifiedName" />
'    <opc:Field Name="LocalizedTextValue" TypeName="ua:LocalizedText" />
'    <opc:Field Name="StatusCodeValue" TypeName="ua:StatusCode" />
'    <opc:Field Name="VariantValue" TypeName="ua:Variant" />
```

```
'    <opc:Field Name="EnumerationValue" TypeName="ua:Int32" />
'    <opc:Field Name="StructureValue" TypeName="ua:ExtensionObject" />
'    <opc:Field Name="Number" TypeName="ua:Variant" />
'    <opc:Field Name="Integer" TypeName="ua:Variant" />
'    <opc:Field Name="UInteger" TypeName="ua:Variant" />
'  </opc:StructuredType>
'
'  <opc:StructuredType Name="ArrayValueDataType" BaseType="ua:ExtensionObject">
'    <opc:Field Name="NoOfBooleanValue" TypeName="opc:Int32" />
'    <opc:Field Name="BooleanValue" TypeName="opc:Boolean" LengthField="NoOfBooleanValue" />
'    <opc:Field Name="NoOfSByteValue" TypeName="opc:Int32" />
```

13.2.8.5 Examples - OPC UA Complex Data - Process a data type

The following example shows how a data type can be processed.

C#

```
// Shows how to process a data type, displaying some of its properties, recursively.

using System;
using System.Linq;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._DataType
{
    class Kind
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            UAGenericObject genericObject;
            try
            {
                genericObject = (UAGenericObject)client.ReadValue(endpointDescriptor,
```

```
nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
            // The data type is in the GenericData.DataType property of the
UAGenericObject.
            DataType dataType = genericObject.GenericData.DataType;

            // Process the data type. We will inspect some of its properties, and dump
them.
            ProcessDataType(dataType, maximumDepth: 2);
        }


        // Process the data type. It can be recursive in itself, so if you do not know
the data type you are dealing with,
        // it is recommended to make safeguards against infinite looping or recursion -
here, the maximumDepth.
        public static void ProcessDataType(DataType dataType, int maximumDepth)
        {
            if (maximumDepth == 0)
                return;

            Console.WriteLine();
            Console.WriteLine("dataType.Name: {0}", dataType.Name);

            switch (dataType.Kind)
            {
                case DataTypeKind.Enumeration:
                    Console.WriteLine("The data type is an enumeration.");
                    var enumerationDataType = (EnumerationDataType) dataType;
                    Console.WriteLine("It has {0} enumeration members.",
enumerationDataType.EnumerationMembers.Count);
                    Console.WriteLine("The names of the enumeration members are: {0}.",
                        String.Join(", ",
enumerationDataType.EnumerationMembers.Select(member => member.Name)));
                    // Here you can process the members, or inspect SizeInBits etc.
                    break;

                case DataTypeKind.Opaque:
                    Console.WriteLine("The data type is opaque.");
                    var opaqueDataType = (OpaqueDataType) dataType;
                    Console.WriteLine("Its size is {0} bits.",
opaqueDataType.SizeInBits);
                    // There isn't much more you can learn about an opaque data type
(well, it may have Description and
                    // other common members). It is, after all, opaque...
                    break;

                case DataTypeKind.Primitive:
                    Console.WriteLine("The data type is primitive.");
                    var primitiveDataType = (PrimitiveDataType) dataType;
                    Console.WriteLine("Its .NET value type is \"{0}\".",
```

```
primitiveDataType.ValueType);
                    // There isn't much more you can learn about the primitive data
type.
                    break;

                case DataTypeKind.Sequence:
                    Console.WriteLine("The data type is a sequence.");
                    var sequenceDataType = (SequenceDataType) dataType;
                    Console.WriteLine("Its length is {0} (-1 means that the length can
vary).", sequenceDataType.Length);

                    Console.WriteLine("A dump of the element data type follows.");
                    ProcessDataType(sequenceDataType.ElementDataType, maximumDepth -
1);
                    break;

                case DataTypeKind.Structured:
                    Console.WriteLine("The data type is structured.");
                    var structuredDataType = (StructuredDataType) dataType;
                    Console.WriteLine("It has {0} data fields.",
structuredDataType.DataFields.Count);
                    Console.WriteLine("The names of the data fields are: {0}.",
                        String.Join(", ", structuredDataType.DataFields.Select(field =>
field.Name)));

                    Console.WriteLine("A dump of each of the data fields follows.");
                    foreach (DataField dataField in structuredDataType.DataFields)
                    {
                        Console.WriteLine();
                        Console.WriteLine("dataField.Name: {0}", dataField.Name);
                        // Note that every data field also has properties like
IsLength, IsOptional, IsSwitch which might
                        // be of interest but we are not dumping them here.
                        ProcessDataType(dataField.DataType, maximumDepth - 1);
                    }
                    break;

                case DataTypeKind.Union:
                    Console.WriteLine("The data type is union.");
                    var unionDataType = (UnionDataType)dataType;
                    Console.WriteLine("It has {0} data fields.",
unionDataType.DataFields.Count);
                    Console.WriteLine("The names of the data fields are: {0}.",
                        String.Join(", ", unionDataType.DataFields.Select(field =>
field.Name)));
                    break;
            }
        }
    }
}
```

Object Pascal

```
// Shows how to process a data type, displaying some of its properties, recursively.

class procedure Kind.Main;
var
```

```
  Client: _EasyUAClient;
  DataType: OpcLabs_BaseLib_TLB._DataType;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node. We know that this node returns complex data, so we can type cast to
UAGenericObject.

  try
    GenericObject := IUnknown(Client.ReadValue(EndpointDescriptor, NodeDescriptor)) as
_UAGenericObject;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // The data type is in the GenericData.DataType property of the UAGenericObject.
  DataType := genericObject.GenericData.DataType;

  // Process the data type. We will inspect some of its properties, and dump them.
  ProcessDataType(DataType, 2);
end;

// Process the data type. It can be recursive in itself, so if you do not know the data
type you are dealing with,
// it is recommended to make safeguards against infinite looping or recursion - here,
the maximumDepth.
class procedure Kind.ProcessDataType(DataType: OpcLabs_BaseLib_TLB._DataType;
MaximumDepth: Cardinal);
var
  Count: Cardinal;
  DataField: _DataField;
  Element: OleVariant;
  ElementEnumerator: IEnumVARIANT;
  EnumerationMember: _EnumerationMember;
  EnumerationDataType: _EnumerationDataType;
  FieldNames: string;
  First: boolean;
  MemberNames: string;
  OpaqueDataType: _OpaqueDataType;
  PrimitiveDataType: _PrimitiveDataType;
  SequenceDataType: _SequenceDataType;
  StructuredDataType: _StructuredDataType;
```

```
  TypeName: WideString;
begin
  if MaximumDepth = 0 then
    Exit;

  WriteLn;
  WriteLn('dataType.Name: ', DataType.Name);

  case DataType.Kind of
    DataTypeKind_Enumeration:
      begin
        WriteLn('The data type is an enumeration.');
        EnumerationDataType := DataType as _EnumerationDataType;
        WriteLn(Format('It has %s enumeration members.',
[EnumerationDataType.EnumerationMembers.Count]));
        ElementEnumerator := EnumerationDataType.EnumerationMembers.GetEnumerator;
        MemberNames := '';
        First := True;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          EnumerationMember := IUnknown(Element) as _EnumerationMember;
          if First then
            First := False
          else
            MemberNames := MemberNames + ', ';
          MemberNames := MemberNames + EnumerationMember.Name;
        end;
        WriteLn(Format('The names of the enumeration members are: %s.',
[MemberNames]));
        // Here you can process the members, or inspect SizeInBits etc.
      end;
    DataTypeKind_Opaque:
      begin
        WriteLn('The data type is opaque.');
        OpaqueDataType := DataType as _OpaqueDataType;
        WriteLn(Format('Its size is %s bits.', [OpaqueDataType.SizeInBits]));
        // There isn't much more you can learn about an opaque data type (well, it may
have Description and
        // other common members). It is, after all, opaque...
      end;
    DataTypeKind_Primitive:
      begin
        WriteLn('The data type is primitive.');
        PrimitiveDataType := DataType as _PrimitiveDataType;
        PrimitiveDataType.ValueType.Get_ToString(TypeName);
        WriteLn(Format('Its .NET value type is "%s".', [TypeName]));
        // There isn't much more you can learn about the primitive data type.
      end;
    DataTypeKind_Sequence:
      begin
        WriteLn('The data type is a sequence.');
        SequenceDataType := DataType as _SequenceDataType;
        WriteLn(Format('Its length is %s (-1 means that the length can vary).',
[SequenceDataType.Length.ToString]));
        WriteLn('A dump of the element data type follows.');
        ProcessDataType(SequenceDataType.ElementDataType, MaximumDepth - 1);
      end;
```

```
    DataTypeKind_Structured:
      begin
        WriteLn('The data type is structured.');
        StructuredDataType := DataType as _StructuredDataType;
        WriteLn(Format('It has %s data fields.',
[StructuredDataType.DataFields.Count.ToString]));
        ElementEnumerator := StructuredDataType.DataFields.GetEnumerator;
        FieldNames := '';
        First := True;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          if First then
            First := False
          else
            FieldNames := FieldNames + ', ';
          FieldNames := FieldNames + Element.Name;
        end;
        WriteLn(Format('The names of the data fields are: %s.', [FieldNames]));

        WriteLn('A dump of each of the data fields follows.');
        ElementEnumerator := StructuredDataType.DataFields.GetEnumerator;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          DataField := IUnknown(Element) as _DataField;
          WriteLn;
          WriteLn(Format('dataField.Name: %s', [DataField.Name]));
          // Note that every data field also has properties like IsLength, IsOptional,
IsSwitch which might
          // be of interest but we are not dumping them here.
          ProcessDataType(DataField.DataType, MaximumDepth - 1);
        end;
    end;
  end;

end;
```

VB.NET

```
' Shows how to process a data type, displaying some of its properties, recursively.

Imports System
Imports System.Linq
Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._DataType

    Friend Class Kind

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
```

```
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            Dim genericObject As UAGenericObject
            Try
                genericObject = CType(client.ReadValue(endpointDescriptor,
nodeDescriptor), UAGenericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
            ' The data type is in the GenericData.DataType property of the
UAGenericObject.

            Dim dataType As DataType = genericObject.GenericData.DataType

            ' Process the data type. We will inspect some of its properties, and dump
them.
            ProcessDataType(dataType, maximumDepth:=2)
        End Sub


        ' Process the data type. It can be recursive in itself, so if you do not know
the data type you are dealing with,
        ' it is recommended to make safeguards against infinite looping or recursion -
here, the maximumDepth.
        Public Shared Sub ProcessDataType(dataType As DataType, ByVal maximumDepth As
Integer)
            If (maximumDepth = 0) Then
                Return
            End If

            Console.WriteLine()
            Console.WriteLine("dataType.Name: {0}", dataType.Name)

            Select Case (dataType.Kind)
                Case DataTypeKind.Enumeration
                    Console.WriteLine("The data type is an enumeration.")
                    Dim enumerationDataType = CType(dataType, EnumerationDataType)
                    Console.WriteLine("It has {0} enumeration members.",
enumerationDataType.EnumerationMembers.Count)
                    Console.WriteLine("The names of the enumeration members are: {0}.",
_
                                      String.Join(", ",
enumerationDataType.EnumerationMembers.Select(Function(member) member.Name)))
```

```
                    ' Here you can process the members, or inspect SizeInBits etc.

                Case DataTypeKind.Opaque
                    Console.WriteLine("The data type is opaque.")
                    Dim opaqueDataType = CType(dataType, OpaqueDataType)
                    Console.WriteLine("Its size is {0} bits.",
opaqueDataType.SizeInBits)
                    ' There isn't much more you can learn about an opaque data type
(well, it may have Description and
                    ' other common members). It is, after all, opaque...

                Case DataTypeKind.Primitive
                    Console.WriteLine("The data type is primitive.")
                    Dim primitiveDataType = CType(dataType, PrimitiveDataType)
                    Console.WriteLine("Its .NET value type is ""{0}"".",
primitiveDataType.ValueType)
                    ' There isn't much more you can learn about the primitive data
type.

                Case DataTypeKind.Sequence
                    Console.WriteLine("The data type is a sequence.")
                    Dim sequenceDataType = CType(dataType, SequenceDataType)
                    Console.WriteLine("Its length is {0} (-1 means that the length can
vary).", sequenceDataType.Length)
                    Console.WriteLine("A dump of the element data type follows.")
                    ProcessDataType(sequenceDataType.ElementDataType, (maximumDepth -
1))

                Case DataTypeKind.Structured
                    Console.WriteLine("The data type is structured.")
                    Dim structuredDataType = CType(dataType, StructuredDataType)
                    Console.WriteLine("It has {0} data fields.",
structuredDataType.DataFields.Count)
                    Console.WriteLine("The names of the data fields are: {0}.", _
                                      String.Join(", ",
structuredDataType.DataFields.Select(Function(field) field.Name)))
                    Console.WriteLine("A dump of each of the data fields follows.")

                    For Each dataField As DataField In structuredDataType.DataFields
                        Console.WriteLine()
                        Console.WriteLine("dataField.Name: {0}", dataField.Name)
                        ' Note that every data field also has properties like IsLength,
IsOptional, IsSwitch which might
                        ' be of interest but we are not dumping them here.
                        ProcessDataType(dataField.DataType, (maximumDepth - 1))
                    Next

                Case DataTypeKind.Union
                    Console.WriteLine("The data type is union.")
                    Dim unionDataType = CType(dataType, UnionDataType)
                    Console.WriteLine("It has {0} data fields.",
unionDataType.DataFields.Count)
                    Console.WriteLine("The names of the data fields are: {0}.",
                                      String.Join(", ",
unionDataType.DataFields.Select(Function(field) field.Name)))

            End Select
```

```
        End Sub
    End Class
End Namespace
```

13.2.8.6 Examples - OPC UA Complex Data - Process generic data

The following example shows how generic data can be processed.

C#

```
// Shows how to process generic data type, displaying some of its properties,
recursively.

using System;
using System.Collections.Generic;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._GenericData
{
    class DataTypeKind1
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            UAGenericObject genericObject;
            try
            {
                genericObject = (UAGenericObject)client.ReadValue(endpointDescriptor,
nodeDescriptor);
            }
            catch (UAException uaException)
            {
```

```
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Process the generic data type. We will inspect some of its properties,
and dump them.
            ProcessGenericData(genericObject.GenericData, maximumDepth: 2);
        }


        // Process the generic data type. Its structure can sometimes be quite deep,
therefore we are limiting the depth
        // of the recursion using maximumDepth.
        public static void ProcessGenericData(GenericData genericData, int
maximumDepth)
        {
            if (maximumDepth == 0)
                return;

            Console.WriteLine();
            Console.WriteLine("genericData.DataType: {0}", genericData.DataType);

            switch (genericData.DataTypeKind)
            {
                case DataTypeKind.Enumeration:
                    Console.WriteLine("The generic data is an enumeration.");
                    var enumerationData = (EnumerationData) genericData;
                    Console.WriteLine("Its value is {0}.", enumerationData.Value);
                    // There is also a ValueName that you can inspect (if known).
                    break;

                case DataTypeKind.Opaque:
                    Console.WriteLine("The generic data is opaque.");
                    var opaqueData = (OpaqueData) genericData;
                    Console.WriteLine("Its size is {0} bits.", opaqueData.SizeInBits);
                    Console.WriteLine("The data bytes are {0}.",
BitConverter.ToString(opaqueData.ByteArray));
                    // Use the Value property (a BitArray) if you need to access the
value bit by bit.
                    break;

                case DataTypeKind.Primitive:
                    Console.WriteLine("The generic data is primitive.");
                    var primitiveData = (PrimitiveData) genericData;
                    Console.WriteLine("Its value is \"{0}\".", primitiveData.Value);
                    break;

                case DataTypeKind.Sequence:
                    Console.WriteLine("The generic data is a sequence.");
                    var sequenceData = (SequenceData) genericData;
                    Console.WriteLine("It has {0} elements.",
sequenceData.Elements.Count);
                    Console.WriteLine("A dump of the elements follows.");
                    foreach (GenericData element in sequenceData.Elements)
                        ProcessGenericData(element, maximumDepth - 1);
                    break;
```

```
                case DataTypeKind.Structured:
                    Console.WriteLine("The generic data is structured.");
                    var structuredData = (StructuredData) genericData;
                    Console.WriteLine("It has {0} field data members.",
structuredData.FieldData.Count);
                    Console.WriteLine("The names of the fields are: {0}.",
                        String.Join(", ", structuredData.FieldData.Keys));

                    Console.WriteLine("A dump of each of the fields follows.");
                    foreach (KeyValuePair<string, GenericData> pair in
structuredData.FieldData)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Field name: {0}", pair.Key);
                        ProcessGenericData(pair.Value, maximumDepth - 1);
                    }
                    break;

                case DataTypeKind.Union:
                    Console.WriteLine("The generic data is union.");
                    var unionData = (UnionData)genericData;
                    Console.WriteLine("The name of current field is: {0}",
unionData.FieldName);
                    Console.WriteLine("Current field value is: {0}",
unionData.FieldValue);
                    break;
            }
        }
    }
}
```

Object Pascal

```
// Shows how to process generic data type, displaying some of its properties,
recursively

class procedure DataTypeKind1.Main;
var
  Client: _EasyUAClient;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node. We know that this node returns complex data, so we can type cast to
UAGenericObject.
```

```
  try
    GenericObject := _UAGenericObject(IUnknown(Client.ReadValue(EndpointDescriptor,
NodeDescriptor)));
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Process the generic data type. We will inspect some of its properties, and dump
them.
  ProcessGenericData(GenericObject.GenericData, 2);
end;

Function VariantToBytes(Const Value: OleVariant): TBytes;
Var
  Size: Integer;
  pData: Pointer;
Begin
  Size := Succ(VarArrayHighBound(Value, 1) - VarArrayLowBound(Value, 1));
  SetLength(Result, Size);
  pData := VarArrayLock(Value);
  Try
    Move(pData^, Pointer(Result)^, Size);
  Finally
    VarArrayUnlock(Value);
  End;
End;

class procedure DatatypeKind1.ProcessGenericData(GenericData:
OpcLabs_BaseLib_TLB._GenericData; MaximumDepth: Cardinal);
var
  ByteArray: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVARIANT;
  EnumerationData: _EnumerationData;
  First: boolean;
  Keys: string;
  OpaqueData: _OpaqueData;
  PrimitiveData: _PrimitiveData;
  SequenceData: _SequenceData;
  StructuredData: _StructuredData;
  Value: OpcLabs_BaseLib_TLB._GenericData;
begin
  if MaximumDepth = 0 then
    Exit;

  WriteLn;
  WriteLn('genericData.DataType: ', GenericData.DataType.ToString);

  case GenericData.DataTypeKind of
    DataTypeKind_Enumeration:
      begin
        WriteLn('The generic data is an enumeration.');
```

```
        EnumerationData := GenericData as _EnumerationData;
        WriteLn(Format('Its value is %s.', [EnumerationData.Value.ToString]));
        // There is also a ValueName that you can inspect (if known).
      end;
    DataTypeKind_Opaque:
      begin
        WriteLn('The generic data is opaque.');
        OpaqueData := GenericData as _OpaqueData;
        WriteLn(Format('Its size is %d bits.', [OpaqueData.SizeInBits]));
        TVarData(ByteArray).VType := varArray;
        TVarData(ByteArray).VArray := PVarArray(OpaqueData.ByteArray);
        WriteLn(Format('The data bytes are %s.',
[TEncoding.ANSI.GetString(VariantToBytes(ByteArray))]));
        // Use the Value property (a BitArray) if you need to access the value bit by
bit.
      end;
    DataTypeKind_Primitive:
      begin
        WriteLn('The generic data is primitive.');
        PrimitiveData := GenericData as _PrimitiveData;
        WriteLn(Format('Its value is "%s".', [PrimitiveData.Value]));
      end;
    DataTypeKind_Sequence:
      begin
        WriteLn('The generic data is a sequence.');
        SequenceData := GenericData as _SequenceData;
        WriteLn(Format('It has %s elements.', [SequenceData.Elements.Count.ToString]));
        WriteLn('A dump of the elements follows.');
        ElementEnumerator := SequenceData.Elements.GetEnumerator;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          ProcessGenericData(IUnknown(Element) as OpcLabs_BaseLib_TLB._GenericData,
MaximumDepth - 1);
        end;
      end;
    DataTypeKind_Structured:
      begin
        WriteLn('The generic data is structured.');
        StructuredData := GenericData as _StructuredData;
        WriteLn(Format('It has %s field data members.',
[StructuredData.FieldData.Count.ToString]));
        ElementEnumerator := StructuredData.FieldData.GetEnumerator;
        Keys := '';
        First := True;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
        begin
          if First then
            First := False
          else
            Keys := Keys + ', ';
          Keys := Keys + Element.Key;
        end;
        WriteLn(Format('The names of the fields are: %s.', [Keys]));

        WriteLn('A dump of each of the fields follows.');
        ElementEnumerator := StructuredData.FieldData.GetEnumerator;
        while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
```

```
        begin
          WriteLn;
          WriteLn(Format('Field name: %s', [Element.Key]));
          Value := IUnknown(Element.Value) as OpcLabs_BaseLib_TLB._GenericData;
          ProcessGenericData(Value, MaximumDepth - 1);
        end;
      end;
  end;

end;
```

VB.NET

```
' Shows how to process generic data type, displaying some of its properties,
recursively.

Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._GenericData

    Friend Class DataTypeKind1

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node. We know that this node returns complex data, so we can type
cast to UAGenericObject.
            Dim genericObject As UAGenericObject
            Try
                genericObject = CType(client.ReadValue(endpointDescriptor,
nodeDescriptor), UAGenericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Process the generic data type. We will inspect some of its properties,
and dump them.
```

```
            ProcessGenericData(genericObject.GenericData, maximumDepth:=2)
        End Sub


        ' Process the generic data type. Its structure can sometimes be quite deep,
therefore we are limiting the depth
        ' of the recursion using maximumDepth.
        Public Shared Sub ProcessGenericData(ByVal genericData As GenericData, ByVal
maximumDepth As Integer)
            If (maximumDepth = 0) Then
                Return
            End If

            Console.WriteLine()
            Console.WriteLine("genericData.DataType: {0}", genericData.DataType)

            Select Case (genericData.DataTypeKind)
                Case DataTypeKind.Enumeration
                    Console.WriteLine("The generic data is an enumeration.")
                    Dim enumerationData = CType(genericData, EnumerationData)
                    Console.WriteLine("Its value is {0}.", enumerationData.Value)
                    ' There is also a ValueName that you can inspect (if known).

                Case DataTypeKind.Opaque
                    Console.WriteLine("The generic data is opaque.")
                    Dim opaqueData = CType(genericData, OpaqueData)
                    Console.WriteLine("Its size is {0} bits.", opaqueData.SizeInBits)
                    Console.WriteLine("The data bytes are {0}.",
BitConverter.ToString(opaqueData.ByteArray))
                    ' Use the Value property (a BitArray) if you need to access the
value bit by bit.

                Case DataTypeKind.Primitive
                    Console.WriteLine("The generic data is primitive.")
                    Dim primitiveData = CType(genericData, PrimitiveData)
                    Console.WriteLine("Its value is ""{0}"".", primitiveData.Value)

                Case DataTypeKind.Sequence
                    Console.WriteLine("The generic data is a sequence.")
                    Dim sequenceData = CType(genericData, SequenceData)
                    Console.WriteLine("It has {0} elements.",
sequenceData.Elements.Count)
                    Console.WriteLine("A dump of the elements follows.")
                    For Each element As GenericData In sequenceData.Elements
                        ProcessGenericData(element, (maximumDepth - 1))
                    Next

                Case DataTypeKind.Structured
                    Console.WriteLine("The generic data is structured.")
                    Dim structuredData = CType(genericData, StructuredData)
                    Console.WriteLine("It has {0} field data members.",
structuredData.FieldData.Count)
                    Console.WriteLine("The names of the fields are: {0}.",
String.Join(", ", structuredData.FieldData.Keys))

                    Console.WriteLine("A dump of each of the fields follows.")
                    For Each pair As KeyValuePair(Of String, GenericData) In
```

```
structuredData.FieldData
                        Console.WriteLine()
                        Console.WriteLine("Field name: {0}", pair.Key)
                        ProcessGenericData(pair.Value, (maximumDepth - 1))
                    Next

                Case DataTypeKind.Union
                    Console.WriteLine("The generic data is union.")
                    Dim unionData = CType(genericData, UnionData)
                    Console.WriteLine("The name of current field is: {0}",
unionData.FieldName)
                    Console.WriteLine("Current field value is: {0}",
unionData.FieldValue)
            End Select

        End Sub
    End Class
End Namespace
```

13.2.8.7 Examples - OPC UA Complex Data - Read a value

The example below shows how to read and display OPC UA complex data.

C#

```
// Shows how to read complex data with OPC UA Complex Data plug-in.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._EasyUAClient
{
    class ReadValue
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; // [ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node which returns complex data. This is done in the same way as regular reads - just the data
            // returned is different.
            object value;
            try
            {
                value = client.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display basic information about what we have read.
            Console.WriteLine(value);


            // We know that this node returns complex data, so we can type cast to UAGenericObject.
            var genericObject = (UAGenericObject) value;
```

```
            // The actual data is in the GenericData property of the UAGenericObject.
            //
            // If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
            // specifier. In the debugger, you can view the same by displaying the private DebugView property.
            Console.WriteLine();
            Console.WriteLine("{0:V}", genericObject.GenericData);

            // For processing the internals of the data, refer to examples for GenericData and DataType classes.


            // Example output (truncated):
            //
            //(ScalarValueDataType) structured
            //
            //(ScalarValueDataType) structured
            //  [BooleanValue] (Boolean) primitive; True {System.Boolean}
            //  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
            //  [ByteValue] (Byte) primitive; 153 {System.Byte}
            //  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
            //  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
            //  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
            //  [ExpandedNodeIdValue] (ExpandedNodeId) structured
            //    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
            //    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
            //    [NodeIdType] (NodeIdType) enumeration; 3 (String)
            //    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
            //    [String] (StringNodeId) structured
            //      [Identifier] (CharArray) primitive; "????" {System.String}
            //      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
            //  [FloatValue] (Float) primitive; 78.37176 {System.Single}
            //  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
            //  [Int16Value] (Int16) primitive; 2793 {System.Int16}
            //  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
            //  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
            //  [Integer] (Variant) structured
            //    [ArrayDimensionsSpecified] sequence[1]
            //      [0] (Bit) primitive; False {System.Boolean}
            //    [ArrayLengthSpecified] sequence[1]
            //      [0] (Bit) primitive; False {System.Boolean}
            //    [Int64] sequence[1]
            //      [0] (Int64) primitive; 0 {System.Int64}
            //    [VariantType] sequence[6]
            //      [0] (Bit) primitive; False {System.Boolean}
            //      [1] (Bit) primitive; False {System.Boolean}
            //      [2] (Bit) primitive; False {System.Boolean}
            //      [3] (Bit) primitive; True {System.Boolean}
            //      [4] (Bit) primitive; False {System.Boolean}
            //      [5] (Bit) primitive; False {System.Boolean}
            //  [LocalizedTextValue] (LocalizedText) structured
            //    [Locale] (CharArray) primitive; "ko" {System.String}
            //    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
            //    [Reserved1] sequence[6]
            //      [0] (Bit) primitive; False {System.Boolean}
            //      [1] (Bit) primitive; False {System.Boolean}
            //      [2] (Bit) primitive; False {System.Boolean}
            //      [3] (Bit) primitive; False {System.Boolean}
            //      [4] (Bit) primitive; False {System.Boolean}
            //      [5] (Bit) primitive; False {System.Boolean}
            //    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
            //    [TextSpecified] (Bit) primitive; True {System.Boolean}
            //  [NodeIdValue] (NodeId) structured
        }
    }
}
```

Object Pascal

```
// Shows how to read complex data with OPC UA Complex Data plug-in.

class procedure ReadValue.Main;
var
  Client: _EasyUAClient;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
  Value: OleVariant;
begin
  // Define which server and node we will work with.
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  // [ObjectsFolder]/Data.Static.Scalar.StructureValue
```

```
// Instantiate the client object
Client := CoEasyUAClient.Create;

// Read a node which returns complex data. This is done in the same way as regular reads - just the data
// returned is different.

try
  Value := Client.ReadValue(EndpointDescriptor, NodeDescriptor);
except
  on E: EOleException do
  begin
    WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    Exit;
  end;
end;

// Display basic information about what we have read.
WriteLn('value: ', Value);

// We know that this node returns complex data, so it is a UAGenericObject.
GenericObject := IUnknown(Value) as _UAGenericObject;

// The actual data is in the GenericData property of the UAGenericObject.
//
// If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
// specifier. In the debugger, you can view the same by displaying the private DebugView property.
WriteLn;
WriteLn(GenericObject.GenericData.ToString_2['V', nil]);

// For processing the internals of the data, refer to examples for GenericData and DataType classes.

// Example output (truncated):
//
//(ScalarValueDataType) structured
//
//(ScalarValueDataType) structured
//  [BooleanValue] (Boolean) primitive; True {System.Boolean}
//  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
//  [ByteValue] (Byte) primitive; 153 {System.Byte}
//  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
//  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
//  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
//  [ExpandedNodeIdValue] (ExpandedNodeId) structured
//    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
//    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
//    [NodeIdType] (NodeIdType) enumeration; 3 (String)
//    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
//    [String] (StringNodeId) structured
//      [Identifier] (CharArray) primitive; "????" {System.String}
//      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
//  [FloatValue] (Float) primitive; 78.37176 {System.Single}
//  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
//  [Int16Value] (Int16) primitive; 2793 {System.Int16}
//  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
//  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
//  [Integer] (Variant) structured
//    [ArrayDimensionsSpecified] sequence[1]
//      [0] (Bit) primitive; False {System.Boolean}
//    [ArrayLengthSpecified] sequence[1]
//      [0] (Bit) primitive; False {System.Boolean}
//    [Int64] sequence[1]
//      [0] (Int64) primitive; 0 {System.Int64}
//    [VariantType] sequence[6]
//      [0] (Bit) primitive; False {System.Boolean}
//      [1] (Bit) primitive; False {System.Boolean}
//      [2] (Bit) primitive; False {System.Boolean}
//      [3] (Bit) primitive; True {System.Boolean}
//      [4] (Bit) primitive; False {System.Boolean}
//      [5] (Bit) primitive; False {System.Boolean}
//  [LocalizedTextValue] (LocalizedText) structured
//    [Locale] (CharArray) primitive; "ko" {System.String}
//    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
//    [Reserved1] sequence[6]
//      [0] (Bit) primitive; False {System.Boolean}
//      [1] (Bit) primitive; False {System.Boolean}
//      [2] (Bit) primitive; False {System.Boolean}
//      [3] (Bit) primitive; False {System.Boolean}
//      [4] (Bit) primitive; False {System.Boolean}
//      [5] (Bit) primitive; False {System.Boolean}
//    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
//    [TextSpecified] (Bit) primitive; True {System.Boolean}
//  [NodeIdValue] (NodeId) structured
```

```
end;
```

PHP

```
// Shows how to read complex data with OPC UA Complex Data plug-in.

// Define which server and node we will work with.
$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
$NodeDescriptor = "nsu=http://test.org/UA/Data/;i=10239";  // [ObjectsFolder]/Data.Static.Scalar.StructureValue

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Read a node which returns complex data. This is done in the same way as regular reads - just the data
// returned is different.

try
{
    $Value = $Client->ReadValue($EndpointDescriptor, $NodeDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display basic information about what we have read.
printf("Value: %s\n", $Value);

// We know that this node returns complex data, so it is a UAGenericObject.
$GenericObject = $Value;

// The actual data is in the GenericData property of the UAGenericObject.
//
// If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
// specifier. In the debugger, you can view the same by displaying the private DebugView property.
printf("\n");
printf("%s\n", $GenericObject->GenericData->ToString_3("V"));

// For processing the internals of the data, refer to examples for GenericData and DataType classes.

// Example output (truncated):
//
//(ScalarValueDataType) structured
//
//(ScalarValueDataType) structured
//  [BooleanValue] (Boolean) primitive; True {System.Boolean}
//  [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
//  [ByteValue] (Byte) primitive; 153 {System.Byte}
//  [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
//  [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
//  [EnumerationValue] (Int32) primitive; 0 {System.Int32}
//  [ExpandedNodeIdValue] (ExpandedNodeId) structured
//    [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
//    [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
//    [NodeIdType] (NodeIdType) enumeration; 3 (String)
//    [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
//    [String] (StringNodeId) structured
//      [Identifier] (CharArray) primitive; "????" {System.String}
//      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
//  [FloatValue] (Float) primitive; 78.37176 {System.Single}
//  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
//  [Int16Value] (Int16) primitive; 2793 {System.Int16}
//  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
//  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
//  [Integer] (Variant) structured
//    [ArrayDimensionsSpecified] sequence[1]
//      [0] (Bit) primitive; False {System.Boolean}
//    [ArrayLengthSpecified] sequence[1]
//      [0] (Bit) primitive; False {System.Boolean}
//    [Int64] sequence[1]
//      [0] (Int64) primitive; 0 {System.Int64}
//    [VariantType] sequence[6]
//      [0] (Bit) primitive; False {System.Boolean}
//      [1] (Bit) primitive; False {System.Boolean}
//      [2] (Bit) primitive; False {System.Boolean}
//      [3] (Bit) primitive; True {System.Boolean}
//      [4] (Bit) primitive; False {System.Boolean}
//      [5] (Bit) primitive; False {System.Boolean}
//  [LocalizedTextValue] (LocalizedText) structured
//    [Locale] (CharArray) primitive; "ko" {System.String}
```

```
//    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
//    [Reserved1] sequence[6]
//      [0] (Bit) primitive; False {System.Boolean}
//      [1] (Bit) primitive; False {System.Boolean}
//      [2] (Bit) primitive; False {System.Boolean}
//      [3] (Bit) primitive; False {System.Boolean}
//      [4] (Bit) primitive; False {System.Boolean}
//      [5] (Bit) primitive; False {System.Boolean}
//    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
//    [TextSpecified] (Bit) primitive; True {System.Boolean}
//  [NodeIdValue] (NodeId) structured
```

VB.NET

```
' Shows how to read complex data with OPC UA Complex Data plug-in.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._EasyUAClient

    Friend Class ReadValue

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node which returns complex data. This is done in the same way as regular reads - just the data
            ' returned is different.
            Dim value As Object
            Try
                value = client.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display basic information about what we have read.
            Console.WriteLine(value)


            ' We know that this node returns complex data, so we can type cast to UAGenericObject.
            Dim genericObject = CType(value, UAGenericObject)

            ' The actual data is in the GenericData property of the UAGenericObject.
            '
            ' If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
            ' specifier. In the debugger, you can view the same by displaying the private DebugView property.
            Console.WriteLine()
            Console.WriteLine("{0:V}", genericObject.GenericData)

            ' For processing the internals of the data, refer to examples for GenericData and DataType classes.


            ' Example output (truncated):
            '
            '(ScalarValueDataType) structured
            '
            '(ScalarValueDataType) structured
            '   [BooleanValue] (Boolean) primitive; True {System.Boolean}
            '   [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
            '   [ByteValue] (Byte) primitive; 153 {System.Byte}
            '   [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
            '   [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
            '   [EnumerationValue] (Int32) primitive; 0 {System.Int32}
            '   [ExpandedNodeIdValue] (ExpandedNodeId) structured
            '     [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
            '     [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
            '     [NodeIdType] (NodeIdType) enumeration; 3 (String)
            '     [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
```

```
            '    [String] (StringNodeId) structured
            '      [Identifier] (CharArray) primitive; "????" {System.String}
            '      [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
            '  [FloatValue] (Float) primitive; 78.37176 {System.Single}
            '  [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
            '  [Int16Value] (Int16) primitive; 2793 {System.Int16}
            '  [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
            '  [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
            '  [Integer] (Variant) structured
            '    [ArrayDimensionsSpecified] sequence[1]
            '      [0] (Bit) primitive; False {System.Boolean}
            '    [ArrayLengthSpecified] sequence[1]
            '      [0] (Bit) primitive; False {System.Boolean}
            '    [Int64] sequence[1]
            '      [0] (Int64) primitive; 0 {System.Int64}
            '    [VariantType] sequence[6]
            '      [0] (Bit) primitive; False {System.Boolean}
            '      [1] (Bit) primitive; False {System.Boolean}
            '      [2] (Bit) primitive; False {System.Boolean}
            '      [3] (Bit) primitive; True {System.Boolean}
            '      [4] (Bit) primitive; False {System.Boolean}
            '      [5] (Bit) primitive; False {System.Boolean}
            '  [LocalizedTextValue] (LocalizedText) structured
            '    [Locale] (CharArray) primitive; "ko" {System.String}
            '    [LocaleSpecified] (Bit) primitive; True {System.Boolean}
            '    [Reserved1] sequence[6]
            '      [0] (Bit) primitive; False {System.Boolean}
            '      [1] (Bit) primitive; False {System.Boolean}
            '      [2] (Bit) primitive; False {System.Boolean}
            '      [3] (Bit) primitive; False {System.Boolean}
            '      [4] (Bit) primitive; False {System.Boolean}
            '      [5] (Bit) primitive; False {System.Boolean}
            '    [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
            '    [TextSpecified] (Bit) primitive; True {System.Boolean}
            '  [NodeIdValue] (NodeId) structured
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to read and display data of an attribute (value, timestamps, and status code).

Option Explicit

' Define which server and node we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
Dim nodeDescriptor: nodeDescriptor = _
    "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Read a node which returns complex data. This is done in the same way as regular reads - just the data
' returned is different.
On Error Resume Next
Dim Value: Set Value = Client.ReadValue(endpointDescriptor, nodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display basic information about what we have read.
WScript.Echo Value


' We know that this node returns complex data, so it is a UAGenericObject.
Dim GenericObject: Set GenericObject = Value

' The actual data is in the GenericData property of the UAGenericObject.
'
' If we want to see the whole hierarchy of the received complex data, we can format it with the "V" (verbose)
' specifier. In the debugger, you can view the same by displaying the private DebugView property.
WScript.Echo
WScript.Echo GenericObject.GenericData.ToString_2("V", Nothing)

' For processing the internals of the data, refer to examples for GenericData and DataType classes.
```

```
' Example output (truncated):
'
'(ScalarValueDataType) structured
'
'(ScalarValueDataType) structured
'   [BooleanValue] (Boolean) primitive; True {System.Boolean}
'   [ByteStringValue] (ByteString) primitive; System.Byte[] {System.Byte[]}
'   [ByteValue] (Byte) primitive; 153 {System.Byte}
'   [DateTimeValue] (DateTime) primitive; 5/11/2013 4:32:00 PM {System.DateTime}
'   [DoubleValue] (Double) primitive; -8.93178007363702E+27 {System.Double}
'   [EnumerationValue] (Int32) primitive; 0 {System.Int32}
'   [ExpandedNodeIdValue] (ExpandedNodeId) structured
'     [NamespaceURI] (CharArray) primitive; "http://samples.org/UA/memorybuffer/Instance" {System.String}
'     [NamespaceURISpecified] (Bit) primitive; True {System.Boolean}
'     [NodeIdType] (NodeIdType) enumeration; 3 (String)
'     [ServerIndexSpecified] (Bit) primitive; False {System.Boolean}
'     [String] (StringNodeId) structured
'       [Identifier] (CharArray) primitive; "????" {System.String}
'       [NamespaceIndex] (UInt16) primitive; 0 {System.UInt16}
'   [FloatValue] (Float) primitive; 78.37176 {System.Single}
'   [GuidValue] (Guid) primitive; 8129cdaf-24d9-8140-64f2-3a6d7a957fd7 {System.Guid}
'   [Int16Value] (Int16) primitive; 2793 {System.Int16}
'   [Int32Value] (Int32) primitive; 1133391074 {System.Int32}
'   [Int64Value] (Int64) primitive; -1039109760798965779 {System.Int64}
'   [Integer] (Variant) structured
'     [ArrayDimensionsSpecified] sequence[1]
'       [0] (Bit) primitive; False {System.Boolean}
'     [ArrayLengthSpecified] sequence[1]
'       [0] (Bit) primitive; False {System.Boolean}
'     [Int64] sequence[1]
'       [0] (Int64) primitive; 0 {System.Int64}
'     [VariantType] sequence[6]
'       [0] (Bit) primitive; False {System.Boolean}
'       [1] (Bit) primitive; False {System.Boolean}
'       [2] (Bit) primitive; False {System.Boolean}
'       [3] (Bit) primitive; True {System.Boolean}
'       [4] (Bit) primitive; False {System.Boolean}
'       [5] (Bit) primitive; False {System.Boolean}
'   [LocalizedTextValue] (LocalizedText) structured
'     [Locale] (CharArray) primitive; "ko" {System.String}
'     [LocaleSpecified] (Bit) primitive; True {System.Boolean}
'     [Reserved1] sequence[6]
'       [0] (Bit) primitive; False {System.Boolean}
'       [1] (Bit) primitive; False {System.Boolean}
'       [2] (Bit) primitive; False {System.Boolean}
'       [3] (Bit) primitive; False {System.Boolean}
'       [4] (Bit) primitive; False {System.Boolean}
'       [5] (Bit) primitive; False {System.Boolean}
'     [Text] (CharArray) primitive; "? ?? ??+ ??? ??) ?: ???? ?! ?!" {System.String}
'     [TextSpecified] (Bit) primitive; True {System.Boolean}
'   [NodeIdValue] (NodeId) structured
```

13.2.8.8 Examples - OPC UA Complex Data - Resolve a data type

The example below shows how to resolve a data type. The main work is done by the ResolveDataType Method.

C#

```
// Shows how to obtain object describing the data type of complex data node with OPC UA Complex
Data plug-in.

using System;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.BaseLib.OperationModel.Generic;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.InformationModel;
using OpcLabs.EasyOpc.UA.Plugins.ComplexData;
```

```
namespace UADocExamples.ComplexData._IEasyUAClientComplexData
{
    class ResolveDataType
    {
        public static void Main1()
        {
            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Obtain the data type ID.
            //
            // In many cases, you would be able to obtain the data type ID of a particular node by
reading its DataType
            // attribute, or easier, by calling the extension method ReadDataType on the
IEasyUAClient interface.
            // The sample server, however, shows a more advanced approach in which the data type ID
refers to an abstract
            // data type, and the actual values are then sub-types of this base data type. This
abstract data type does not
            // have any encodings associated with it and it is therefore not possible to extract
its description from the
            // server. We therefore use a hard-coded data type ID for one of the sub-types in this
example.
            //
            // The code to obtain the data type ID for given node would normally look like this:
            //    UANodeId dataTypeId = client.ReadDataType(
            //        endpointDescriptor,
            //        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            //
            UANodeId dataTypeId = "nsu=http://test.org/UA/Data/;i=9440";    // ScalarValueDataType

            // Get the IEasyUAClientComplexData service from the client. This is needed for
advanced complex data
            // operations.
            IEasyUAClientComplexData complexData = client.GetService<IEasyUAClientComplexData>();

            // Resolve the data type ID to the data type object, containing description of the data
type.
            ValueResult<DataType> dataTypeResult = complexData.ResolveDataType(
                new UAModelNodeDescriptor(endpointDescriptor, dataTypeId),
                UABrowseNames.DefaultBinary);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead if you want
exception be thrown.
            if (!dataTypeResult.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", dataTypeResult.ErrorMessageBrief);
                return;
            }

            // The actual data type is in the Value property.
            // Display basic information about what we have obtained.
            Console.WriteLine(dataTypeResult.Value);

            // If we want to see the whole hierarchy of the received data type, we can format it
with the "V" (verbose)
            // specifier. In the debugger, you can view the same by displaying the private
DebugView property.
            Console.WriteLine();
```

```
            Console.WriteLine("{0:V}", dataTypeResult.Value);

            // For processing the internals of the data type, refer to examples for GenericData
class.


            // Example output (truncated):
            //
            //ScalarValueDataType = structured
            //
            //ScalarValueDataType = structured
            //   [BooleanValue] Boolean = primitive(System.Boolean)
            //   [ByteStringValue] ByteString = primitive(System.Byte[])
            //   [ByteValue] Byte = primitive(System.Byte)
            //   [DateTimeValue] DateTime = primitive(System.DateTime)
            //   [DoubleValue] Double = primitive(System.Double)
            //   [EnumerationValue] Int32 = primitive(System.Int32)
            //   [ExpandedNodeIdValue] ExpandedNodeId = structured
            //     [ByteString] optional ByteStringNodeId = structured
            //       [Identifier] ByteString = primitive(System.Byte[])
            //       [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //     [FourByte] optional FourByteNodeId = structured
            //       [Identifier] UInt16 = primitive(System.UInt16)
            //       [NamespaceIndex] Byte = primitive(System.Byte)
            //     [Guid] optional GuidNodeId = structured
            //       [Identifier] Guid = primitive(System.Guid)
            //       [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //     [NamespaceURI] optional CharArray = primitive(System.String)
            //     [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
            //     [NodeIdType] switch NodeIdType = enumeration(6)
            //       TwoByte = 0
            //       FourByte = 1
            //       Numeric = 2
            //       String = 3
            //       Guid = 4
            //       ByteString = 5
            //     [Numeric] optional NumericNodeId = structured
            //       [Identifier] UInt32 = primitive(System.UInt32)
            //       [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //     [ServerIndex] optional UInt32 = primitive(System.UInt32)
            //     [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
            //     [String] optional StringNodeId = structured
            //       [Identifier] CharArray = primitive(System.String)
            //       [NamespaceIndex] UInt16 = primitive(System.UInt16)
            //     [TwoByte] optional TwoByteNodeId = structured
            //       [Identifier] Byte = primitive(System.Byte)
            //   [FloatValue] Float = primitive(System.Single)
            //   [GuidValue] Guid = primitive(System.Guid)
            //   [Int16Value] Int16 = primitive(System.Int16)
            //   [Int32Value] Int32 = primitive(System.Int32)
            //   [Int64Value] Int64 = primitive(System.Int64)
            //   [Integer] Variant = structured
            //     [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
            //     [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            //     [ArrayLength] length optional Int32 = primitive(System.Int32)
            //     [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            //     [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
            //     [Byte] optional sequence[*] of Byte = primitive(System.Byte)
        }
    }
}
```

Object Pascal

```
// Shows how to obtain object describing the data type of complex data node with OPC UA Complex
Data plug-in.

class procedure ResolveDataType.Main;
var
  Client: _EasyUAClient;
  ComplexData: _EasyUAClientComplexData;
  DataType: _DataType;
  DataTypeId: string;
  DataTypeResult: _ValueResult;
  EncodingName: _UAQualifiedName;
  EndpointDescriptor: string;
  ModelNodeDescriptor: _UAModelNodeDescriptor;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
    //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
    //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain the data type ID.
  //
  // In many cases, you would be able to obtain the data type ID of a particular node by reading
its DataType
  // attribute.
  // The sample server, however, shows a more advanced approach in which the data type ID refers to
an abstract data type, and
  // the actual values are then sub-types of this base data type. This abstract data type does not
have any encodings
  // associated with it and it is therefore not possible to extract its description from the
server. We therefore use
  // a hard-coded data type ID for one of the sub-types in this example.
  DataTypeId := 'nsu=http://test.org/UA/Data/;i=9440';    // ScalarValueDataType

  // Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex
data
  // operations.
  ComplexData :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData,
OpcLabs.EasyOpcUA')) as _EasyUAClientComplexData;

  // Resolve the data type ID to the data type object, containing description of the data type.
  ModelNodeDescriptor := CoUAModelNodeDescriptor.Create;
  ModelNodeDescriptor.EndpointDescriptor.UrlString := EndpointDescriptor;
  ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText := DataTypeId;
  EncodingName := CoUAQualifiedName.Create;
  EncodingName.StandardName := 'DefaultBinary';
  DataTypeResult := ComplexData.ResolveDataType(ModelNodeDescriptor, EncodingName);
  if not DataTypeResult.Succeeded then
  begin
    WriteLn(Format('*** Failure: %s', [DataTypeResult.ErrorMessageBrief]));
    Exit;
  end;

  // The actual data type is in the Value property.
  DataType := IUnknown(DataTypeResult.Value) as _DataType;

  // Display basic information about what we have obtained.
  WriteLn(DataType.ToString);

  // If we want to see the whole hierarchy of the received data type, we can format it with the "V"
(verbose)
```

```
  // specifier.
  Writeln;
  Writeln(DataType.ToString_2['V', nil]);

  // For processing the internals of the data type, refer to examples for GenericData class.


  // Example output (truncated):
  //
  //ScalarValueDataType = structured
  //
  //ScalarValueDataType = structured
  //  [BooleanValue] Boolean = primitive(System.Boolean)
  //  [ByteStringValue] ByteString = primitive(System.Byte[])
  //  [ByteValue] Byte = primitive(System.Byte)
  //  [DateTimeValue] DateTime = primitive(System.DateTime)
  //  [DoubleValue] Double = primitive(System.Double)
  //  [EnumerationValue] Int32 = primitive(System.Int32)
  //  [ExpandedNodeIdValue] ExpandedNodeId = structured
  //    [ByteString] optional ByteStringNodeId = structured
  //      [Identifier] ByteString = primitive(System.Byte[])
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [FourByte] optional FourByteNodeId = structured
  //      [Identifier] UInt16 = primitive(System.UInt16)
  //      [NamespaceIndex] Byte = primitive(System.Byte)
  //    [Guid] optional GuidNodeId = structured
  //      [Identifier] Guid = primitive(System.Guid)
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [NamespaceURI] optional CharArray = primitive(System.String)
  //    [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
  //    [NodeIdType] switch NodeIdType = enumeration(6)
  //      TwoByte = 0
  //      FourByte = 1
  //      Numeric = 2
  //      String = 3
  //      Guid = 4
  //      ByteString = 5
  //    [Numeric] optional NumericNodeId = structured
  //      [Identifier] UInt32 = primitive(System.UInt32)
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [ServerIndex] optional UInt32 = primitive(System.UInt32)
  //    [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
  //    [String] optional StringNodeId = structured
  //      [Identifier] CharArray = primitive(System.String)
  //      [NamespaceIndex] UInt16 = primitive(System.UInt16)
  //    [TwoByte] optional TwoByteNodeId = structured
  //      [Identifier] Byte = primitive(System.Byte)
  //  [FloatValue] Float = primitive(System.Single)
  //  [GuidValue] Guid = primitive(System.Guid)
  //  [Int16Value] Int16 = primitive(System.Int16)
  //  [Int32Value] Int32 = primitive(System.Int32)
  //  [Int64Value] Int64 = primitive(System.Int64)
  //  [Integer] Variant = structured
  //    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
  //    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
  //    [ArrayLength] length optional Int32 = primitive(System.Int32)
  //    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
  //    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
  //    [Byte] optional sequence[*] of Byte = primitive(System.Byte)

end;
```

VB.NET

```
' Shows how to obtain object describing the data type of complex data node with OPC UA Complex Data
plug-in.

Imports System
Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.BaseLib.OperationModel.Generic
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.InformationModel
Imports OpcLabs.EasyOpc.UA.Plugins.ComplexData

Namespace UADocExamples.ComplexData._IEasyUAClientComplexData

    Friend Class ResolveDataType

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Obtain the data type ID.
            '
            ' In many cases, you would be able to obtain the data type ID of a particular node by
reading its DataType
            ' attribute, or easier, by calling the extension method ReadDataType on the
IEasyUAClient interface. The sample
            ' server, however, shows a more advanced approach in which the data type ID refers to
an abstract data type,
            ' and the actual values are then sub-types of this base data type. This abstract data
type does not have any
            ' encodings associated with it and it is therefore not possible to extract its
description from the server.
            ' We therefore use a hard-coded data type ID for one of the sub-types in this example.
            '
            ' The code to obtain the data type ID for given node would normally look like this:
            '    UANodeId dataTypeId = client.ReadDataType(
            '        endpointDescriptor,
            '        "nsu=http://test.org/UA/Data/;i=10239");    //
[ObjectsFolder]/Data.Static.Scalar.StructureValue
            '
            Dim dataTypeId As UANodeId = "nsu=http://test.org/UA/Data/;i=9440"  '
ScalarValueDataType

            ' Get the IEasyUAClientComplexData service from the client. This is needed for advanced
complex data
            ' operations.
            Dim complexData As IEasyUAClientComplexData = client.GetService(Of
IEasyUAClientComplexData)()

            ' Resolve the data type ID to the data type object, containing description of the data
type.
            Dim dataTypeResult As ValueResult(Of DataType) = complexData.ResolveDataType( _
                New UAModelNodeDescriptor(endpointDescriptor, dataTypeId), _
                UABrowseNames.DefaultBinary)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if you want
exception be thrown.
            If Not dataTypeResult.Succeeded Then
```

```
            Console.WriteLine("*** Failure: {0}", dataTypeResult.ErrorMessageBrief)
            Exit Sub
        End If

        ' The actual data type is in the Value property.
        ' Display basic information about what we have obtained.
        Console.WriteLine(dataTypeResult.Value)

        ' If we want to see the whole hierarchy of the received data type, we can format it
with the "V" (verbose)
        ' specifier. In the debugger, you can view the same by displaying the private DebugView
property.
        Console.WriteLine()
        Console.WriteLine("{0:V}", dataTypeResult.Value)

        ' For processing the internals of the data type, refer to examples for GenericData
class.


        ' Example output (truncated):
        '
        'ScalarValueDataType = structured
        '
        'ScalarValueDataType = structured
        '   [BooleanValue] Boolean = primitive(System.Boolean)
        '   [ByteStringValue] ByteString = primitive(System.Byte[])
        '   [ByteValue] Byte = primitive(System.Byte)
        '   [DateTimeValue] DateTime = primitive(System.DateTime)
        '   [DoubleValue] Double = primitive(System.Double)
        '   [EnumerationValue] Int32 = primitive(System.Int32)
        '   [ExpandedNodeIdValue] ExpandedNodeId = structured
        '     [ByteString] optional ByteStringNodeId = structured
        '       [Identifier] ByteString = primitive(System.Byte[])
        '       [NamespaceIndex] UInt16 = primitive(System.UInt16)
        '     [FourByte] optional FourByteNodeId = structured
        '       [Identifier] UInt16 = primitive(System.UInt16)
        '       [NamespaceIndex] Byte = primitive(System.Byte)
        '     [Guid] optional GuidNodeId = structured
        '       [Identifier] Guid = primitive(System.Guid)
        '       [NamespaceIndex] UInt16 = primitive(System.UInt16)
        '     [NamespaceURI] optional CharArray = primitive(System.String)
        '     [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
        '     [NodeIdType] switch NodeIdType = enumeration(6)
        '       TwoByte = 0
        '       FourByte = 1
        '       Numeric = 2
        '       String = 3
        '       Guid = 4
        '       ByteString = 5
        '     [Numeric] optional NumericNodeId = structured
        '       [Identifier] UInt32 = primitive(System.UInt32)
        '       [NamespaceIndex] UInt16 = primitive(System.UInt16)
        '     [ServerIndex] optional UInt32 = primitive(System.UInt32)
        '     [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
        '     [String] optional StringNodeId = structured
        '       [Identifier] CharArray = primitive(System.String)
        '       [NamespaceIndex] UInt16 = primitive(System.UInt16)
        '     [TwoByte] optional TwoByteNodeId = structured
        '       [Identifier] Byte = primitive(System.Byte)
        '   [FloatValue] Float = primitive(System.Single)
        '   [GuidValue] Guid = primitive(System.Guid)
        '   [Int16Value] Int16 = primitive(System.Int16)
        '   [Int32Value] Int32 = primitive(System.Int32)
        '   [Int64Value] Int64 = primitive(System.Int64)
        '   [Integer] Variant = structured
```

```
            '    [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
            '    [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            '    [ArrayLength] length optional Int32 = primitive(System.Int32)
            '    [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
            '    [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
            '    [Byte] optional sequence[*] of Byte = primitive(System.Byte)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to obtain object describing the data type of complex data node with OPC UA Complex
Data plug-in.

Option Explicit

' Define which server we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain the data type ID.
'
' In many cases, you would be able to obtain the data type ID of a particular node by reading its
DataType
' attribute.
' The sample server, however, shows a more advanced approach in which the data type ID refers to an
abstract data type, and
' the actual values are then sub-types of this base data type. This abstract data type does not
have any encodings
' associated with it and it is therefore not possible to extract its description from the server.
We therefore use
' a hard-coded data type ID for one of the sub-types in this example.
Dim dataTypeId: dataTypeId = "nsu=http://test.org/UA/Data/;i=9440"    ' ScalarValueDataType

' Get the IEasyUAClientComplexData service from the client. This is needed for advanced complex
data
' operations.
Dim ComplexData: Set ComplexData = _
    Client.GetServiceByName("OpcLabs.EasyOpc.UA.Plugins.ComplexData.IEasyUAClientComplexData,
OpcLabs.EasyOpcUA")

' Resolve the data type ID to the data type object, containing description of the data type.
Dim ModelNodeDescriptor: Set ModelNodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.InformationModel.UAModelNodeDescriptor")
ModelNodeDescriptor.EndpointDescriptor.UrlString = endpointDescriptor
ModelNodeDescriptor.NodeDescriptor.NodeId.ExpandedText = dataTypeId
Dim EncodingName: Set EncodingName =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UAQualifiedName")
EncodingName.StandardName = "DefaultBinary"
Dim DataTypeResult: Set DataTypeResult = ComplexData.ResolveDataType(ModelNodeDescriptor,
EncodingName)
If Not DataTypeResult.Succeeded Then
    WScript.Echo "*** Failure: "  & DataTypeResult.Exception.GetBaseException().Message
    WScript.Quit
End If

' The actual data type is in the Value property.
Dim DataType: Set DataType = DataTypeResult.Value
```

```
' Display basic information about what we have obtained.
WScript.Echo DataType

' If we want to see the whole hierarchy of the received data type, we can format it with the "V"
(verbose)
' specifier.
WScript.Echo
WScript.Echo DataType.ToString_2("V", Nothing)

' For processing the internals of the data type, refer to examples for GenericData class.


' Example output (truncated):
'
'ScalarValueDataType = structured
'
'ScalarValueDataType = structured
'   [BooleanValue] Boolean = primitive(System.Boolean)
'   [ByteStringValue] ByteString = primitive(System.Byte[])
'   [ByteValue] Byte = primitive(System.Byte)
'   [DateTimeValue] DateTime = primitive(System.DateTime)
'   [DoubleValue] Double = primitive(System.Double)
'   [EnumerationValue] Int32 = primitive(System.Int32)
'   [ExpandedNodeIdValue] ExpandedNodeId = structured
'     [ByteString] optional ByteStringNodeId = structured
'       [Identifier] ByteString = primitive(System.Byte[])
'       [NamespaceIndex] UInt16 = primitive(System.UInt16)
'     [FourByte] optional FourByteNodeId = structured
'       [Identifier] UInt16 = primitive(System.UInt16)
'       [NamespaceIndex] Byte = primitive(System.Byte)
'     [Guid] optional GuidNodeId = structured
'       [Identifier] Guid = primitive(System.Guid)
'       [NamespaceIndex] UInt16 = primitive(System.UInt16)
'     [NamespaceURI] optional CharArray = primitive(System.String)
'     [NamespaceURISpecified] switch Bit = primitive(System.Boolean)
'     [NodeIdType] switch NodeIdType = enumeration(6)
'       TwoByte = 0
'       FourByte = 1
'       Numeric = 2
'       String = 3
'       Guid = 4
'       ByteString = 5
'     [Numeric] optional NumericNodeId = structured
'       [Identifier] UInt32 = primitive(System.UInt32)
'       [NamespaceIndex] UInt16 = primitive(System.UInt16)
'     [ServerIndex] optional UInt32 = primitive(System.UInt32)
'     [ServerIndexSpecified] switch Bit = primitive(System.Boolean)
'     [String] optional StringNodeId = structured
'       [Identifier] CharArray = primitive(System.String)
'       [NamespaceIndex] UInt16 = primitive(System.UInt16)
'     [TwoByte] optional TwoByteNodeId = structured
'       [Identifier] Byte = primitive(System.Byte)
'   [FloatValue] Float = primitive(System.Single)
'   [GuidValue] Guid = primitive(System.Guid)
'   [Int16Value] Int16 = primitive(System.Int16)
'   [Int32Value] Int32 = primitive(System.Int32)
'   [Int64Value] Int64 = primitive(System.Int64)
'   [Integer] Variant = structured
'     [ArrayDimensions] optional sequence[*] of Int32 = primitive(System.Int32)
'     [ArrayDimensionsSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
'     [ArrayLength] length optional Int32 = primitive(System.Int32)
'     [ArrayLengthSpecified] switch sequence[1] of Bit = primitive(System.Boolean)
'     [Boolean] optional sequence[*] of Boolean = primitive(System.Boolean)
'     [Byte] optional sequence[*] of Byte = primitive(System.Byte)
```

13.2.8.9 Examples - OPC UA Complex Data - Retrieve sub-types of a data type

The example below determines sub-types of a a given data type:

C#

```
// This example shows how to retrieve all sub-types of a given data type, recursively.

using System;
using OpcLabs.BaseLib.OperationModel.Generic;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.Extensions;

namespace UADocExamples._EasyUAClient
{
    class RetrieveAllDataTypes
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Retrieve all sub-types of the 'Structure' data type.
            ValueResult<UANodeDescriptor[]> result = client.RetrieveAllDataTypes(
                endpointDescriptor, UADataTypeIds.Structure);
            // Check if the operation succeeded. Use the ThrowIfFailed method instead
if you want exception be thrown.
            if (!result.Succeeded)
            {
                Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief);
                return;
            }

            // Display results. Note that all node descriptors have node IDs in them;
but the default display format shows
            // the browse paths, which are more readable, when they are available.
            foreach (UANodeDescriptor nodeDescriptor in result.Value)
                Console.WriteLine(nodeDescriptor);

            // Example output:
            //
            //NodeId="Structure"
```

```
//BrowsePath="[Structure]/Argument"
//BrowsePath="[Structure]/StatusResult"
//BrowsePath="[Structure]/UserTokenPolicy"
//BrowsePath="[Structure]/ApplicationDescription"
//BrowsePath="[Structure]/EndpointDescription"
//BrowsePath="[Structure]/UserIdentityToken"
//BrowsePath="[Structure]/EndpointConfiguration"
//BrowsePath="[Structure]/SupportedProfile"
//BrowsePath="[Structure]/BuildInfo"
//BrowsePath="[Structure]/SoftwareCertificate"
//BrowsePath="[Structure]/SignedSoftwareCertificate"
//BrowsePath="[Structure]/AddNodesItem"
//BrowsePath="[Structure]/AddReferencesItem"
//BrowsePath="[Structure]/DeleteNodesItem"
//BrowsePath="[Structure]/DeleteReferencesItem"
//BrowsePath="[Structure]/ScalarTestType"
//BrowsePath="[Structure]/ArrayTestType"
//BrowsePath="[Structure]/CompositeTestType"
//BrowsePath="[Structure]/RegisteredServer"
//BrowsePath="[Structure]/ContentFilterElement"
//BrowsePath="[Structure]/ContentFilter"
//BrowsePath="[Structure]/FilterOperand"
//BrowsePath="[Structure]/HistoryEvent"
//BrowsePath="[Structure]/MonitoringFilter"
//BrowsePath="[Structure]/RedundantServerDataType"
//BrowsePath="[Structure]/SamplingIntervalDiagnosticsDataType"
//BrowsePath="[Structure]/ServerDiagnosticsSummaryDataType"
//BrowsePath="[Structure]/ServerStatusDataType"
//BrowsePath="[Structure]/SessionDiagnosticsDataType"
//BrowsePath="[Structure]/SessionSecurityDiagnosticsDataType"
//BrowsePath="[Structure]/ServiceCounterDataType"
//BrowsePath="[Structure]/SubscriptionDiagnosticsDataType"
//BrowsePath="[Structure]/ModelChangeStructureDataType"
//BrowsePath="[Structure]/Range"
//BrowsePath="[Structure]/EUInformation"
//BrowsePath="[Structure]/Annotation"
//BrowsePath="[Structure]/ProgramDiagnosticDataType"
//BrowsePath="[Structure]/SemanticChangeStructureDataType"
//BrowsePath="[Structure]/HistoryEventFieldList"
//BrowsePath="[Structure]/AggregateConfiguration"
//BrowsePath="[Structure]/EnumValueType"
//BrowsePath="[Structure]/TimeZoneDataType"
//BrowsePath="[Structure]/EndpointUrlListDataType"
//BrowsePath="[Structure]/NetworkGroupDataType"
//BrowsePath="[Structure]/AxisInformation"
//BrowsePath="[Structure]/XVType"
//BrowsePath="[Structure]/ComplexNumberType"
//BrowsePath="[Structure]/DoubleComplexNumberType"
//BrowsePath="[Structure]/ScalarValueDataType"
//BrowsePath="[Structure]/ArrayValueDataType"
//BrowsePath="[Structure]/UserScalarValueDataType"
//BrowsePath="[Structure]/UserArrayValueDataType"
//BrowsePath="[Structure]/UserIdentityToken/AnonymousIdentityToken"
//BrowsePath="[Structure]/UserIdentityToken/UserNameIdentityToken"
//BrowsePath="[Structure]/UserIdentityToken/X509IdentityToken"
//BrowsePath="[Structure]/UserIdentityToken/IssuedIdentityToken"
//BrowsePath="[Structure]/FilterOperand/ElementOperand"
```

```
            //BrowsePath="[Structure]/FilterOperand/LiteralOperand"
            //BrowsePath="[Structure]/FilterOperand/AttributeOperand"
            //BrowsePath="[Structure]/FilterOperand/SimpleAttributeOperand"
            //BrowsePath="[Structure]/MonitoringFilter/EventFilter"
        }
    }
}
```

VB.NET

```
' This example shows how to retrieve all sub-types of a given data type, recursively.

Imports OpcLabs.BaseLib.OperationModel.Generic
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.Extensions

Namespace UADocExamples._EasyUAClient
    Friend Class RetrieveAllDataTypes
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Retrieve all sub-types of the 'Structure' data type.
            Dim result As ValueResult(Of UANodeDescriptor()) =
client.RetrieveAllDataTypes( _
                endpointDescriptor, UADataTypeIds.Structure)
            ' Check if the operation succeeded. Use the ThrowIfFailed method instead if
you want exception be thrown.
            If Not result.Succeeded Then
                Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief)
            End If

            ' Display results. Note that all node descriptors have node IDs in them;
but the default display format shows
            ' the browse paths, which are more readable, when they are available.
            For Each nodeDescriptor As UANodeDescriptor In result.Value
                Console.WriteLine(nodeDescriptor)
            Next nodeDescriptor

            ' Example output:
            '
            'NodeId="Structure"
            'BrowsePath="[Structure]/Argument"
            'BrowsePath="[Structure]/StatusResult"
            'BrowsePath="[Structure]/UserTokenPolicy"
            'BrowsePath="[Structure]/ApplicationDescription"
            'BrowsePath="[Structure]/EndpointDescription"
            'BrowsePath="[Structure]/UserIdentityToken"
            'BrowsePath="[Structure]/EndpointConfiguration"
```

```
            'BrowsePath="[Structure]/SupportedProfile"
            'BrowsePath="[Structure]/BuildInfo"
            'BrowsePath="[Structure]/SoftwareCertificate"
            'BrowsePath="[Structure]/SignedSoftwareCertificate"
            'BrowsePath="[Structure]/AddNodesItem"
            'BrowsePath="[Structure]/AddReferencesItem"
            'BrowsePath="[Structure]/DeleteNodesItem"
            'BrowsePath="[Structure]/DeleteReferencesItem"
            'BrowsePath="[Structure]/ScalarTestType"
            'BrowsePath="[Structure]/ArrayTestType"
            'BrowsePath="[Structure]/CompositeTestType"
            'BrowsePath="[Structure]/RegisteredServer"
            'BrowsePath="[Structure]/ContentFilterElement"
            'BrowsePath="[Structure]/ContentFilter"
            'BrowsePath="[Structure]/FilterOperand"
            'BrowsePath="[Structure]/HistoryEvent"
            'BrowsePath="[Structure]/MonitoringFilter"
            'BrowsePath="[Structure]/RedundantServerDataType"
            'BrowsePath="[Structure]/SamplingIntervalDiagnosticsDataType"
            'BrowsePath="[Structure]/ServerDiagnosticsSummaryDataType"
            'BrowsePath="[Structure]/ServerStatusDataType"
            'BrowsePath="[Structure]/SessionDiagnosticsDataType"
            'BrowsePath="[Structure]/SessionSecurityDiagnosticsDataType"
            'BrowsePath="[Structure]/ServiceCounterDataType"
            'BrowsePath="[Structure]/SubscriptionDiagnosticsDataType"
            'BrowsePath="[Structure]/ModelChangeStructureDataType"
            'BrowsePath="[Structure]/Range"
            'BrowsePath="[Structure]/EUInformation"
            'BrowsePath="[Structure]/Annotation"
            'BrowsePath="[Structure]/ProgramDiagnosticDataType"
            'BrowsePath="[Structure]/SemanticChangeStructureDataType"
            'BrowsePath="[Structure]/HistoryEventFieldList"
            'BrowsePath="[Structure]/AggregateConfiguration"
            'BrowsePath="[Structure]/EnumValueType"
            'BrowsePath="[Structure]/TimeZoneDataType"
            'BrowsePath="[Structure]/EndpointUrlListDataType"
            'BrowsePath="[Structure]/NetworkGroupDataType"
            'BrowsePath="[Structure]/AxisInformation"
            'BrowsePath="[Structure]/XVType"
            'BrowsePath="[Structure]/ComplexNumberType"
            'BrowsePath="[Structure]/DoubleComplexNumberType"
            'BrowsePath="[Structure]/ScalarValueDataType"
            'BrowsePath="[Structure]/ArrayValueDataType"
            'BrowsePath="[Structure]/UserScalarValueDataType"
            'BrowsePath="[Structure]/UserArrayValueDataType"
            'BrowsePath="[Structure]/UserIdentityToken/AnonymousIdentityToken"
            'BrowsePath="[Structure]/UserIdentityToken/UserNameIdentityToken"
            'BrowsePath="[Structure]/UserIdentityToken/X509IdentityToken"
            'BrowsePath="[Structure]/UserIdentityToken/IssuedIdentityToken"
            'BrowsePath="[Structure]/FilterOperand/ElementOperand"
            'BrowsePath="[Structure]/FilterOperand/LiteralOperand"
            'BrowsePath="[Structure]/FilterOperand/AttributeOperand"
            'BrowsePath="[Structure]/FilterOperand/SimpleAttributeOperand"
            'BrowsePath="[Structure]/MonitoringFilter/EventFilter"
        End Sub
    End Class
End Namespace
```

13.2.8.10 Examples - OPC UA Complex Data - Subscribe to data change

The example below shows how to subscribe to OPC UA complex data and display the incoming values.

C#

```
// Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._EasyUAClient
{
    class SubscribeDataChange
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10867"; //
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

            // Instantiate the client object and hook the event handler.
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            // Subscribe to a node which returns complex data. This is done in the same
way as regular subscribes - just
            // the data returned is different.
            Console.WriteLine("Subscribing...");
            client.SubscribeDataChange(endpointDescriptor, nodeDescriptor,
samplingInterval:1000);

            Console.WriteLine("Processing data change events for 20 seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            // Unhook the event handler.
            client.DataChangeNotification -= client_DataChangeNotification;
```

```
            // Example output:
            //
            //Subscribing...
            //Processing data change events for 20 seconds...
            //(ScalarValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ScalarValueDataType) structured
            //(ScalarValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ScalarValueDataType) structured
            //(ArrayValueDataType) structured
            //(ArrayValueDataType) structured
            //(ScalarValueDataType) structured
            //(ScalarValueDataType) structured
            //(ScalarValueDataType) structured
            //Unsubscribing...
            //Waiting for 5 seconds...
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("Value: {0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);

            // For processing the internals of the data, refer to examples for
GenericData and DataType classes.
        }
    }
}
```

Object Pascal

```
// Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

type
  TClientEventHandlers33 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers33.OnDataChangeNotification(
  ASender: TObject;
```

```
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display value
  if eventArgs.Succeeded then
      WriteLn('Value: ', eventArgs.AttributeData.Value.ToString)
  else
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);

  // For processing the internals of the data, refer to examples for GenericData and
DataType classes.
end;

class procedure SubscribeDataChange.Main;
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers33;
  EndpointDescriptor: string;
  NodeDescriptor: string;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10867';  //
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers33.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  // Subscribe to a node which returns complex data. This is done in the same way as
regular subscribes - just
  // the data returned is different.
  WriteLn('Subscribing...');
  Client.SubscribeDataChange(EndpointDescriptor, NodeDescriptor, 1000);

  WriteLn('Processing data change events for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);

  // Example output:
  //
  //Subscribing...
  //Processing data change events for 20 seconds...
  //(ScalarValueDataType) structured
```

```
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ScalarValueDataType) structured
  //(ScalarValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ScalarValueDataType) structured
  //(ArrayValueDataType) structured
  //(ArrayValueDataType) structured
  //(ScalarValueDataType) structured
  //(ScalarValueDataType) structured
  //(ScalarValueDataType) structured
  //Unsubscribing...
  //Waiting for 5 seconds...
end;
```

PHP

```
// Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display value
        if ($E->Succeeded)
        printf("Value: %s\n", $E->AttributeData->Value);
        else
        printf("*** Failure : %s\n", $E->ErrorMessageBrief);
        // For processing the internals of the data, refer to examples for GenericData
and DataType classes.
    }
}

// Define which server and node we will work with.
$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
$NodeDescriptor = "nsu=http://test.org/UA/Data/;i=10867";  //
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

// Subscribe to a node which returns complex data. This is done in the same way as
regular subscribes - just
// the data returned is different.
printf("Subscribing...\n");
$Client->SubscribeDataChange($EndpointDescriptor, $NodeDescriptor, 1000);

printf("Processing data change events for 20 seconds...\n");
```

```
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 20);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

VB.NET

```
' Shows how to subscribe to complex data with OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._EasyUAClient

    Friend Class SubscribeDataChange

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10867"  '
[ObjectsFolder]/Data.Dynamic.Scalar.StructureValue

            ' Instantiate the client object and hook the event handler.
            Dim client = New EasyUAClient
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            ' Subscribe to a node which returns complex data. This is done in the same
way as regular subscribes - just
            ' the data returned is different.
            Console.WriteLine("Subscribing...")
            client.SubscribeDataChange(endpointDescriptor, nodeDescriptor,
samplingInterval:=1000)

            Console.WriteLine("Processing data change events for 20 seconds...")
            Threading.Thread.Sleep((20 * 1000))

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep((5 * 1000))

            ' Unhook the event handler.
            RemoveHandler client.DataChangeNotification, AddressOf
```

```
client_DataChangeNotification


            ' Example output:
            '
            'Subscribing...
            'Processing data change events for 20 seconds...
            '(ScalarValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ScalarValueDataType) structured
            '(ScalarValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ScalarValueDataType) structured
            '(ArrayValueDataType) structured
            '(ArrayValueDataType) structured
            '(ScalarValueDataType) structured
            '(ScalarValueDataType) structured
            '(ScalarValueDataType) structured
            'Unsubscribing...
            'Waiting for 5 seconds...
        End Sub

        Private Shared Sub client_DataChangeNotification(sender As Object, e As
EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If

            ' For processing the internals of the data, refer to examples for
GenericData and DataType classes.
        End Sub
    End Class
End Namespace
```

13.2.8.11 Examples - OPC UA Complex Data - Use a shared data type model provider

C#

```
// Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.Plugins.ComplexData;

namespace UADocExamples.ComplexData._UAComplexDataPluginParameters
{
```

```
    class IsolatedDataTypeModelProvider
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; // [ObjectsFolder]/Data.Static.Scalar.StructureValue


            // We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
            // model provider.

            // Configure the first client object.
            var client1 = new EasyUAClient();
            UAComplexDataPluginParameters complexDataPluginParameters1 =
                client1.InstanceParameters.PluginConfigurations.Find<UAComplexDataPluginParameters>();
            Debug.Assert(complexDataPluginParameters1 != null);
            complexDataPluginParameters1.IsolatedDataTypeModelProvider = false;

            // Configure the second client object.
            var client2 = new EasyUAClient();
            UAComplexDataPluginParameters complexDataPluginParameters2 =
                client2.InstanceParameters.PluginConfigurations.Find<UAComplexDataPluginParameters>();
            Debug.Assert(complexDataPluginParameters2 != null);
            complexDataPluginParameters2.IsolatedDataTypeModelProvider = false;


            // We will now read the same complex data node using the two client objects.
            //
            // There is no noticeable difference in the results from the default state in which the client objects are
            // set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
            // obtained during the read on the first client object and cached inside the data type model provider are reused
            // during the read on the second client object, making this and the subsequent operations more efficient.

            // Read the complex data node using the first client.
            object value1;
            try
            {
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine(value1);

            // Read the complex data node using the second client.
            object value2;
            try
            {
                value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine(value2);
        }
    }
}
```

Object Pascal

```
// Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

class procedure IsolatedDataTypeModelProvider.Main;
var
  Client1, Client2: _EasyUAClient;
  ComplexDataPluginParameters1, ComplexDataPluginParameters2: OpcLabs_EasyOpcUA_TLB._UAComplexDataPluginParameters;
  EndpointDescriptor: string;
  NodeDescriptor: string;
  Value1, Value2: OleVariant;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
    'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  // [ObjectsFolder]/Data.Static.Scalar.StructureValue
```

```
  // We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
  // model provider.

  // Configure the first client object.
  Client1 := CoEasyUAClient.Create;
  ComplexDataPluginParameters1 :=
IUnknown(Client1.InstanceParameters.PluginConfigurations.Find('OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters'))
as OpcLabs_EasyOpcUA_TLB._UAComplexDataPluginParameters;
  ComplexDataPluginParameters1.IsolatedDataTypeModelProvider := false;

  // Configure the second client object.
  Client2 := CoEasyUAClient.Create;
  ComplexDataPluginParameters2 :=
IUnknown(Client2.InstanceParameters.PluginConfigurations.Find('OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters'))
as OpcLabs_EasyOpcUA_TLB._UAComplexDataPluginParameters;
  ComplexDataPluginParameters2.IsolatedDataTypeModelProvider := false;

  // We will now read the same complex data node using the two client objects.
  //
  // There is no noticeable difference in the results from the default state in which the client objects are
  // set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
  // obtained during the read on the first client object and cached inside the data type model provider are reused
  // during the read on the second client object, making this and the subsequent operations more efficient.

  // Read the complex data node using the first client.
  try
    Value1 := Client1.ReadValue(EndpointDescriptor, NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
  WriteLn(Value1);

  // Read the complex data node using the second client.
  try
    Value2 := Client2.ReadValue(EndpointDescriptor, NodeDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;
  WriteLn(Value2);

end;
```

VB.NET

```
' Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.Plugins.ComplexData

Namespace UADocExamples.ComplexData._UAComplexDataPluginParameters

    Friend Class IsolatedDataTypeModelProvider

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue


            ' We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
            ' model provider.

            ' Configure the first client object.
            Dim client1 = New EasyUAClient
            Dim complexDataPluginParameters1 As UAComplexDataPluginParameters = _
                client1.InstanceParameters.PluginConfigurations.Find(Of UAComplexDataPluginParameters)()
            Debug.Assert(complexDataPluginParameters1 IsNot Nothing)
            complexDataPluginParameters1.IsolatedDataTypeModelProvider = False
```

```
            ' Configure the second client object.
            Dim client2 = New EasyUAClient
            Dim complexDataPluginParameters2 As UAComplexDataPluginParameters = _
                client2.InstanceParameters.PluginConfigurations.Find(Of UAComplexDataPluginParameters)()
            Debug.Assert(complexDataPluginParameters2 IsNot Nothing)
            complexDataPluginParameters2.IsolatedDataTypeModelProvider = False


            ' We will now read the same complex data node using the two client objects.
            '
            ' There is no noticeable difference in the results from the default state in which the client objects are
            ' set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
            ' obtained during the read on the first client object and cached inside the data type model provider are reused
            ' during the read on the second client object, making this and the subsequent operations more efficient.

            ' Read the complex data node using the first client.
            Dim value1 As Object
            Try
                value1 = client1.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value1)

            ' Read the complex data node using the second client.
            Dim value2 As Object
            Try
                value2 = client2.ReadValue(endpointDescriptor, nodeDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try
            Console.WriteLine(value2)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to configure the OPC UA Complex Data plug-in to use a shared data type model provider.

Option Explicit

' Define which server and node we will work with.
Dim endpointDescriptor: endpointDescriptor = _
    "http://opcua.demo-this.com:51211/UA/SampleServer"
'or "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
Dim nodeDescriptor: nodeDescriptor = _
    "nsu=http://test.org/UA/Data/;i=10239"  ' [ObjectsFolder]/Data.Static.Scalar.StructureValue


' We will create two instances of EasyUAClient class, and configure each of them to use the shared data type
' model provider.

' Configure the first client object.
Dim Client1: Set Client1 = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ComplexDataPluginParameters1: Set ComplexDataPluginParameters1 = Client1.InstanceParameters.PluginConfigurations.Find( _
    "OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters")
ComplexDataPluginParameters1.IsolatedDataTypeModelProvider = False

' Configure the second client object.
Dim Client2: Set Client2 = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ComplexDataPluginParameters2: Set ComplexDataPluginParameters2 = Client2.InstanceParameters.PluginConfigurations.Find( _
    "OpcLabs.EasyOpc.UA.Plugins.ComplexData.UAComplexDataPluginParameters")
ComplexDataPluginParameters2.IsolatedDataTypeModelProvider = False

' We will now read the same complex data node using the two client objects.
'
' There is no noticeable difference in the results from the default state in which the client objects are
' set to use per-instance data type model provider. But, with the shared data type model provider, the metadata
' obtained during the read on the first client object and cached inside the data type model provider are reused
' during the read on the second client object, making this and the subsequent operations more efficient.

' Read the complex data node using the first client.
On Error Resume Next
Dim Value1: Set Value1 = Client1.ReadValue(endpointDescriptor, nodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo Value1

' Read the complex data node using the second client.
```

```
On Error Resume Next
Dim Value2: Set Value2 = Client2.ReadValue(endpointDescriptor, nodeDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo Value2
```

13.2.8.12 Examples - OPC UA Complex Data - Write a value

The example below first reads a structured value from the server, and then manipulates (changes) its elements. The modified value is then written to the server. In this case, the data type ID (and also the data type) can be taken over from what has been provided by the Read.

C#

```
// Shows how to write complex data with OPC UA Complex Data plug-in.

using System;
using OpcLabs.BaseLib.DataTypeModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.ComplexData;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.ComplexData._EasyUAClient
{
    class WriteValue
    {
        public static void Main1()
        {
            // Define which server and node we will work with.
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor =
                "nsu=http://test.org/UA/Data/;i=10239"; //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            // Instantiate the client object.
            var client = new EasyUAClient();

            // Read a node which returns complex data.
            // We know that this node returns complex data, so we can type cast to
UAGenericObject.
            Console.WriteLine("Reading...");
            UAGenericObject genericObject;
            try
            {
                genericObject = (UAGenericObject)client.ReadValue(endpointDescriptor,
nodeDescriptor);
            }
            catch (UAException uaException)
            {
```

```
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }


            // Modify the data read.
            // This node returns one of the two data types, randomly (this is not
common, usually the type is fixed). The
            // data types are sub-types of one common type which the data type of the
node. We therefore use the data type
            // ID in the returned UAGenericObject to detect which data type has been
returned.

            // For processing the internals of the data, refer to examples for
GenericData and DataType classes.
            // We know how the data is structured, and have hard-coded a logic that
modifies certain values inside. It is
            // also possible to discover the structure of the data type in the program,
and write generic clients that can
            // cope with any kind of complex data.
            //
            // Note that the code below is not fully robust - it will throw an
exception if the data is not as expected.
            Console.WriteLine("Modifying...");
            Console.WriteLine(genericObject.DataTypeId);
            if
(genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9440"))
// ScalarValueDataType
            {
                // Negate the byte in the "ByteValue" field.
                var structuredData = (StructuredData)genericObject.GenericData;
                var byteValue = (PrimitiveData)structuredData.FieldData["ByteValue"];
                byteValue.Value = (Byte)~((Byte)byteValue.Value);
                Console.WriteLine(byteValue.Value);
            }
            else if
(genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9669"))
// ArrayValueDataType
            {
                // Negate bytes at indexes 0 and 1 of the array in the "ByteValue"
field.
                var structuredData = (StructuredData)genericObject.GenericData;
                var byteValue = (SequenceData)structuredData.FieldData["ByteValue"];
                var element0 = (PrimitiveData)byteValue.Elements[0];
                var element1 = (PrimitiveData)byteValue.Elements[1];
                element0.Value = (Byte)~((Byte)element0.Value);
                element1.Value = (Byte)~((Byte)element1.Value);
                Console.WriteLine(element0.Value);
                Console.WriteLine(element1.Value);
            }


            // Write the modified complex data back to the node.
            // The data type ID in the UAGenericObject is borrowed without change from
what we have read, so that the server
            // knows which data type we are writing. The data type ID not necessary if
```

```
writing precisely the same data type
            // as the node has (not a subtype).
            Console.WriteLine("Writing...");
            try
            {
                client.WriteValue(endpointDescriptor, nodeDescriptor, genericObject);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

Object Pascal

```
// Shows how to write complex data with OPC UA Complex Data plug-in.

class procedure WriteValue.Main;
var
  ArrayValueDataType: _UANodeDescriptor;
  ByteValue: _PrimitiveData;
  ByteValue2: _SequenceData;
  Client: _EasyUAClient;
  Element0, Element1: _PrimitiveData;
  EndpointDescriptor: string;
  GenericObject: _UAGenericObject;
  NodeDescriptor: string;
  ScalarValueDataType: _UANodeDescriptor;
  StructuredData: _StructuredData;
begin
  // Define which server and node we will work with.
  EndpointDescriptor :=
   'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  //or 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  NodeDescriptor := 'nsu=http://test.org/UA/Data/;i=10239';  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Read a node which returns complex data.
  // We know that this node returns complex data, so we can type cast to
UAGenericObject.
  WriteLn('Reading...');

  try
    GenericObject := _UAGenericObject(IUnknown(Client.ReadValue(EndpointDescriptor,
NodeDescriptor)));
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
```

```
    end;
  end;

  // Modify the data read.
  // This node returns one of the two data types, randomly (this is not common, usually
the type is fixed). The
  // data types are sub-types of one common type which the data type of the node. We
therefore use the data type
  // ID in the returned UAGenericObject to detect which data type has been returned.

  // For processing the internals of the data, refer to examples for GenericData and
DataType classes.
  // We know how the data is structured, and have hard-coded a logic that modifies
certain values inside. It is
  // also possible to discover the structure of the data type in the program, and write
generic clients that can
  // cope with any kind of complex data.
  //
  // Note that the code below is not fully robust - it will throw an exception if the
data is not as expected.

  WriteLn('Modifying...');
  WriteLn(GenericObject.DataTypeId.ToString);
  ScalarValueDataType := CoUANodeDescriptor.Create;
  ScalarValueDataType.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=9440'; //
ScalarValueDataType
  if GenericObject.DataTypeId.NodeDescriptor.Match(ScalarValueDataType) then
  begin
    // Negate the byte in the "ByteValue" field.
    StructuredData := IUnknown(GenericObject.GenericData) as _StructuredData;
    ByteValue := IUnknown(StructuredData.FieldData['ByteValue']) as _PrimitiveData;
    ByteValue.Value := Byte(not (Byte(byteValue.Value)));
    WriteLn(ByteValue.Value);
  end
  else
  begin
    ArrayValueDataType := CoUANodeDescriptor.Create;
    ArrayValueDataType.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=9669'; //
ArrayValueDataType
    if GenericObject.DataTypeId.Nodedescriptor.Match(ArrayValueDataType) then
    begin
      // Negate bytes at indexes 0 and 1 of the array in the "ByteValue" field.
      StructuredData := IUnknown(GenericObject.GenericData) as _StructuredData;
      ByteValue2 := IUnknown(StructuredData.FieldData['ByteValue']) as _SequenceData;
      Element0 := IUnknown(ByteValue2.Elements[0]) as _PrimitiveData;
      Element1 := IUnknown(ByteValue2.Elements[1]) as _PrimitiveData;
      Element0.Value := Byte(not (Byte(element0.Value)));
      Element1.Value := Byte(not (Byte(element1.Value)));
      WriteLn(Element0.Value);
      WriteLn(Element1.Value);
    end;
  end;

  // Write the modified complex data back to the node.
  // The data type ID in the UAGenericObject is borrowed without change from what we
have read, so that the server
  // knows which data type we are writing. The data type ID not necessary if writing
```

```
precisely the same data type
  // as the node has (not a subtype).
  WriteLn('Writing...');
  try
    Client.WriteValue(EndpointDescriptor, NodeDescriptor, GenericObject);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PHP

```
// Shows how to write complex data with OPC UA Complex Data plug-in.

// Define which server and node we will work with.
$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
$NodeDescriptor = "nsu=http://test.org/UA/Data/;i=10239";  //
[ObjectsFolder]/Data.Static.Scalar.StructureValue

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Read a node which returns complex data.
// We know that this node returns complex data, so we can type cast to UAGenericObject.
printf("Reading...\n");

try
{
    $GenericObject = $Client->ReadValue($EndpointDescriptor, $NodeDescriptor);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Modify the data read.
// This node returns one of the two data types, randomly (this is not common, usually
the type is fixed). The
// data types are sub-types of one common type which the data type of the node. We
therefore use the data type
// ID in the returned UAGenericObject to detect which data type has been returned.

// For processing the internals of the data, refer to examples for GenericData and
DataType classes.
// We know how the data is structured, and have hard-coded a logic that modifies
certain values inside. It is
// also possible to discover the structure of the data type in the program, and write
generic clients that can
// cope with any kind of complex data.
```

```
//
// Note that the code below is not fully robust - it will throw an exception if the
data is not as expected.

printf("Modifying...\n");
printf("%s\n", $GenericObject->DataTypeId);
$ScalarValueDataType = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
$ScalarValueDataType->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=9440"; //
ScalarValueDataType
if ($GenericObject->DataTypeId->NodeDescriptor->Match($ScalarValueDataType)) {
    // Negate the byte in the "ByteValue" field.
    $StructuredData = $GenericObject->GenericData->AsStructuredData();
    $ByteValue = $StructuredData->FieldData["ByteValue"]->AsPrimitiveData();
    $ByteValue->Value = ~($ByteValue->Value) & 255;
    printf("%s\n", $ByteValue->Value);
}
else {
    $ArrayValueDataType = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
    $ArrayValueDataType->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=9669";
// ArrayValueDataType
    if ($GenericObject->DataTypeId->Nodedescriptor->Match($ArrayValueDataType)) {
        // Negate bytes at indexes 0 and 1 of the array in the "ByteValue" field.
        $StructuredData = $GenericObject->GenericData->AsStructuredData();
        $ByteValue2 = $StructuredData->FieldData["ByteValue"]->AsSequenceData();
        $Element0 = $ByteValue2->Elements[0]->AsPrimitiveData();
        $Element1 = $ByteValue2->Elements[1]->AsPrimitiveData();
        $Element0->Value = ~($Element0->Value) & 255;
        $Element1->Value = ~($Element1->Value) & 255;
        printf("%s\n", $Element0->Value);
        printf("%s\n", $Element1->Value);
    }
}

// Write the modified complex data back to the node.
// The data type ID in the UAGenericObject is borrowed without change from what we have
read, so that the server
// knows which data type we are writing. The data type ID not necessary if writing
precisely the same data type
// as the node has (not a subtype).
printf("Writing...\n");
try
{
    $Client->WriteValue($EndpointDescriptor, $NodeDescriptor, $GenericObject);
}
catch (com_exception $e)
{
    printf("Failure: %s\n", $e->getMessage());
    Exit();
}
```

VB.NET

```
' Shows how to write complex data with OPC UA Complex Data plug-in.

Imports System
Imports OpcLabs.BaseLib.DataTypeModel
Imports OpcLabs.EasyOpc.UA
```

```
Imports OpcLabs.EasyOpc.UA.ComplexData
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.ComplexData._EasyUAClient

    Friend Class WriteValue

        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Define which node we will work with.
            Dim nodeDescriptor As UANodeDescriptor = _
                "nsu=http://test.org/UA/Data/;i=10239"  '
[ObjectsFolder]/Data.Static.Scalar.StructureValue

            ' Instantiate the client object.
            Dim client = New EasyUAClient

            ' Read a node which returns complex data.
            ' We know that this node returns complex data, so we can type cast to
UAGenericObject.
            Console.WriteLine("Reading...")
            Dim genericObject As UAGenericObject
            Try
                genericObject = CType(client.ReadValue(endpointDescriptor,
nodeDescriptor), UAGenericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try


            ' Modify the data read.
            ' This node returns one of the two data types, randomly (this is not
common, usually the type is fixed). The
            ' data types are sub-types of one common type which the data type of the
node. We therefore use the data type
            ' ID in the returned UAGenericObject to detect which data type has been
returned.
            ' For processing the internals of the data, refer to examples for
GenericData and DataType classes.
            ' We know how the data is structured, and have hard-coded a logic that
modifies certain values inside. It is
            ' also possible to discover the structure of the data type in the program,
and write generic clients that can
            ' cope with any kind of complex data.
            '
            ' Note that the code below is not fully robust - it will throw an exception
if the data is not as expected.
            Console.WriteLine("Modifying...")
```

```
            Console.WriteLine(genericObject.DataTypeId)
            If
genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9440")
Then    ' ScalarValueDataType
                ' Negate the byte in the "ByteValue" field.
                Dim structuredData = CType(genericObject.GenericData, StructuredData)
                Dim byteValue = CType(structuredData.FieldData("ByteValue"),
PrimitiveData)
                byteValue.Value = CType(Not CType(byteValue.Value, Byte), Byte)
                Console.WriteLine(byteValue.Value)
            ElseIf
genericObject.DataTypeId.NodeDescriptor.Match("nsu=http://test.org/UA/Data/;i=9669")
Then    ' ArrayValueDataType
                ' Negate bytes at indexes 0 and 1 of the array in the "ByteValue"
field.
                Dim structuredData = CType(genericObject.GenericData, StructuredData)
                Dim byteValue = CType(structuredData.FieldData("ByteValue"),
SequenceData)
                Dim element0 = CType(byteValue.Elements(0), PrimitiveData)
                Dim element1 = CType(byteValue.Elements(1), PrimitiveData)
                element0.Value = CType(Not CType(element0.Value, Byte), Byte)
                element1.Value = CType(Not CType(element1.Value, Byte), Byte)
                Console.WriteLine(element0.Value)
                Console.WriteLine(element1.Value)
            End If


            ' Write the modified complex data back to the node.
            ' The data type ID in the UAGenericObject is borrowed without change from
what we have read, so that the server
            ' knows which data type we are writing. The data type ID not necessary if
writing precisely the same data type
            ' as the node has (not a subtype).
            Console.WriteLine("Writing...")
            Try
                client.WriteValue(endpointDescriptor, nodeDescriptor, genericObject)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

13.2.9 Examples - OPC UA GDS

13.2.9.1 Examples - OPC UA GDS - Check certificate status

C#

```
// Shows how to check if an application needs to update its certificate.
```

```
using System;
using OpcLabs.BaseLib.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Application;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Gds._EasyUACertificateManagementClient
{
    class GetCertificateStatus
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Register our client application with the GDS, so that we obtain an
application ID that we need later.
            var client = new EasyUAClient();
            IEasyUAClientApplication clientApplication =
client.GetService<IEasyUAClientApplication>();
            UANodeId applicationId;
            try
            {
                applicationId = clientApplication.RegisterToGds(gdsEndpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }
            Console.WriteLine("Application ID: {0}", applicationId);

            // Instantiate the certificate management client object
            var certificateManagementClient = new EasyUACertificateManagementClient();

            // Check if the application needs to update its certificate.
            bool updateRequired;
            try
            {
                updateRequired =
certificateManagementClient.GetCertificateStatus(gdsEndpointDescriptor, applicationId,
                    UANodeId.Null, UANodeId.Null);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
```

```
            Console.WriteLine("Update required: {0}", updateRequired);
        }


        // Example output:
        //Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=aec94459-f513-4979-8619-8383555fca61
        //Update required: False
    }
}
```

Object Pascal

```
// Shows how to check if an application needs to update its certificate.

class procedure GetCertificateStatus.Main;
var
  ApplicationId: _UANodeId;
  CertificateManagementClient: OpcLabs_EasyOpcUA_TLB._EasyUACertificateManagementClient;
  Client: _EasyUAClient;
  ClientApplication: _EasyUAClientApplication;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  NullNodeId: _UANodeId;
  UpdateRequired: boolean;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Register our client application with the GDS, so that we obtain an application ID that
we need later.
  Client := CoEasyUAClient.Create;
  ClientApplication :=
IInterface(Client.GetServiceByName('OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA')) as _EasyUAClientApplication;

  // Create an application registration in the GDS, assigning it a new application ID.
  try
    ApplicationId := ClientApplication.RegisterToGds(GdsEndpointDescriptor);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Application ID: ', ApplicationId.ToString);

  // Instantiate the certificate management client object
  CertificateManagementClient := CoEasyUACertificateManagementClient.Create;

  // Check if the application needs to update its certificate.
  NullNodeId := CoUANodeId.Create;
  UpdateRequired := false;
  try
    UpdateRequired :=
```

```
CertificateManagementClient.GetCertificateStatus(GdsEndpointDescriptor, ApplicationId,
NullNodeId, NullNodeId);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  WriteLn('Update required: ', UpdateRequired);

  // Example output:
  //Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=aec94459-
f513-4979-8619-8383555fca61
  //Update required: FALSE

end;
```

VB.NET

```
' Shows how to check if an application needs to update its certificate.

Imports OpcLabs.BaseLib.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Application
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUACertificateManagementClient
    Friend Class GetCertificateStatus
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Register our client application with the GDS, so that we obtain an
application ID that we need later.
            Dim client = New EasyUAClient()
            Dim clientApplication = client.GetService(Of IEasyUAClientApplication)()

            ' Create an application registration in the GDS, assigning it a new application
ID.
            Dim applicationId As UANodeId
            Try
                applicationId = clientApplication.RegisterToGds(gdsEndpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Instantiate the certificate management client object
            Dim certificateManagementClient = New EasyUACertificateManagementClient()
```

```
            ' Check if the application needs to update its certificate.
            Dim updateRequired As Boolean
            Try
                updateRequired =
certificateManagementClient.GetCertificateStatus(gdsEndpointDescriptor, applicationId,
                    UANodeId.Null, UANodeId.Null)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("Update required: {0}", updateRequired)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to check if an application needs to update its certificate.

Option Explicit

' Define which GDS we will work with.
Dim GdsEndpointDescriptor: Set GdsEndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
GdsEndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName = "appadmin"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password = "demo"

' Register our client application with the GDS, so that we obtain an application ID that we
need later.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim ClientApplication: Set ClientApplication = _
    Client.GetServiceByName("OpcLabs.EasyOpc.UA.Application.IEasyUAClientApplication,
OpcLabs.EasyOpcUA")
On Error Resume Next
Dim ApplicationId: Set ApplicationId =
ClientApplication.RegisterToGds(GdsEndpointDescriptor)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
WScript.Echo "Application ID: " & ApplicationId

' Instantiate the certificate management client object
Dim CertificateManagementClient: Set CertificateManagementClient = _
    CreateObject("OpcLabs.EasyOpc.UA.Gds.EasyUACertificateManagementClient")

' Check if the application needs to update its certificate.
Dim NullNodeId: Set NullNodeId = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
On Error Resume Next
Dim UpdateRequired: UpdateRequired = CertificateManagementClient.GetCertificateStatus( _
    GdsEndpointDescriptor, ApplicationId, NullNodeId, NullNodeId)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
```

```
End If
On Error Goto 0

' Display results
WScript.Echo "Update required: " & UpdateRequired


' Example output:
'Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=aec94459-f513-
4979-8619-8383555fca61
'Update required: False
```

13.2.9.2 Examples - OPC UA GDS - Find all registrations

C#

```
// Shows how to find all registrations in the GDS.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
{
    class FindApplications
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appuser", "demo");

            // Instantiate the global discovery client object
            var globalDiscoveryClient = new EasyUAGlobalDiscoveryClient();

            // Find all (client or server) applications registered in the GDS.
            UAApplicationDescription[] applicationDescriptionArray;
            try
            {
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor: gdsEndpointDescriptor,
                    startingRecordId: 0,
                    maximumRecordsToReturn: 0,
                    applicationName: "",
                    applicationUriString: "",
                    applicationTypes: UAApplicationTypes.All,
                    productUriString: "",
                    serverCapabilities: new string[0],
                    lastCounterResetTime: out _,
```

```
                    nextRecordId: out _,
                    applications: out applicationDescriptionArray);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // For each application returned by the query, find its registrations in
the GDS.
            foreach (UAApplicationDescription applicationDescription in
applicationDescriptionArray)
            {
                Console.WriteLine();
                Console.WriteLine("Application URI string: {0}",
applicationDescription.ApplicationUriString);

                UAApplicationRecordDataType[] applicationRecordArray;
                try
                {
                    applicationRecordArray = globalDiscoveryClient.FindApplications(
                        gdsEndpointDescriptor,
                        applicationDescription.ApplicationUriString);
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("  *** Failure: {0}",
uaException.GetBaseException().Message);
                    continue;
                }

                // Display results
                foreach (UAApplicationRecordDataType applicationRecord in
applicationRecordArray)
                    Console.WriteLine("  Application ID: {0}",
applicationRecord.ApplicationId);
            }


            // Example output:
            //
            //Application URI string: urn:sampleserver
            //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=09ecaa08-6ec6-462c-a214-1e66a3099107
            //
            //Application URI string: urn:alarmconditionserver
            //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=783e1e9a-8036-43b6-928f-97488c460266
            //
            //Application URI string:
urn:PC:MultiTargetUADocExamples:5.54.1026.1:neutral:null
            //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=9e700ea5-55a6-4c3c-ba9f-b91c890dc519
            //
            //Application URI string: urn:PC:UADocExamples:5.56.0.16:neutral:null
```

```
            //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=e182e28c-086b-4fc7-82c7-70ca7cda3033
            //
            //Application URI string: urn:PC:cscript:5.812.10240.16384
            //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=aec94459-f513-4979-8619-8383555fca61
        }
    }
}
```

Object Pascal

```
// Shows how to find all registrations in the GDS.

class procedure FindApplications.Main;
var
  ApplicationDescription: _UAApplicationDescription;
  ApplicationDescriptionArray: OleVariant;
  ApplicationName: WideString;
  ApplicationRecord: _UAApplicationRecordDataType;
  ApplicationRecordArray: OleVariant;
  ApplicationUriString: WideString;
  GlobalDiscoveryClient: OpcLabs_EasyOpcUA_TLB._EasyUAGlobalDiscoveryClient;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  I, J: integer;
  LastCounterResetTime: TDateTime;
  MaximumRecordsToReturn: Integer;
  NextRecordId: Integer;
  ProductUriString: WideString;
  ServerCapabilities: array of string;
  StartingRecordId: Integer;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Instantiate the global discovery client object
  GlobalDiscoveryClient := CoEasyUAGlobalDiscoveryClient.Create;

  // Find all (client or server) applications registered in the GDS.
  StartingRecordId := 0;
  MaximumRecordsToReturn := 0;
  ApplicationName := '';
  ApplicationUriString := '';
  ProductUriString := '';
  try
    GlobalDiscoveryClient.QueryApplications(
      GdsEndpointDescriptor,
      StartingRecordId,
      MaximumRecordsToReturn,
      ApplicationName,
      ApplicationUriString,
      UAApplicationTypes_All,
      ProductUriString,
```

```
      ServerCapabilities,
      LastCounterResetTime,
      NextRecordId,
      ApplicationDescriptionArray);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // For each application returned by the query, find its registrations in the GDS.
  for I := VarArrayLowBound(ApplicationDescriptionArray,1) to
VarArrayHighBound(ApplicationDescriptionArray,1) do
  begin
    ApplicationDescription := IUnknown(ApplicationDescriptionArray[I]) as
_UAApplicationDescription;
    WriteLn;
    WriteLn('Application URI string: ', ApplicationDescription.ApplicationUriString);
    try
    TVarData(ApplicationRecordArray).VType := varArray or varVariant;
    TVarData(ApplicationRecordArray).VArray := PVarArray(
      GlobalDiscoveryClient.FindApplications(
        GdsEndpointDescriptor,
        ApplicationDescription.ApplicationUriString));
    except
      on E: EOleException do
      begin
        WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
        Continue;
      end;
    end;
      for J := VarArrayLowBound(ApplicationRecordArray, 1) to
VarArrayHighBound(ApplicationRecordArray, 1) do
      begin
        // Display results
        ApplicationRecord := IUnknown(ApplicationRecordArray[J]) as
_UAApplicationRecordDataType;
        WriteLn('  Application ID: ', ApplicationRecord.ApplicationId.ToString);
      end;
  end;

  // Example output:
  //
  //Application URI string: urn:sampleserver
  //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=09ecaa08-6ec6-462c-a214-1e66a3099107
  //
  //Application URI string: urn:alarmconditionserver
  //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=783e1e9a-8036-43b6-928f-97488c460266
  //
  //Application URI string: urn:PC:MultiTargetUADocExamples:5.54.1026.1:neutral:null
  //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=9e700ea5-55a6-4c3c-ba9f-b91c890dc519
  //
  //Application URI string: urn:PC:UADocExamples:5.56.0.16:neutral:null
```

```
  //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=e182e28c-086b-4fc7-82c7-70ca7cda3033
  //
  //Application URI string: urn:PC:cscript:5.812.10240.16384
  //  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/
;ns=2;g=aec94459-f513-4979-8619-8383555fca61

end;
```

VB.NET

```
' Shows how to find all registrations in the GDS.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
    Friend Class FindApplications
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appuser", "demo")

            ' Instantiate the global discovery client object
            Dim globalDiscoveryClient = New EasyUAGlobalDiscoveryClient()

            ' Find all (client or server) applications registered in the GDS.
            Dim applicationDescriptionArray() As UAApplicationDescription = Nothing
            Try
                Dim lastCounterResetTime As DateTime
                Dim nextRecordId As Long
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor:=gdsEndpointDescriptor,
                    startingRecordId:=0,
                    maximumRecordsToReturn:=0,
                    applicationName:="",
                    applicationUriString:="",
                    applicationTypes:=UAApplicationTypes.All,
                    productUriString:="",
                    serverCapabilities:=New String() {},
                    lastCounterResetTime:=lastCounterResetTime,
                    nextRecordId:=nextRecordId,
                    applications:=applicationDescriptionArray)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' For each application returned by the query, find its registrations in the
GDS.
```

```
            For Each applicationDescription As UAApplicationDescription In
applicationDescriptionArray
                Console.WriteLine()
                Console.WriteLine("Application URI string: {0}",
applicationDescription.ApplicationUriString)

                Dim applicationRecordArray() As UAApplicationRecordDataType
                Try
                    applicationRecordArray = globalDiscoveryClient.FindApplications(
                        gdsEndpointDescriptor,
                        applicationDescription.ApplicationUriString)
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Continue For
                End Try

                ' Display results
                For Each applicationRecord As UAApplicationRecordDataType In
applicationRecordArray
                    Console.WriteLine("  Application ID: {0}",
applicationRecord.ApplicationId)
                Next applicationRecord
            Next applicationDescription
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Shows how to find all registrations in the GDS.

Option Explicit

Const UAApplicationTypes_All = 7

' Define which GDS we will work with.
Dim GdsEndpointDescriptor: Set GdsEndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
GdsEndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName = "appadmin"
GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password = "demo"

' Instantiate the global discovery client object
Dim GlobalDiscoveryClient: Set GlobalDiscoveryClient =
CreateObject("OpcLabs.EasyOpc.UA.Gds.EasyUAGlobalDiscoveryClient")

' Find all (client or server) applications registered in the GDS.
Dim startingRecordId: startingRecordId = 0
Dim maximumRecordsToReturn: maximumRecordsToReturn = 0
Dim applicationName: applicationName = ""
Dim applicationUriString: applicationUriString = ""
Dim productUriString: productUriString = ""
Dim serverCapabilities: serverCapabilities = Array()
Dim lastCounterResetTime
Dim nextRecordId
```

```
Dim applicationDescriptionArray
On Error Resume Next
GlobalDiscoveryClient.QueryApplications _
    GdsEndpointDescriptor, _
    startingRecordId, _
    maximumRecordsToReturn, _
    applicationName, _
    applicationUriString, _
    UAApplicationTypes_All, _
    productUriString, _
    serverCapabilities, _
    lastCounterResetTime, _
    nextRecordId, _
    applicationDescriptionArray
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' For each application returned by the query, find its registrations in the GDS.
Dim ApplicationDescription
For Each ApplicationDescription In applicationDescriptionArray
    WScript.Echo
    WScript.Echo "Application URI string: " &
ApplicationDescription.ApplicationUriString

    On Error Resume Next
    Dim applicationRecordArray: applicationRecordArray =
GlobalDiscoveryClient.FindApplications( _
        gdsEndpointDescriptor, _
        ApplicationDescription.ApplicationUriString)
    If Err.Number <> 0 Then
        WScript.Echo "  *** Failure: " & Err.Source & ": " & Err.Description
    Else
        Dim ApplicationRecord
        For Each ApplicationRecord In applicationRecordArray
            ' Display results
            WScript.Echo "  Application ID: " & ApplicationRecord.ApplicationId
        Next
    End If
    On Error Goto 0
Next


' Example output:
'
'Application URI string: urn:sampleserver
'  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=09ecaa08-
6ec6-462c-a214-1e66a3099107
'
'Application URI string: urn:alarmconditionserver
'  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=783e1e9a-
8036-43b6-928f-97488c460266
'
'Application URI string: urn:PC:MultiTargetUADocExamples:5.54.1026.1:neutral:null
'  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=9e700ea5-
```

```
55a6-4c3c-ba9f-b91c890dc519
'
'Application URI string: urn:PC:UADocExamples:5.56.0.16:neutral:null
'  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=e182e28c-
086b-4fc7-82c7-70ca7cda3033
'
'Application URI string: urn:PC:cscript:5.812.10240.16384
'  Application ID: nsu=http://opcfoundation.org/UA/GDS/applications/ ;ns=2;g=aec94459-
f513-4979-8619-8383555fca61
```

13.2.9.3 Examples - OPC UA GDS - Query for applications

C#

```
// Shows how to find client or server applications that meet the specified filters,
using the global discovery client.

using System;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
{
    class QueryApplications
    {
        public static void Main1()
        {
            // Instantiate the global discovery client object
            var globalDiscoveryClient = new EasyUAGlobalDiscoveryClient();

            // Find all (client or server) applications registered in the GDS.
            UAApplicationDescription[] applicationDescriptionArray;
            try
            {
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor:"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer",
                    startingRecordId:0,
                    maximumRecordsToReturn:0,
                    applicationName:"",
                    applicationUriString:"",
                    applicationTypes:UAApplicationTypes.All,
                    productUriString:"",
                    serverCapabilities:new string[0],
                    lastCounterResetTime:out _,
                    nextRecordId: out _,
                    applications: out applicationDescriptionArray);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
```

```
            }

            // Display results
            foreach (UAApplicationDescription applicationDescription in
applicationDescriptionArray)
            {
                Console.WriteLine();
                Console.WriteLine("Application name: {0}",
applicationDescription.ApplicationName);
                Console.WriteLine("Application type: {0}",
applicationDescription.ApplicationType);
                Console.WriteLine("Application URI string: {0}",
applicationDescription.ApplicationUriString);
                Console.WriteLine("Discovery URI strings: {0}",
applicationDescription.DiscoveryUriStrings);
            }
        }
    }
}
```

Object Pascal

```
// Shows how to find client or server applications that meet the specified filters,
using the global discovery client.

class procedure QueryApplications.Main;
var
  ApplicationDescription: _UAApplicationDescription;
  ApplicationDescriptionArray: OleVariant;
  ApplicationName: WideString;
  ApplicationUriString: WideString;
  GlobalDiscoveryClient: OpcLabs_EasyOpcUA_TLB._EasyUAGlobalDiscoveryClient;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  I: integer;
  LastCounterResetTime: TDateTime;
  MaximumRecordsToReturn: Integer;
  NextRecordId: Integer;
  ProductUriString: WideString;
  ServerCapabilities: array of string;
  StartingRecordId: Integer;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';

  // Instantiate the global discovery client object
  GlobalDiscoveryClient := CoEasyUAGlobalDiscoveryClient.Create;

  // Find all (client or server) applications registered in the GDS.
  StartingRecordId := 0;
  MaximumRecordsToReturn := 0;
  ApplicationName := '';
  ApplicationUriString := '';
  ProductUriString := '';
  try
    GlobalDiscoveryClient.QueryApplications(
```

```
      GdsEndpointDescriptor,
      StartingRecordId,
      MaximumRecordsToReturn,
      ApplicationName,
      ApplicationUriString,
      UAApplicationTypes_All,
      ProductUriString,
      ServerCapabilities,
      LastCounterResetTime,
      NextRecordId,
      ApplicationDescriptionArray);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  for I := VarArrayLowBound(ApplicationDescriptionArray,1) to
VarArrayHighBound(ApplicationDescriptionArray,1) do
  begin
    ApplicationDescription := IUnknown(ApplicationDescriptionArray[I]) as
_UAApplicationDescription;
    WriteLn;
    WriteLn('Application name: ', ApplicationDescription.ApplicationName);
    WriteLn('Application type: ', ApplicationDescription.ApplicationType);
    WriteLn('Application URI string: ', ApplicationDescription.ApplicationUriString);
    WriteLn('Discovery URI strings: ',
ApplicationDescription.DiscoveryUriStrings.ToString);
  end;

end;
```

VB.NET

```
' Shows how to find client or server applications that meet the specified filters,
using the global discovery client.

Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
    Friend Class QueryApplications
        Public Shared Sub Main1()

            ' Instantiate the global discovery client object
            Dim globalDiscoveryClient = New EasyUAGlobalDiscoveryClient()

            ' Find all (client or server) applications registered in the GDS.
            Dim applicationDescriptionArray() As UAApplicationDescription = Nothing
            Try
                Dim lastCounterResetTime As DateTime
                Dim nextRecordId As Long
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor:="opc.tcp://opcua.demo-
```

```
this.com:58810/GlobalDiscoveryServer",
                    startingRecordId:=0,
                    maximumRecordsToReturn:=0,
                    applicationName:="",
                    applicationUriString:="",
                    applicationTypes:=UAApplicationTypes.All,
                    productUriString:="",
                    serverCapabilities:=New String() {},
                    lastCounterResetTime:=lastCounterResetTime,
                    nextRecordId:=nextRecordId,
                    applications:=applicationDescriptionArray)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each applicationDescription As UAApplicationDescription In
applicationDescriptionArray
                Console.WriteLine()
                Console.WriteLine("Application name: {0}",
applicationDescription.ApplicationName)
                Console.WriteLine("Application type: {0}",
applicationDescription.ApplicationType)
                Console.WriteLine("Application URI string: {0}",
applicationDescription.ApplicationUriString)
                Console.WriteLine("Discovery URI strings: {0}",
applicationDescription.DiscoveryUriStrings)
            Next applicationDescription
        End Sub
    End Class
End Namespace
```

13.2.9.4 Examples - OPC UA GDS - Query for servers

C#

```
// Shows how to find server applications that meet the specified filters, using the
global discovery client.

using System;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
{
    class QueryServers
    {
        public static void Main1()
        {
            // Instantiate the global discovery client object
            var globalDiscoveryClient = new EasyUAGlobalDiscoveryClient();
```

```
            // Find all servers registered in the GDS.
            UAServerOnNetwork[] serverOnNetworkArray;
            try
            {
                globalDiscoveryClient.QueryServers(
                    gdsEndpointDescriptor:"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer",
                    startingRecordId:0,
                    maximumRecordsToReturn:0,
                    applicationName:"",
                    applicationUriString:"",
                    productUriString:"",
                    serverCapabilities:new string[0],
                    lastCounterResetTime:out _,
                    serverOnNetworkArray:out serverOnNetworkArray);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UAServerOnNetwork serverOnNetwork in serverOnNetworkArray)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", serverOnNetwork.ServerName);
                Console.WriteLine("Discovery URL string: {0}",
serverOnNetwork.DiscoveryUrlString);
                Console.WriteLine("Server capabilities: {0}",
serverOnNetwork.ServerCapabilities);
            }
        }
    }
}
```

Object Pascal

```
// Shows how to find server applications that meet the specified filters, using the
global discovery client.

class procedure QueryServers.Main;
var
  ApplicationName: WideString;
  ApplicationUriString: WideString;
  GlobalDiscoveryClient: OpcLabs_EasyOpcUA_TLB._EasyUAGlobalDiscoveryClient;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  I: integer;
  LastCounterResetTime: TDateTime;
  MaximumRecordsToReturn: Integer;
  ProductUriString: WideString;
  ServerCapabilities: array of string;
  ServerOnNetwork: _UAServerOnNetwork;
  ServerOnNetworkArray: OleVariant;
  StartingRecordId: Integer;
begin
```

```
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';

  // Instantiate the global discovery client object
  GlobalDiscoveryClient := CoEasyUAGlobalDiscoveryClient.Create;

  // Find all servers registered in the GDS.
  StartingRecordId := 0;
  MaximumRecordsToReturn := 0;
  ApplicationName := '';
  ApplicationUriString := '';
  ProductUriString := '';
  try
    GlobalDiscoveryClient.QueryServers(
      GdsEndpointDescriptor,
      StartingRecordId,
      MaximumRecordsToReturn,
      ApplicationName,
      ApplicationUriString,
      ProductUriString,
      ServerCapabilities,
      LastCounterResetTime,
      ServerOnNetworkArray);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Display results
  for I := VarArrayLowBound(ServerOnNetworkArray,1) to
VarArrayHighBound(ServerOnNetworkArray,1) do
  begin
    ServerOnNetwork := IUnknown(ServerOnNetworkArray[I]) as _UAServerOnNetwork;
    WriteLn;
    WriteLn('Server name: ', ServerOnNetwork.ServerName);
    WriteLn('Discovery URL string: ', ServerOnNetwork.DiscoveryUrlString);
    WriteLn('Server capabilities: ', ServerOnNetwork.ServerCapabilities.ToString);
  end;

end;
```

VB.NET

```
' Shows how to find server applications that meet the specified filters, using the
global discovery client.

Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
    Friend Class QueryServers
        Public Shared Sub Main1()
```

```
            ' Instantiate the global discovery client object
            Dim globalDiscoveryClient = New EasyUAGlobalDiscoveryClient()

            ' Find all servers registered in the GDS.
            Dim serverOnNetworkArray() As UAServerOnNetwork = Nothing
            Try
                Dim lastCounterResetTime As DateTime
                globalDiscoveryClient.QueryServers(
                    gdsEndpointDescriptor:="opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer",
                    startingRecordId:=0,
                    maximumRecordsToReturn:=0,
                    applicationName:="",
                    applicationUriString:="",
                    productUriString:="",
                    serverCapabilities:=New String() {},
                    lastCounterResetTime:=lastCounterResetTime,
                    serverOnNetworkArray:=serverOnNetworkArray)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each serverOnNetwork As UAServerOnNetwork In serverOnNetworkArray
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", serverOnNetwork.ServerName)
                Console.WriteLine("Discovery URL string: {0}",
serverOnNetwork.DiscoveryUrlString)
                Console.WriteLine("Server capabilities: {0}",
serverOnNetwork.ServerCapabilities)
            Next serverOnNetwork
        End Sub
    End Class
End Namespace
```

13.2.9.5 Examples - OPC UA GDS - Unregister all clients

C#

```
// Shows how to unregister all clients from a GDS.

using System;
using System.Collections.Generic;
using System.Linq;
using OpcLabs.BaseLib.Collections.Generic.Extensions;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.Extensions;
using OpcLabs.EasyOpc.UA.Gds;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
{
    class UnregisterApplication
    {
        public static void Main1()
        {
            // Define which GDS we will work with.
            UAEndpointDescriptor gdsEndpointDescriptor =
                ((UAEndpointDescriptor)"opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
                .WithUserNameIdentity("appadmin", "demo");

            // Instantiate the global discovery client object
            var globalDiscoveryClient = new EasyUAGlobalDiscoveryClient();

            // Find application IDs of all client applications registered in the GDS.
            var clientApplicationIds = new HashSet<UANodeId>();
            try
            {
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor: gdsEndpointDescriptor,
                    startingRecordId: 0,
                    maximumRecordsToReturn: 0,
                    applicationName: "",
                    applicationUriString: "",
                    applicationTypes: UAApplicationTypes.Client,
                    productUriString: "",
                    serverCapabilities: new string[0],
                    lastCounterResetTime: out _,
                    nextRecordId: out _,
                    applications: out UAApplicationDescription[]
applicationDescriptionArray);

                foreach (UAApplicationDescription applicationDescription in
applicationDescriptionArray)
                {
                    var applicationRecordArray =
globalDiscoveryClient.FindApplications(
                        gdsEndpointDescriptor,
                        applicationDescription.ApplicationUriString);

clientApplicationIds.AddRange(applicationRecordArray.Select(description =>
description.ApplicationId));
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Unregister all client applications found.
            foreach (UANodeId applicationId in clientApplicationIds)
            {
                Console.WriteLine();
                Console.WriteLine("Application ID: {0}", applicationId);
```

```
                try
                {
                    globalDiscoveryClient.UnregisterApplication(gdsEndpointDescriptor,
applicationId);
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                    continue;
                }
                Console.WriteLine("Unregistered.");
            }
        }
    }
}
```

Object Pascal

```
// Shows how to unregister all clients from a GDS.

class procedure UnregisterApplication.Main;
var
  ApplicationDescription: _UAApplicationDescription;
  ApplicationDescriptionArray: OleVariant;
  ApplicationId: _UANodeId;
  ApplicationName: WideString;
  ApplicationRecord: _UAApplicationRecordDataType;
  ApplicationRecordArray: OleVariant;
  ApplicationUriString: WideString;
  ClientApplicationIds: TList<_UANodeID>;
  GlobalDiscoveryClient: OpcLabs_EasyOpcUA_TLB._EasyUAGlobalDiscoveryClient;
  GdsEndpointDescriptor: _UAEndpointDescriptor;
  I, J: integer;
  LastCounterResetTime: TDateTime;
  MaximumRecordsToReturn: Integer;
  NextRecordId: Integer;
  ProductUriString: WideString;
  ServerCapabilities: array of string;
  StartingRecordId: Integer;
begin
  // Define which GDS we will work with.
  GdsEndpointDescriptor := CoUAEndpointDescriptor.Create;
  GdsEndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.UserName := 'appadmin';
  GdsEndpointDescriptor.UserIdentity.UserNameTokenInfo.Password := 'demo';

  // Instantiate the global discovery client object
  GlobalDiscoveryClient := CoEasyUAGlobalDiscoveryClient.Create;

  // Find application IDs of all client applications registered in the GDS.
  ClientApplicationIds := TList<_UANodeID>.Create();
  StartingRecordId := 0;
  MaximumRecordsToReturn := 0;
  ApplicationName := '';
```

```
  ApplicationUriString := '';
  ProductUriString := '';
  try
    GlobalDiscoveryClient.QueryApplications(
      GdsEndpointDescriptor,
      StartingRecordId,
      MaximumRecordsToReturn,
      ApplicationName,
      ApplicationUriString,
      UAApplicationTypes_Client,
      ProductUriString,
      ServerCapabilities,
      LastCounterResetTime,
      NextRecordId,
      ApplicationDescriptionArray);

    for I := VarArrayLowBound(ApplicationDescriptionArray,1) to
VarArrayHighBound(ApplicationDescriptionArray,1) do
    begin
      ApplicationDescription := IUnknown(ApplicationDescriptionArray[I]) as
_UAApplicationDescription;
      TVarData(ApplicationRecordArray).VType := varArray or varVariant;
      TVarData(ApplicationRecordArray).VArray :=
PVarArray(GlobalDiscoveryClient.FindApplications(
        GdsEndpointDescriptor, ApplicationDescription.ApplicationUriString));
      for J := VarArrayLowBound(ApplicationRecordArray,1) to
VarArrayHighBound(ApplicationRecordArray,1) do
      begin
        ApplicationRecord := IUnknown(ApplicationRecordArray[J]) as
_UAApplicationRecordDataType;
        ClientApplicationIds.Add(ApplicationRecord.ApplicationId);
      end;
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // Unregister all client applications found.
  for ApplicationId in ClientApplicationIds do
  begin
    WriteLn;
    WriteLn('Application ID: ', ApplicationId.ToString);
    try
      GlobalDiscoveryClient.UnregisterApplication(
        GdsEndpointDescriptor, ApplicationId);
    except
      on E: EOleException do
      begin
        WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
        Continue;
      end;
    end;
    WriteLn('Unregistered.');
  end;
```

```
  FreeAndNil(ClientApplicationIds);
end;
```

VB.NET

```
' Shows how to unregister all clients from a GDS.

Imports System.Linq
Imports OpcLabs.BaseLib.Collections.Generic.Extensions
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.Extensions
Imports OpcLabs.EasyOpc.UA.Gds
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Gds._EasyUAGlobalDiscoveryClient
    Friend Class UnregisterApplication
        Public Shared Sub Main1()

            ' Define which GDS we will work with.
            Dim gdsEndpointDescriptor As UAEndpointDescriptor =
                New UAEndpointDescriptor("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer") _
                .WithUserNameIdentity("appadmin", "demo")

            ' Instantiate the global discovery client object
            Dim globalDiscoveryClient = New EasyUAGlobalDiscoveryClient()

            ' Find application IDs of all client applications registered in the GDS.
            Dim clientApplicationIds = New HashSet(Of UANodeId)
            Try
                Dim lastCounterResetTime As DateTime
                Dim nextRecordId As Long
                Dim applicationDescriptionArray() As UAApplicationDescription = Nothing
                globalDiscoveryClient.QueryApplications(
                    gdsEndpointDescriptor:=gdsEndpointDescriptor,
                    startingRecordId:=0,
                    maximumRecordsToReturn:=0,
                    applicationName:="",
                    applicationUriString:="",
                    applicationTypes:=UAApplicationTypes.Client,
                    productUriString:="",
                    serverCapabilities:=New String() {},
                    lastCounterResetTime:=lastCounterResetTime,
                    nextRecordId:=nextRecordId,
                    applications:=applicationDescriptionArray)

                For Each applicationDescription As UAApplicationDescription In
applicationDescriptionArray
                    Dim applicationRecordArray =
globalDiscoveryClient.FindApplications(
                        gdsEndpointDescriptor,
                        applicationDescription.ApplicationUriString)

clientApplicationIds.AddRange(applicationRecordArray.Select(Function(description)
description.ApplicationId))
```

```
                Next applicationDescription
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Unregister all client applications found.
            For Each applicationId As UANodeId In clientApplicationIds
                Console.WriteLine()
                Console.WriteLine("Application ID: {0}", applicationId)

                Try
                    globalDiscoveryClient.UnregisterApplication(gdsEndpointDescriptor,
applicationId)
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Continue For
                End Try
                Console.WriteLine("Unregistered.")
            Next applicationId
        End Sub
    End Class
End Namespace
```

13.2.10 Examples - OPC UA Interaction

13.2.10.1 Examples - OPC UA Interaction - Accept HTTPS certificate

The example below triggers the component to ask the user whether he/she wants to accept the server's HTTPS certificate.

C#

```
// This example shows how in a console application, the user is asked to accept a server HTTPS certificate.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    partial class AcceptCertificate
    {
        public static void Https()
        {
            // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.

EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";

            // Instantiate the client object.
            var client = new EasyUAClient();

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
```

```
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// This example shows how in a console application, the user is asked to accept a server HTTPS certificate.

class procedure AcceptCertificate.Https;
var
  AttributeData: _UAAttributeData;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  EndpointDescriptor: string;
begin
  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.

ClientConfiguration.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();


  // Define which server we will work with.
  EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;
  try
    // Obtain attribute data.
    // The component automatically triggers the necessary user interaction during the first operation.
    AttributeData := Client.Read(EndpointDescriptor, 'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results.
  WriteLn('Value: ', AttributeData.Value);
  WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
  WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
  WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);

  FreeAndNil(ClientConfiguration);
end;
```

VB.NET

```
' This example shows how in a console application, the user is asked to accept a server HTTPS certificate.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Interaction
    Partial Friend Class AcceptCertificate
        Public Shared Sub Https()
```

```
            ' Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.

EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

The program output may look like this:

13.2.10.2 Examples - OPC UA Interaction - Accept instance certificate

The example below triggers the component to ask the user whether he/she wants to accept the server's instance certificate.

C#

```
// This example shows how in a console application, the user is asked to accept a server instance certificate.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Engine;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    partial class AcceptCertificate
    {
        public static void Instance()
        {
            // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.

EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = UAMessageSecurityModes.Secure;

            // Instantiate the client object.
```

```
            var client = new EasyUAClient();

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// This example shows how in a console application, the user is asked to accept a server instance certificate.

class procedure AcceptCertificate.Instance;
var
  Arguments: OleVariant;
  AttributeData: _UAAttributeData;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  EndpointSelectionPolicy: _UAEndpointSelectionPolicy;
  ReadArguments: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.

ClientConfiguration.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();


  // Define which server we will work with.
  ReadArguments := CoUAReadArguments.Create;
  ReadArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  // Require secure connection, in order to enforce the certificate check.
  EndpointSelectionPolicy := CoUAEndpointSelectionPolicy.Create;
  EndpointSelectionPolicy.AllowedMessageSecurityModes := UAMessageSecurityModes_Secure;
  ReadArguments.EndpointDescriptor.EndpointSelectionPolicy := EndpointSelectionPolicy;
  ReadArguments.NodeDescriptor.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=10853';

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := ReadArguments;

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;

  // Obtain attribute data.
  // The component automatically triggers the necessary user interaction during the first operation.
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

  Result := IInterface(Results[0]) as _UAAttributeDataResult;
  if Result.Succeeded then
  begin
    AttributeData := Result.AttributeData;
    // Display results.
    WriteLn('Value: ', AttributeData.Value);
    WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
```

```
    WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
    WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);
  end
  else
    WriteLn('*** Failure: ', Result.ErrorMessageBrief);

  FreeAndNil(ClientConfiguration);
end;
```

VB.NET

```
' This example shows how in a console application, the user is asked to accept a server instance certificate.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Engine
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Interaction
    Partial Friend Class AcceptCertificate
        Public Shared Sub Instance()

            ' Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.

EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = UAMessageSecurityModes.Secure

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

The program output may look like this:

13.2.10.3 Examples - OPC UA Interaction - Allow endpoint domain

The example below triggers the component to ask the user whether he/she wants to accept an instance certificate whose domain does not match the URL used to connect to the server.

C#

```
// This example shows how in a console application, the user is asked to allow a server
instance certificate with
// mismatched domain name.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    class AllowEndpointDomain
    {
        public static void Main1()
        {
            // Define which server we will work with.
            // Note that extra '.' at the end of the domain name. For the purpose of
this example, it allows us to address
            // the same domain, but cause a mismatch with what the names that are
listed in the server instance certificate.
            UAEndpointDescriptor endpointDescriptor = "opc.tcp://opcua.demo-
this.com.:51210/UA/SampleServer";

            // Instantiate the client object.
            var client = new EasyUAClient()
            {
                // Enforce the endpoint domain check.
                Isolated = true,
                IsolatedParameters = {SessionParameters = {CheckEndpointDomain = true}}
            };

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction
during the first operation.
                attributeData = client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// This example shows how in a console application, the user is asked to allow a server
instance certificate with
// mismatched domain name.

class procedure AllowEndpointDomain.Main;
var
  AttributeData: _UAAttributeData;
  Client: _EasyUAClient;
  EndpointDescriptor: string;
begin
  // Define which server we will work with.
  // Note that extra '.' at the end of the domain name. For the purpose of this
example, it allows us to address
  // the same domain, but cause a mismatch with what the names that are listed in the
server instance certificate.
  EndpointDescriptor := 'opc.tcp://opcua.demo-this.com.:51210/UA/SampleServer';

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;
  // Enforce the endpoint domain check.
  Client.Isolated := true;
  Client.IsolatedParameters.SessionParameters.CheckEndpointDomain := true;

  try
    // Obtain attribute data.
    // The component automatically triggers the necessary user interaction during the
first operation.
    AttributeData := Client.Read(EndpointDescriptor,
'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results.
  WriteLn('Value: ', AttributeData.Value);
  WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
  WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
  WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);
end;
```

VB.NET

```
' This example shows how in a console application, the user is asked to allow a server
instance certificate with
' mismatched domain name.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.Engine
```

```
Namespace UADocExamples.Interaction
    Friend Class AllowEndpointDomain
        Public Shared Sub Main1()

            ' Define which server we will work with.
            ' Note that extra '.' at the end of the domain name. For the purpose of
this example, it allows us to address
            ' the same domain, but cause a mismatch with what the names that are listed
in the server instance certificate.
            Dim endpointDescriptor As UAEndpointDescriptor = "opc.tcp://opcua.demo-
this.com.:51210/UA/SampleServer"

            ' Instantiate the client object.
            Dim client = New EasyUAClient() With _
            { _
                .Isolated = True, _
                .IsolatedParameters = New EasyUAAdaptableParameters() With _
                { _
                    .SessionParameters = New UASmartSessionParameters() With _
                    { _
                        .CheckEndpointDomain = True _
                    } _
                } _
            }

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction
during the first operation.
                attributeData = client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how in a console application, the user is asked to allow a
server instance certificate with
Rem mismatched domain name.

Option Explicit

' Define which server we will work with.
' Note that extra '.' at the end of the domain name. For the purpose of this example,
```

```
it allows us to address
' the same domain, but cause a mismatch with what the names that are listed in the
server instance certificate.
Dim endpointDescriptor: endpointDescriptor = "opc.tcp://opcua.demo-
this.com.:51210/UA/SampleServer"

' Instantiate the client object.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
' Enforce the endpoint domain check.
Client.Isolated = True
Client.IsolatedParameters.SessionParameters.CheckEndpointDomain = True

' Obtain attribute data.
' The component automatically triggers the necessary user interaction during the first
operation.
On Error Resume Next
Dim AttributeData: Set AttributeData = Client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Value: " & AttributeData.Value
WScript.Echo "ServerTimestamp: " & AttributeData.ServerTimestamp
WScript.Echo "SourceTimestamp: " & AttributeData.SourceTimestamp
WScript.Echo "StatusCode: " & AttributeData.StatusCode

' Example output:
'
'OPC-UA Endpoint Domain Mismatch
'The effective host name in endpoint URL returned by the server does not match any of
the domain names in the server certificate.
'Endpoint URL as returned by the server: opc.tcp://opcua.demo-
this.com.:51210/UA/SampleServer
'The server certificate is for following domain names or IP addresses: opcua.demo-
this.com
'This may be an indication of a spoofing attempt. Do you want to allow the endpoint
anyway? [Y/n]: Y
'Value: -1.285897E+14
'ServerTimestamp: 11/28/2019 1:34:23 PM
'SourceTimestamp: 11/28/2019 1:34:23 PM
'StatusCode: Good
```

The program output may look like this:

13.2.10.4 Examples - OPC UA Interaction - Turn off output colorization

The example below configures the component to use uncolorized console output, and then triggers some user interaction.

C#

```
// Shows how to configure the OPC UA Console Interaction plug-in by turning off the output colorization.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.Console.Interaction;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Engine;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples.Interaction
{
    partial class ConsoleInteraction
    {
        public static void ColorizeOutput()
        {
            // Configure the shared plug-in.

            // Find the parameters object of the plug-in.
            ConsoleInteractionParameters consoleInteractionPluginParameters =
                EasyUAClient.SharedParameters.PluginConfigurations.Find<ConsoleInteractionParameters>();
            Debug.Assert(consoleInteractionPluginParameters != null);
            // Change the parameter.
            consoleInteractionPluginParameters.ColorizeOutput = false;


            // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

            // Define which server we will work with.
            UAEndpointDescriptor endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = new UAEndpointSelectionPolicy(UAMessageSecurityModes.Secure);

            // Instantiate the client object.
            var client = new EasyUAClient();

            UAAttributeData attributeData;
            try
            {
                // Obtain attribute data.
                // The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
                return;
            }

            // Display results.
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);
        }
    }
}
```

Object Pascal

```
// Shows how to configure the OPC UA Console Interaction plug-in by turning off the output colorization.

class procedure ConsoleInteraction.ColorizeOutput;
var
  Arguments: OleVariant;
  AttributeData: _UAAttributeData;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  ConsoleInteractionParameters: _ConsoleInteractionParameters;
  EndpointSelectionPolicy: _UAEndpointSelectionPolicy;
  ReadArguments: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  // Configure the shared plug-in.

  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Find the parameters object of the plug-in.
  ConsoleInteractionParameters :=
```

```
IUnknown(ClientConfiguration.SharedParameters.PluginConfigurations.Find('OpcLabs.BaseLib.Console.Interaction.ConsoleInteractionParameters'))
as _ConsoleInteractionParameters;
  // Change the parameter.
  ConsoleInteractionParameters.ColorizeOutput := False;

  // Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
  ClientConfiguration.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear();

  // Define which server we will work with.
  ReadArguments := CoUAReadArguments.Create;
  ReadArguments.EndpointDescriptor.UrlString := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
  // Require secure connection, in order to enforce the certificate check.
  EndpointSelectionPolicy := CoUAEndpointSelectionPolicy.Create;
  EndpointSelectionPolicy.AllowedMessageSecurityModes := UAMessageSecurityModes_Secure;
  ReadArguments.EndpointDescriptor.EndpointSelectionPolicy := EndpointSelectionPolicy;
  ReadArguments.NodeDescriptor.NodeId.ExpandedText := 'nsu=http://test.org/UA/Data/;i=10853';

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := ReadArguments;

  // Instantiate the client object.
  Client := CoEasyUAClient.Create;

  // Obtain attribute data.
  // The component automatically triggers the necessary user interaction during the first operation.
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

  Result := IInterface(Results[0]) as _UAAttributeDataResult;
  if Result.Succeeded then
  begin
    AttributeData := Result.AttributeData;
    // Display results.
    WriteLn('Value: ', AttributeData.Value);
    WriteLn('ServerTimestamp: ', DateTimeToStr(AttributeData.ServerTimestamp));
    WriteLn('SourceTimestamp: ', DateTimeToStr(AttributeData.SourceTimestamp));
    WriteLn('StatusCode: ', AttributeData.StatusCode.ToString);
  end
  else
    WriteLn('*** Failure: ', Result.ErrorMessageBrief);

  FreeAndNil(ClientConfiguration);
end;
```

VB.NET

```
' Shows how to configure the OPC UA Console Interaction plug-in by turning off the output colorization.

Imports OpcLabs.BaseLib.Console.Interaction
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Engine
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples.Interaction
    Friend Class ConsoleInteraction
        Public Shared Sub ColorizeOutput()

            ' Configure the shared plug-in.

            ' Find the parameters object of the plug-in.
            Dim consoleInteractionPluginParameters =
                EasyUAClient.SharedParameters.PluginConfigurations.Find(Of ConsoleInteractionParameters)()
            Debug.Assert(consoleInteractionPluginParameters IsNot Nothing)
            ' Change the parameter.
            consoleInteractionPluginParameters.ColorizeOutput = False

            ' Do not implicitly trust any endpoint URLs. We want the user be asked explicitly.
            EasyUAClient.SharedParameters.EngineParameters.CertificateAcceptancePolicy.TrustedEndpointUrlStrings.Clear()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' Require secure connection, in order to enforce the certificate check.
            endpointDescriptor.EndpointSelectionPolicy = New UAEndpointSelectionPolicy(UAMessageSecurityModes.Secure)

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            Dim attributeData As UAAttributeData
            Try
                ' Obtain attribute data.
                ' The component automatically triggers the necessary user interaction during the first operation.
                attributeData = client.Read(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results.
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
```

```
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)
        End Sub
    End Class
End Namespace
```

The program output may look like this:

13.2.11 Examples - OPC UA PubSub

13.2.11.1 Examples - OPC UA PubSub - Callback using a lambda

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and process them using
// a callback method.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Callback()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor =
"opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            Console.WriteLine("Subscribing...");
```

```
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, (sender, args) =>
            {
                // Display the dataset.
                if (args.Succeeded)
                {
                    // An event with null DataSetData just indicates a successful
connection.
                    if (!(args.DataSetData is null))
                    {
                        Console.WriteLine();
                        Console.WriteLine($"Dataset data: {args.DataSetData}");
                        foreach (KeyValuePair<string, UADataSetFieldData> pair in
args.DataSetData.FieldDataDictionary)
                            Console.WriteLine(pair);
                    }
                }
                else
                {
                    Console.WriteLine();
                    Console.WriteLine($"*** Failure: {args.ErrorMessageBrief}");
                }
            });

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-
01-01T00:00:00.000; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-
a715-0979884b55ae, fields: 100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
```

```
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and process them using
' a callback method.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
the interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub Callback()

            ' Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor =
"opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor,
                Sub(sender, eventArgs)
                    '  Display the dataset.
```

```
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful
connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}",
eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String,
UADataSetFieldData) In eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                    End If
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    ''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
    '[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
    '
    'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
    '[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
```

```
    '[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '...

End Namespace
```

F#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and process them using
// a callback method.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

module _EasyUASubscriber.SubscribeDataSet

open OpcLabs.EasyOpc.UA.PubSub
open System
open System.Threading

let Callback =

    // Define the PubSub connection we will work with.
    let pubSubConnectionDescriptor = new UAPubSubConnectionDescriptor()
    pubSubConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString <-
"opc.udp://239.0.0.1"
    // In some cases you may have to set the interface (network adapter) name that
needs to be used, similarly to
    // the statement below. Your actual interface name may differ, of course.
    //pubSubConnectionDescriptor.ResourceAddress.InterfaceName <- "Ethernet"

    // Instantiate the subscriber object.
    let subscriber = new EasyUASubscriber()

    Console.WriteLine("Subscribing...");
    // The callback is a delegate that displays the value
    let handle =
        subscriber.SubscribeDataSet(
            pubSubConnectionDescriptor,
            new EasyUADataSetMessageEventHandler(
                fun sender eventArgs ->
                    // Display the dataset.
                    if eventArgs.Succeeded then
                        // An event with null DataSetData just indicates a successful
connection.
                        if eventArgs.DataSetData <> null then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}",
eventArgs.DataSetData)
                            for pair in eventArgs.DataSetData.FieldDataDictionary do
                                Console.WriteLine(pair)
```

```
                    else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)))

    Console.WriteLine("Processing dataset message events for 20 seconds...")
    Thread.Sleep(20 * 1000)

    Console.WriteLine("Unsubscribing...")
    subscriber.UnsubscribeAllDataSets()

    Console.WriteLine("Waiting for 1 second...")
    // Unsubscribe operation is asynchronous, messages may still come for a short
while.
    Thread.Sleep(1 * 1000)

    Console.WriteLine("Finished.")



    // Example output:
    //
    //Subscribing...
    //Processing dataset message events for 20 seconds...
    //
    ////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
    //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
    //
    //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-
a715-0979884b55ae, fields: 100
    //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    //...
```

13.2.11.2 Examples - OPC UA PubSub - Ethernet UADP mapping

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection with Ethernet UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.
// The UADemoPublisher must be told to use the Ethernet transport: run it with the -eth
switch on the command line.
//
// The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made
available, together with its
// dependencies) for the Ethernet transport to work, and additional software
installation may be needed as well. Refer to
// the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Ethernet()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            // "opc.eth" is the scheme for OPC UA Ethernet. "FF-FF-FF-FF-FF-FF" is the
Ethernet broadcast address.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.eth://FF-FF-
FF-FF-FF-FF";
            // In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_Ethernet;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
```

```
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_Ethernet(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful
connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-
01-01T00:00:00.000; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-
a715-0979884b55ae, fields: 100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
```

```
Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection with Ethernet UADP mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
the interface name to be used.
' The UADemoPublisher must be told to use the Ethernet transport: run it With the -eth
switch On the command line.
'
' The OpcLabs.Pcap assembly needs to be referenced in your project (Or otherwise made
available, together with its
' dependencies) for the Ethernet transport to work, And additional software
installation may be needed as well. Refer to
' the documentation for more information.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub Ethernet()

            ' Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            ' "opc.eth" Is the scheme for OPC UA Ethernet. "FF-FF-FF-FF-FF-FF" Is the
Ethernet broadcast address.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor =
"opc.eth://FF-FF-FF-FF-FF-FF"
            ' In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"
```

```
            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf
subscriber_DataSetMessage_Ethernet

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_Ethernet(ByVal sender As Object,
ByVal e As EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful
connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    ''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
    '[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
```

```
01T00:00:00.000; Good]
    '
    'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
    '[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection with Ethernet UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.
// The UADemoPublisher must be told to use the Ethernet transport: run it with the -eth
switch on the command line.
//
// The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made
available, together with its
// dependencies) for the Ethernet transport to work, and additional software
installation may be needed as well. Refer to
// the documentation for more information.

type
  TSubscriberEventHandlers73 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Ethernet;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers73;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor :=
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
```

```
  // "opc.eth" is the scheme for OPC UA Ethernet. "FF-FF-FF-FF-FF-FF" is the Ethernet
broadcast address.
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.eth://FF-
FF-FF-FF-FF-FF';
  // In some cases you may have to set the interface (network adapter) name that needs
to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers73.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers73.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator :=
eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
```

```
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

VBScript

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection with Ethernet UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher
tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.
Rem
Rem The OpcLabs.Pcap assembly needs to be made available, together with its
dependencies, for the Ethernet transport to
Rem work, and additional software installation may be needed as well. Refer to the
documentation for more information.

Option Explicit

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
```

```
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
' "opc.eth" is the scheme for OPC UA Ethernet. "FF-FF-FF-FF-FF-FF" is the Ethernet
broadcast address.
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.eth://FF-FF-
FF-FF-FF-FF"
' In some cases you may have to set the interface (network adapter) name that needs to
be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
```

```
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

13.2.11.3 Examples - OPC UA PubSub - Event pull

The example below subscribes to all dataset messages on an OPC-UA PubSub connection with UDP UADP mapping. The connection is specified by its physical parameters, using the scheme name "opc.udp" and the IP address of the multicast group to listen on.

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
// the incoming datasets.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    class PullDataSetMessage
    {
        public static void Main1()
```

```
        {
            // Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor =
"opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Instantiate the subscriber object.
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var subscriber = new EasyUASubscriber {PullDataSetMessageQueueCapacity =
1000};

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            int endTick = Environment.TickCount + 20 * 1000;
            do
            {
                EasyUADataSetMessageEventArgs eventArgs =
subscriber.PullDataSetMessage(2 * 1000);
                if (eventArgs != null)
                {
                    // Display the dataset.
                    if (eventArgs.Succeeded)
                    {
                        // An event with null DataSetData just indicates a successful
connection.
                        if (!(eventArgs.DataSetData is null))
                        {
                            Console.WriteLine();
                            Console.WriteLine($"Dataset data:
{eventArgs.DataSetData}");
                            foreach (KeyValuePair<string, UADataSetFieldData> pair in
eventArgs.DataSetData.FieldDataDictionary)
                                Console.WriteLine(pair);
                        }
                    }
                    else
                    {
                        Console.WriteLine();
                        Console.WriteLine($"*** Failure:
{eventArgs.ErrorMessageBrief}");
                    }
                }
            } while (Environment.TickCount < endTick);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
```

```
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-
01-01T00:00:00.000; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-
a715-0979884b55ae, fields: 100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
' the incoming datasets.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
```

```
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
the interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class PullDataSetMessage
        Public Shared Sub Main1()

            ' Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor =
"opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Instantiate the subscriber object.
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            Dim subscriber = New EasyUASubscriber()
            subscriber.PullDataSetMessageQueueCapacity = 1000

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Dim endTick As Integer = Environment.TickCount + 20 * 1000
            Do
                Dim eventArgs As EasyUADataSetMessageEventArgs =
subscriber.PullDataSetMessage(2 * 1000)
                If Not eventArgs Is Nothing Then
                    ' Display the dataset.
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful
connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}",
eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String,
UADataSetFieldData) In eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                    End If
                End If
            Loop While Environment.TickCount < endTick

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()
```

```
            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    ''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
    '[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
    '
    'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
    '[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
// the incoming datasets.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.


class procedure PullDataSetMessage.Main1;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
```

```
  Count: Cardinal;
  Element: OleVariant;
  EndTick: Cardinal;
  EventArgs: _EasyUADataSetMessageEventArgs;
  PairEnumerator: IEnumVARIANT;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: OpcLabs_EasyOpcUA_TLB._EasyUASubscriber;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor :=
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString :=
'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs
to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  // ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Instantiate the subscriber object.
  Subscriber := CoEasyUASubscriber.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Subscriber.PullDataSetMessageQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message events for 20 seconds...');
  EndTick := Ticks + 20*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Subscriber.PullDataSetMessage(2*1000);
    if EventArgs <> nil then
      begin
        // Display the dataset.
        if EventArgs.Succeeded then
        begin
          // An event with null DataSetData just indicates a successful connection.
          if EventArgs.DatasetData <> nil then
          begin
            WriteLn;
            WriteLn('Dataset data: ', EventArgs.DatasetData.ToString);
            PairEnumerator := EventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
            while (PairEnumerator.Next(1, Element, Count) = S_OK) do
            begin
              WriteLn(Element.ToString);
            end;
          end;
        end
        else
        begin
          WriteLn;
             WriteLn(' *** Failure: ', EventArgs.ErrorMessageBrief);
        end;
      end;
  end;
```

```
  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

VBScript

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
Rem the incoming datasets.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher
tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

Option Explicit

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")

Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
```

```
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString =
"opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to
be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Instantiate the subscriber object.
Dim Subscriber: Set Subscriber =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Subscriber.PullDataSetMessageQueueCapacity = 1000

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
Dim endTime: endTime = Now() + 20*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Subscriber.PullDataSetMessage(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Display the dataset.
        If EventArgs.Succeeded Then
            ' An event with null DataSetData just indicates a successful connection.
            If Not (EventArgs.DataSetData Is Nothing) Then
                WScript.Echo
                WScript.Echo "Dataset data: " & EventArgs.DataSetData
                Dim Pair: For Each Pair in EventArgs.DataSetData.FieldDataDictionary
                    WScript.Echo Pair
                Next
            End If
        Else
            WScript.Echo
            WScript.Echo "*** Failure: " & EventArgs.ErrorMessageBrief
        End If
    End If
Loop While Now() < endTime

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
```

```
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection, and pull events, and display
Rem the incoming datasets.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher
tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

Private Sub PullDataSetMessage_Main1_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments = New
EasyUASubscribeDataSetArguments
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString =
"opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs
to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    'ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Instantiate the subscriber object.
    Dim Subscriber: Set Subscriber = New EasyUASubscriber
    ' In order to use event pull, you must set a non-zero queue capacity upfront.
    Subscriber.PullDataSetMessageQueueCapacity = 1000

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." &
vbCrLf
```

```
    Dim endTime: endTime = Now() + 20 * (1 / 24 / 60 / 60)
    Do
        Dim EventArgs: Set EventArgs = Subscriber.PullDataSetMessage(2 * 1000)
        If Not (EventArgs Is Nothing) Then
            ' Display the dataset.
            If EventArgs.Succeeded Then
                ' An event with null DataSetData just indicates a successful
connection.
                If Not (EventArgs.DataSetData Is Nothing) Then
                    OutputText = OutputText & vbCrLf
                    OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData
& vbCrLf
                    Dim Pair: For Each Pair In
EventArgs.DataSetData.FieldDataDictionary
                        OutputText = OutputText & Pair & vbCrLf
                    Next
                End If
            Else
                OutputText = OutputText & vbCrLf
                OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief
& vbCrLf
            End If
        End If
    Loop While Now() < endTime

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
```

```
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

13.2.11.4 Examples - OPC UA PubSub - Extract a field from the message

C#

```
// This example shows how to subscribe to dataset messages and extract data of a specific field.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void ExtractField()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1);

            // Define the metadata, with the use of collection initializer for its fields. For UADP, the order of
field
            // metadata must correspond to the order of fields in the dataset message. If the field names were
contained
            // in the dataset message (such as in JSON), or if we knew the metadata from some other source, this
step would
            // not be needed.
            // Since the encoding is not RawData, we do not have to specify the type information for the fields.
            var metaData = new UADataSetMetaData
            {
                new UAFieldMetaData("BoolToggle"),
                new UAFieldMetaData("Int32"),
                new UAFieldMetaData("Int32Fast"),
                new UAFieldMetaData("DateTime")
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_ExtractField;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);
```

```
            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_ExtractField(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    // Extract field data, looking up the field by its name.
                    UADataSetFieldData int32FastValueData = e.DataSetData.FieldDataDictionary["Int32Fast"];
                    Console.WriteLine(int32FastValueData);
                }
            }
            else
            {
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
        //6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
        //6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
        //6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
        //6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
        //6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
        //6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
        //6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
        //7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
        //7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
        //7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
        //7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
        //7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
        //7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
        //7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and extract data of a specific field.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub ExtractField()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
```

```
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) Is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1)

            ' Define the metadata, with the use of collection initializer for its fields. For UADP, the order of
field
            ' metadata must correspond to the order of fields in the dataset message. If the field names were
contained
            ' in the dataset message (such as in JSON), Or if we knew the metadata from some other source, this
step would
            ' Not be needed.
            ' Since the encoding Is Not RawData, we do Not have to specify the type information for the fields.
            Dim metaData = New UADataSetMetaData From {
                New UAFieldMetaData("BoolToggle"),
                New UAFieldMetaData("Int32"),
                New UAFieldMetaData("Int32Fast"),
                New UAFieldMetaData("DateTime")
            }

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_ExtractField

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_ExtractField(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    ' Extract field data, looking up the field by its name.
                    Dim int32FastValueData = e.DataSetData.FieldDataDictionary("Int32Fast")
                    Console.WriteLine(int32FastValueData)
                End If
            Else
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
    '6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
    '6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
    '6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
    '6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
    '6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
    '6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
    '6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
    '7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
    '7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
    '7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
    '7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
```

```
    '7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
    '7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
    '7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and extract data of a specific field.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers74 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.ExtractField;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Field1, Field2, Field3, Field4: _UAFieldMetaData;
  MetaData: _UADataSetMetaData;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers74;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
  // If the field names were contained in the dataset message (such as in JSON), or if we knew the metadata from
some other
  // source, this step would not be needed.
  // Since the encoding is not RawData, we do not have to specify the type information for the fields.
  MetaData := CoUADataSetMetaData.Create;
  //
  Field1 := CoUAFieldMetaData.Create;
  Field1.Name := 'BoolToggle';
  MetaData.Add(Field1);
  //
  Field2 := CoUAFieldMetaData.Create;
  Field2.Name := 'Int32';
  MetaData.Add(Field2);
  //
  Field3 := CoUAFieldMetaData.Create;
  Field3.Name := 'Int32Fast';
  MetaData.Add(Field3);
  //
  Field4 := CoUAFieldMetaData.Create;
  Field4.Name := 'DateTime';
  MetaData.Add(Field4);
  //
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData := MetaData;

  // Instantiate the subscriber object and hook events.
```

```
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers74.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers74.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Dnt32FastValueData: _UADataSetFieldData;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      // Extract field data, looking up the field by its name.
      Dnt32FastValueData := eventArgs.DataSetData.FieldDataDictionary.Item['Int32Fast'];
      WriteLn(Dnt32FastValueData.ToString);
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
//6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
//6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
//6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
//6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
//6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
//6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
//6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
//7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
//7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
//7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
//7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
//7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
//7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
//7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and extract data of a specific field.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
```

```
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
' If the field names were contained in the dataset message (such as in JSON), or if we knew the metadata from
some other
' source, this step would not be needed.
' Since the encoding is not RawData, we do not have to specify the type information for the fields.
Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
'
Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field1.Name = "BoolToggle"
MetaData.Add(Field1)
'
Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field2.Name = "Int32"
MetaData.Add(Field2)
'
Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field3.Name = "Int32Fast"
MetaData.Add(Field3)
'
Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field4.Name = "DateTime"
MetaData.Add(Field4)
'
Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
```

```
            ' Extract field data, looking up the field by its name.
            Dim Int32FastValueData: Set Int32FastValueData = e.DataSetData.FieldDataDictionary.Item("Int32Fast")
            WScript.Echo Int32FastValueData
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
'6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
'6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
'6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
'6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
'6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
'6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
'6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
'7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
'7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
'7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
'7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
'7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
'7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
'7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages extract data of a specific field.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber4 As EasyUASubscriber

Private Sub SubscribeDataSet_ExtractField_Command_Click()
    OutputText = ""

    Const UAPublisherIdType_UInt64 = 4

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
    ' If the field names were contained in the dataset message (such as in JSON), or if we knew the metadata from
some other
    ' source, this step would not be needed.
    ' Since the encoding is not RawData, we do not have to specify the type information for the fields.
    Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
    '
```

```
    Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field1.Name = "BoolToggle"
    MetaData.Add (Field1)
    '
    Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field2.Name = "Int32"
    MetaData.Add (Field2)
    '
    Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field3.Name = "Int32Fast"
    MetaData.Add (Field3)
    '
    Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field4.Name = "DateTime"
    MetaData.Add (Field4)
    '
    Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

    ' Instantiate the subscriber object and hook events.
    Set Subscriber4 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber4.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber4.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber4 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber4_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            ' Extract field data, looking up the field by its name.
            Dim Int32FastValueData: Set Int32FastValueData =
EventArgs.DataSetData.FieldDataDictionary.Item("Int32Fast")
            OutputText = OutputText & Int32FastValueData & vbCrLf
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'6502 {System.Int32} @2019-10-06T10:02:01.254,647,600,00; Good
'6538 {System.Int32} @2019-10-06T10:02:01.755,010,700,00; Good
'6615 {System.Int32} @2019-10-06T10:02:02.255,780,200,00; Good
'6687 {System.Int32} @2019-10-06T10:02:02.756,495,900,00; Good
'6769 {System.Int32} @2019-10-06T10:02:03.257,320,200,00; Good
'6804 {System.Int32} @2019-10-06T10:02:03.757,667,300,00; Good
'6877 {System.Int32} @2019-10-06T10:02:04.258,405,000,00; Good
'6990 {System.Int32} @2019-10-06T10:02:04.759,532,900,00; Good
'7063 {System.Int32} @2019-10-06T10:02:05.260,257,200,00; Good
'7163 {System.Int32} @2019-10-06T10:02:05.761,261,800,00; Good
'7255 {System.Int32} @2019-10-06T10:02:06.262,176,800,00; Good
'7321 {System.Int32} @2019-10-06T10:02:06.762,839,800,00; Good
'7397 {System.Int32} @2019-10-06T10:02:07.263,598,900,00; Good
'7454 {System.Int32} @2019-10-06T10:02:07.764,168,900,00; Good
'7472 {System.Int32} @2019-10-06T10:02:08.264,350,400,00; Good
```

```
'...
```

13.2.11.5 Examples - OPC UA PubSub - MQTT JSON mapping using TCP

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection with MQTT JSON mapping using
// TCP.
//
// The following assemblies need to be referenced in your project (or otherwise made
available, together with their
// dependencies) for the MQTT transport and JSON message mapping to work.
// - OpcLabs.MqttNet
// - OpcLabs.UAPubSubJson
// - Newtonsoft.Json
// Refer to the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Engine;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void MqttJsonTcp()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            // Default port with MQTT is 1883.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "mqtt://opcua-
pubsub.demo-this.com";
            // Specify the transport protocol mapping.
            pubSubConnectionDescriptor.TransportProfileUriString =
UAPubSubTransportProfileUriStrings.MqttJson;

            // Define the arguments for subscribing to the dataset, specifying the MQTT
topic name.
            var subscribeDataSetArguments = new
UASubscribeDataSetArguments(pubSubConnectionDescriptor)
            {
                DataSetSubscriptionDescriptor = {CommunicationParameters =
{BrokerDataSetReaderTransportParameters =
                {
                    QueueName = "opcuademo/json"
                }}}
            };
```

```
            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_MqttJsonTcp;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_MqttJsonTcp(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful
connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessage}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: 2020-01-21T17:07:19.778,836,700,00; Good; Data; publisher=
[String]31, class=eae79794-1af7-4f96-8401-4096cd1d8908, fields: 4
        //[BoolToggle, True {System.Boolean} @2020-01-21T16:07:19.778,836,700,00; Good]
        //[Int32, 482 {System.Int64} @2020-01-21T16:07:19.778,836,700,00; Good]
        //[Int32Fast, 2287 {System.Int64} @2020-01-21T16:07:19.778,836,700,00; Good]
        //[DateTime, 1/21/2020 5:07:19 PM {System.DateTime} @2020-01-
```

```
21T16:07:19.778,836,700,00; Good]
        //
        //Dataset data: Good; Data; publisher=[String]32, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 482 {System.Int32}; Good]
        //[Int32Fast, 2287 {System.Int32}; Good]
        //[DateTime, 1/21/2020 5:07:19 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=[String]32, fields: 100
        //[Mass_0, 82 {System.Int64}; Good]
        //[Mass_1, 182 {System.Int64}; Good]
        //[Mass_2, 282 {System.Int64}; Good]
        //[Mass_3, 382 {System.Int64}; Good]
        //[Mass_4, 482 {System.Int64}; Good]
        //[Mass_5, 582 {System.Int64}; Good]
        //[Mass_6, 682 {System.Int64}; Good]
        //[Mass_7, 782 {System.Int64}; Good]
        //[Mass_8, 882 {System.Int64}; Good]
        //[Mass_9, 982 {System.Int64}; Good]
        //[Mass_10, 1082 {System.Int64}; Good]
        ////...
    }
}
```

13.2.11.6 Examples - OPC UA PubSub - MQTT UADP mapping using TCP

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub
connection with MQTT UADP mapping using
// TCP.
//
// The OpcLabs.MqttNet assembly needs to be referenced in your project (or otherwise
made available, together with its
// dependencies) for the MQTT transport to work. Refer to the documentation for more
information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Engine;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void MqttUadpTcp()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion
from a string.
            // Default port with MQTT is 1883.
```

```
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "mqtt://opcua-
pubsub.demo-this.com";
            // Specify the transport protocol mapping.
            pubSubConnectionDescriptor.TransportProfileUriString =
UAPubSubTransportProfileUriStrings.MqttUadp;

            // Define the arguments for subscribing to the dataset, specifying the MQTT
topic name.
            var subscribeDataSetArguments = new
UASubscribeDataSetArguments(pubSubConnectionDescriptor)
            {
                DataSetSubscriptionDescriptor = {CommunicationParameters =
{BrokerDataSetReaderTransportParameters =
                {
                    QueueName = "opcuademo/uadp/none"
                }}}
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_MqttUadpTcp;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_MqttUadpTcp(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful
connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
```

```
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessage}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-
8401-4096cd1d8908, fields: 4
        //[#0, False {System.Boolean}; Good]
        //[#1, 6685 {System.Int32}; Good]
        //[#2, 1444 {System.Int32}; Good]
        //[#3, 1/4/2020 6:06:20 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-
a715-0979884b55ae, fields: 100
        //[#0, 85 {System.Int64}; Good]
        //[#1, 185 {System.Int64}; Good]
        //[#2, 285 {System.Int64}; Good]
        //[#3, 385 {System.Int64}; Good]
        //[#4, 485 {System.Int64}; Good]
        //[#5, 585 {System.Int64}; Good]
        //[#6, 685 {System.Int64}; Good]
        //[#7, 785 {System.Int64}; Good]
        //[#8, 885 {System.Int64}; Good]
        //[#9, 985 {System.Int64}; Good]
        //[#10, 1085 {System.Int64}; Good]
        //...
    }
}
```

13.2.11.7 Examples - OPC UA PubSub - Obtain all published datasets

The example below uses the ListAllPublishedDataSetNames Method to retrieve all published dataset from the PubSub configuration. The published datasets are actually organized in the PubSub configuration using a hierarchical structure of dataset folders. The word "all" in the method name denotes that the method will truly return all published datasets in the configuration, not just those at the "root" of the dataset folder structure, or in any specified dataset folder. The method will act recursively in the dataset folder structure, if needed. There are also methods that allow you to work with contents of individual dataset folders. Note that the published dataset names are unique across the PubSub configuration as a whole (regardless of their location in the dataset folder structure), and therefore a published dataset name is sufficient to identify the published dataset, and the dataset folder path is not strictly necessary.

C#

```
// This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.
```

```
using System;
using OpcLabs.BaseLib.Collections.Specialized;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions;

namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
{
    partial class GetMethods
    {
        public static void PublishedDataSets()
        {
            // Instantiate the publish-subscribe client object.
            var publishSubscribeClient = new EasyUAPublishSubscribeClient();

            try
            {
                Console.WriteLine("Loading the configuration...");
                // Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                // specified that it has to be copied to the project's output
directory.
                IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary");

                // Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                // Server with appropriate information model.
                //IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                //
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010");

                // Get the names of all published datasets in the PubSub configuration.
                StringCollection publishedDataSetNames =
pubSubConfiguration.ListAllPublishedDataSetNames();

                foreach (string publishedDataSetName in publishedDataSetNames)
                {
                    Console.WriteLine($"Published dataset: {publishedDataSetName}");

                    // You can use the statement below to obtain parameters of the
published dataset.
                    //UAPublishedDataSetElement publishedDataSetElement =
                    //
pubSubConfiguration.GetPublishedDataSetElement(publishedDataSetName);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine($"*** Failure:
{uaException.InnerException.Message}");
            }

            Console.WriteLine("Finished.");
```

```
        }

        // Example output:
        //
        //Loading the configuration...
        //Published dataset: Simple
        //Published dataset: AllTypes
        //Published dataset: MassTest
        //Published dataset: AllTypes-Dynamic
        //Finished.
    }
}
```

VB.NET

```
' This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions

Namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
    Partial Friend Class GetMethods
        Public Shared Sub PublishedDataSets()

            ' Instantiate the publish-subscribe client object.
            Dim publishSubscribeClient = New EasyUAPublishSubscribeClient()

            Try
                Console.WriteLine("Loading the configuration...")
                ' Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                ' specified that it has to be copied to the project's output directory.
                Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary")

                ' Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                ' Server with appropriate information model.
                'Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                '
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010")

                ' Get the names of all published datasets in the PubSub configuration.
                Dim publishedDataSetNames =
pubSubConfiguration.ListAllPublishedDataSetNames()

                For Each publishedDataSetName As String In publishedDataSetNames
                    Console.WriteLine($"Published dataset: {publishedDataSetName}")

                    ' You can use the statement below to obtain parameters of the
published dataset.
                    'Dim publishedDataSetElement As UAPublishedDataSetElement =
```

```
                    '
pubSubConfiguration.GetPublishedDataSetElement(publishedDataSetName)
                Next publishedDataSetName
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Loading the configuration...
    'Published dataset Simple
    'Published dataset: AllTypes
    'Published dataset: MassTest
    'Published dataset: AllTypes-Dynamic
    'Finished.

End Namespace
```

Object Pascal

```
// This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

class procedure GetMethods.PublishedDataSets;
var
  //EndpointDescriptor: _UAEndpointDescriptor;
  I: Integer;
  PublishedDataSetName: string;
  PublishedDataSetNames: _StringCollection;
  //PublishedDataSetElement: _UAPublishedDataSetElement;
  PublishSubscribeClient: _EasyUAPublishSubscribeClient;
  PubSubConfiguration: _UAReadOnlyPubSubConfiguration;
begin
  // Instantiate the publish-subscribe client object.
  PublishSubscribeClient := CoEasyUAPublishSubscribeClient.Create;

  try
    WriteLn('Loading the configuration...');
    // Load the PubSub configuration from a file. The file itself is included alongside
the script.
    PubSubConfiguration :=
PublishSubscribeClient.LoadReadOnlyConfiguration('UADemoPublisher-Default.uabinary');

    // Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    // with appropriate information model.
    //EndpointDescriptor := CoUAEndpointDescriptor.Create;
    //EndpointDescriptor.UrlString := 'opc.tcp://localhost:48010';
    //PubSubConfiguration :=
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor);

    // Get the names of PubSub connections in the configuration, regardless of the
```

```
folder they reside in.
    PublishedDataSetNames := PubSubConfiguration.ListAllPublishedDataSetNames;

    for I := 0 to PublishedDataSetNames.Count-1 do
    begin
      PublishedDataSetName := PublishedDataSetNames[I];
      WriteLn('Published dataset: ', PublishedDataSetName);

      // You can use the statement below to obtain parameters of the published dataset.
      //PublishedDataSetElement :=
PubSubConfiguration.GetPublishedDataSetElement(Unassigned, PublishedDataSetName);
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Finished.');
end;

// Example output:
//
//Loading the configuration...
//Published dataset: Simple
//Published dataset: AllTypes
//Published dataset: MassTest
//Published dataset: AllTypes-Dynamic
//Finished.
```

VBScript

```
Rem This example obtains and prints out information about all published datasets in the
OPC UA PubSub configuration.

Option Explicit

' Instantiate the publish-subscribe client object.
Dim PublishSubscribeClient: Set PublishSubscribeClient =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.InformationModel.EasyUAPublishSubscribeClient")

On Error Resume Next
DumpPublishedDataSets
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Finished."



Sub DumpPublishedDataSets()
    WScript.Echo "Loading the configuration..."
    ' Load the PubSub configuration from a file. The file itself is included alongside
```

```
the script.
    Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-Default.uabinary")

    ' Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    ' with appropriate information model.
    'Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
    'EndpointDescriptor.UrlString = "opc.tcp://localhost:48010"
    'Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor)

    ' Get the names of PubSub connections in the configuration, regardless of the
folder they reside in.
    Dim PublishedDataSetNames: Set PublishedDataSetNames =
PubSubConfiguration.ListAllPublishedDataSetNames

    Dim publishedDataSetName: For Each publishedDataSetName In PublishedDataSetNames
        WScript.Echo "Published dataset: " & publishedDataSetName

        ' You can use the statement below to obtain parameters of the published
dataset.
        'Dim PublishedDataSetElement: Set PublishedDataSetElement =
PubSubConfiguration.GetPublishedDataSetElement(Nothing, publishedDataSetName)
    Next
End Sub



' Example output:
'
'Loading the configuration...
'Published dataset: Simple
'Published dataset: AllTypes
'Published dataset: MassTest
'Published dataset: AllTypes-Dynamic
'Finished.
```

13.2.11.8 Examples - OPC UA PubSub - Obtain all PubSub components

The example below starts at the "root" of the PubSub configuration, and first obtains names of all PubSub connections available (they are at the 1st level). For each PubSub connection, given its name, it obtains names of writer groups configured on that PubSub connection (they are at the 2nd level). And, for each such writer group, given its name, it obtains names of all dataset writers configured on that writer group (they are at the 3rd level). Besides the PubSub object names, the commented parts also show how to obtain more detailed information about each PubSub object.

C#

```
// This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
// OPC UA PubSub configuration.

using System;
using OpcLabs.BaseLib.Collections.Specialized;
using OpcLabs.EasyOpc.UA.OperationModel;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel;
using OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions;

namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
{
    partial class GetMethods
    {
        public static void PubSubComponents()
        {
            // Instantiate the publish-subscribe client object.
            var publishSubscribeClient = new EasyUAPublishSubscribeClient();

            try
            {
                Console.WriteLine("Loading the configuration...");
                // Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                // specified that it has to be copied to the project's output
directory.
                IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary");

                // Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA Server
                // with appropriate information model.
                //IUAReadOnlyPubSubConfiguration pubSubConfiguration =
                //
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010");

                // Get the names of PubSub connections in the configuration, regardless
of the folder they reside in.
                StringCollection pubSubConnectionNames =
pubSubConfiguration.ListPubSubConnectionNames();
                foreach (string pubSubConnectionName in pubSubConnectionNames)
                {
                    Console.WriteLine($"PubSub connection: {pubSubConnectionName}");

                    // You can use the statement below to obtain parameters of the
PubSub connection.
                    //UAPubSubConnectionElement connectionElement =
                    //
pubSubConfiguration.GetConnectionElement(pubSubConnectionName);

                    // Get names of the writer groups on this PubSub connection.
                    StringCollection writerGroupNames =
pubSubConfiguration.ListWriterGroupNames(pubSubConnectionName);
                    foreach (string writerGroupName in writerGroupNames)
                    {
```

```
                        Console.WriteLine($"  Writer group: {writerGroupName}");

                        // You can use the statement below to obtain parameters of the
writer group.
                        //UAWriterGroupElement writerGroupElement =
                        //
pubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName);

                        // Get names of the dataset writers on this writer group.
                        StringCollection dataSetWriterNames =

pubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName);
                        foreach (string dataSetWriterName in dataSetWriterNames)
                        {
                            Console.WriteLine($"    Dataset writer:
{dataSetWriterName}");

                            // You can use the statement below to obtain parameters of
the dataset writer.
                            //UADataSetWriterElement dataSetWriterElement =
pubSubConfiguration.GetDataSetWriterElement(
                            //    pubSubConnectionName, writerGroupName,
dataSetWriterName);
                        }
                    }
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine($"*** Failure:
{uaException.InnerException.Message}");
            }

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Loading the configuration...
        //PubSub connection: FixedLayoutConnection
        //  Writer group: FixedLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: AllTypesWriter
        //    Dataset writer: MassTestWriter
        //PubSub connection: DynamicLayoutConnection
        //  Writer group: DynamicLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: MassTestWriter
        //    Dataset writer: AllTypes-DynamicWriter
        //    Dataset writer: EventSimpleWriter
        //PubSub connection: FlexibleLayoutConnection
        //  Writer group: FlexibleLayoutGroup
        //    Dataset writer: SimpleWriter
        //    Dataset writer: MassTestWriter
        //    Dataset writer: AllTypes-DynamicWriter
        //Finished.
    }
```

```
}
```

VB.NET

```
' This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
' OPC UA PubSub configuration.

Imports OpcLabs.EasyOpc.UA.OperationModel
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
'Imports OpcLabs.EasyOpc.UA.PubSub.Configuration.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel
Imports OpcLabs.EasyOpc.UA.PubSub.InformationModel.Extensions

Namespace UADocExamples.PubSub._IUAReadOnlyPubSubConfiguration
    Partial Friend Class GetMethods
        Public Shared Sub PubSubComponents()

            ' Instantiate the publish-subscribe client object.
            Dim publishSubscribeClient = New EasyUAPublishSubscribeClient()

            Try
                Console.WriteLine("Loading the configuration...")
                ' Load the PubSub configuration from a file. The file itself is at the
root of the project, and we have
                ' specified that it has to be copied to the project's output directory.
                Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                    publishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-
Default.uabinary")

                ' Alternatively, using the statement below, you can access a live
configuration residing in an OPC UA
                ' Server with appropriate information model.
                'Dim pubSubConfiguration As IUAReadOnlyPubSubConfiguration =
                '
publishSubscribeClient.AccessReadOnlyConfiguration("opc.tcp://localhost:48010")

                ' Get the names of PubSub connections in the configuration, regardless
of the folder they reside in.
                Dim pubSubConnectionNames =
pubSubConfiguration.ListPubSubConnectionNames()
                For Each pubSubConnectionName As String In pubSubConnectionNames
                    Console.WriteLine($"PubSub connection: {pubSubConnectionName}")

                    ' You can use the statement below to obtain parameters of the
PubSub connection.
                    'Dim connectionElement As UAPubSubConnectionElement =
                    '    pubSubConfiguration.GetConnectionElement(pubSubConnectionName)

                    ' Get names of the writer groups on this PubSub connection.
                    Dim writerGroupNames =
pubSubConfiguration.ListWriterGroupNames(pubSubConnectionName)
                    For Each writerGroupName As String In writerGroupNames
                        Console.WriteLine($"  Writer group: {writerGroupName}")

                        ' You can use the statement below to obtain parameters of the
writer group.
```

```
                        'Dim writerGroupElement As UAWriterGroupElement =
                        '
pubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName)

                        ' Get names of the dataset writers on this writer group.
                        Dim dataSetWriterNames =

pubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName)
                        For Each dataSetWriterName As String In dataSetWriterNames
                            Console.WriteLine($"    Dataset writer:
{dataSetWriterName}")

                            ' You can use the statement below to obtain parameters of
the dataset writer.
                            'Dim dataSetWriterElement As UADataSetWriterElement =
pubSubConfiguration.GetDataSetWriterElement(
                            '    pubSubConnectionName, writerGroupName,
dataSetWriterName)
                        Next dataSetWriterName
                    Next writerGroupName
                Next pubSubConnectionName
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
            End Try

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Loading the configuration...
    'PubSub connection FixedLayoutConnection
    '  Writer group: FixedLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: AllTypesWriter
    '    Dataset writer: MassTestWriter
    'PubSub connection: DynamicLayoutConnection
    '  Writer group: DynamicLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: MassTestWriter
    '    Dataset writer: AllTypes-DynamicWriter
    '    Dataset writer: EventSimpleWriter
    'PubSub connection: FlexibleLayoutConnection
    '  Writer group: FlexibleLayoutGroup
    '    Dataset writer: SimpleWriter
    '    Dataset writer: MassTestWriter
    '    Dataset writer: AllTypes-DynamicWriter
    'Finished.

End Namespace
```

Object Pascal

```
// This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
```

```
// OPC UA PubSub configuration.

class procedure GetMethods.PubSubComponents;
var
  //DataSetWriterElement: _UADataSetWriterElement;
  DataSetWriterName: string;
  DataSetWriterNames: _StringCollection;
  //EndpointDescriptor: _UAEndpointDescriptor;
  I, J, K: Integer;
  //PubSubConnectionElement: _UAPubSubConnectionElement;
  PubSubConnectionName: string;
  PubSubConnectionNames: _StringCollection;
  PublishSubscribeClient: _EasyUAPublishSubscribeClient;
  PubSubConfiguration: _UAReadOnlyPubSubConfiguration;
  //WriterGroupElement: _UAWriterGroupElement;
  WriterGroupName: string;
  WriterGroupNames: _StringCollection;
begin
  // Instantiate the publish-subscribe client object.
  PublishSubscribeClient := CoEasyUAPublishSubscribeClient.Create;

  try
    WriteLn('Loading the configuration...');
    // Load the PubSub configuration from a file. The file itself is included alongside
the script.
    PubSubConfiguration :=
PublishSubscribeClient.LoadReadOnlyConfiguration('UADemoPublisher-Default.uabinary');

    // Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    // with appropriate information model.
    //EndpointDescriptor := CoUAEndpointDescriptor.Create;
    //EndpointDescriptor.UrlString := 'opc.tcp://localhost:48010';
    //PubSubConfiguration :=
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor);

    // Get the names of PubSub connections in the configuration.
    PubSubConnectionNames := PubSubConfiguration.ListPubSubConnectionNames;

    for I := 0 to PubSubConnectionNames.Count-1 do
    begin
      PubSubConnectionName := PubSubConnectionNames[I];
      WriteLn('PubSub connection: ', PubSubConnectionName);

      // You can use the statement below to obtain parameters of the PubSub connection.
      //PubSubConnectionElement :=
PubSubConfiguration.GetConnectionElement(PubSubConnectionName);

      // Get names of the writer groups on this PubSub connection.
      WriterGroupNames :=
PubSubConfiguration.ListWriterGroupNames(PubSubConnectionName);
      for J := 0 to WriterGroupNames.Count-1 do
      begin
        WriterGroupName := WriterGroupNames[J];
        WriteLn('  Writer group: ', WriterGroupName);

        // You can use the statement below to obtain parameters of the writer group.
```

```
        //WriterGroupElement :=
PubSubConfiguration.GetWriterGroupElement(PubSubConnectionName, WriterGroupName);

        // Get names of the dataset writers on this writer group.
        DataSetWriterNames :=
PubSubConfiguration.ListDataSetWriterNames(PubSubConnectionName, WriterGroupName);
        for K := 0 to DataSetWriterNames.Count-1 do
        begin
          DataSetWriterName := DataSetWriterNames[K];
          WriteLn('    Dataset writer: ', DataSetWriterName);

          // You can use the statement below to obtain parameters of the dataset
writer.
          //DataSetWriterElement :=
PubSubConfiguration.GetDataSetWriterElement(PubSubConnectionName, WriterGroupName,
DataSetWriterName);
        end;
      end;
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  WriteLn('Finished.');
end;

// Example output:
//
//Loading the configuration...
//PubSub connection: FixedLayoutConnection
//  Writer group: FixedLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: AllTypesWriter
//    Dataset writer: MassTestWriter
//PubSub connection: DynamicLayoutConnection
//  Writer group: DynamicLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: MassTestWriter
//    Dataset writer: AllTypes-DynamicWriter
//    Dataset writer: EventSimpleWriter
//PubSub connection: FlexibleLayoutConnection
//  Writer group: FlexibleLayoutGroup
//    Dataset writer: SimpleWriter
//    Dataset writer: MassTestWriter
//    Dataset writer: AllTypes-DynamicWriter
//Finished.
```

VBScript

```
Rem This example obtains and prints out information about PubSub connections, writer
groups, and dataset writers in the
Rem OPC UA PubSub configuration.

Option Explicit
```

```
' Instantiate the publish-subscribe client object.
Dim PublishSubscribeClient: Set PublishSubscribeClient =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.InformationModel.EasyUAPublishSubscribeClient")

On Error Resume Next
DumpPubSubComponents
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Finished."



Sub DumpPubSubComponents()
    WScript.Echo "Loading the configuration..."
    ' Load the PubSub configuration from a file. The file itself is included alongside
the script.
    Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.LoadReadOnlyConfiguration("UADemoPublisher-Default.uabinary")

    ' Alternatively, using the statements below, you can access a live configuration
residing in an OPC UA Server
    ' with appropriate information model.
    'Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
    'EndpointDescriptor.UrlString = "opc.tcp://localhost:48010"
    'Dim PubSubConfiguration: Set PubSubConfiguration =
PublishSubscribeClient.AccessReadOnlyConfiguration(EndpointDescriptor)

    ' Get the names of PubSub connections in the configuration.
    Dim PubSubConnectionNames: Set PubSubConnectionNames =
PubSubConfiguration.ListPubSubConnectionNames
    Dim pubSubConnectionName: For Each pubSubConnectionName In PubSubConnectionNames
        WScript.Echo "PubSub connection: " & pubSubConnectionName

        ' You can use the statement below to obtain parameters of the PubSub
connection.
        'Dim PubSubConnectionElement: Set PubSubConnectionElement =
PubSubConfiguration.GetConnectionElement(pubSubConnectionName)

        ' Get names of the writer groups on this PubSub connection.
        Dim WriterGroupNames: Set WriterGroupNames =
PubSubConfiguration.ListWriterGroupNames(pubSubConnectionName)
        Dim writerGroupName: For Each writerGroupName In WriterGroupNames
            WScript.Echo "  Writer group: " & writerGroupName

            ' You can use the statement below to obtain parameters of the writer group.
            'Dim WriterGroupElement: Set WriterGroupElement =
PubSubConfiguration.GetWriterGroupElement(pubSubConnectionName, writerGroupName)

            ' Get names of the dataset writers on this writer group.
            Dim DataSetWriterNames: Set DataSetWriterNames =
PubSubConfiguration.ListDataSetWriterNames(pubSubConnectionName, writerGroupName)
```

```
            Dim dataSetWriterName: For Each dataSetWriterName In DataSetWriterNames
                WScript.Echo "    Dataset writer: " & dataSetWriterName

                ' You can use the statement below to obtain parameters of the dataset
writer.
                'Dim DataSetWriterElement: Set DataSetWriterElement = _
                '    PubSubConfiguration.GetDataSetWriterElement(pubSubConnectionName,
writerGroupName, dataSetWriterName)
            Next
        Next
    Next
End Sub



' Example output:
'
'Loading the configuration...
'PubSub connection: FixedLayoutConnection
'  Writer group: FixedLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: AllTypesWriter
'    Dataset writer: MassTestWriter
'PubSub connection: DynamicLayoutConnection
'  Writer group: DynamicLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: MassTestWriter
'    Dataset writer: AllTypes-DynamicWriter
'    Dataset writer: EventSimpleWriter
'PubSub connection: FlexibleLayoutConnection
'  Writer group: FlexibleLayoutGroup
'    Dataset writer: SimpleWriter
'    Dataset writer: MassTestWriter
'    Dataset writer: AllTypes-DynamicWriter
'Finished.
```

13.2.11.9 Examples - OPC UA PubSub - Resolve parameters from configuration file

C#

```
// This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
// from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode
// fixed layout messages with RawData field encoding.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

using System;
using System.Collections.Generic;
```

```
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void ResolveFromFile()
        {
            // Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
            // we have. The file itself is at the root of the project, and we have specified that it has
to be copied to the
            // project's output directory.
            var pubSubResolverDescriptor = UAPubSubResolverDescriptor.File("UADemoPublisher-
Default.uabinary");

            // Define the PubSub connection we will work with, using its logical name in the PubSub
configuration.
            var pubSubConnectionDescriptor = new UAPubSubConnectionDescriptor { Name =
"FixedLayoutConnection" };
            // In some cases you may have to set the interface (network adapter) name that needs to be
used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. The writer group and the dataset writer are specified using their
logical names in the
            // PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub
connection.
            var filter = new UASubscribeDataSetFilter("FixedLayoutGroup", "SimpleWriter");

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_ResolveFromFile;
            subscriber.ResolverAccess += subscriber_ResolverAccess_ResolveFromFile;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, pubSubConnectionDescriptor, filter);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_ResolveFromFile(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
```

```
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        private static void subscriber_ResolverAccess_ResolveFromFile(object sender,
EasyUAResolverAccessEventArgs e)
        {
            // Display resolution information.
            Console.WriteLine(e);
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3072 {System.Int32}; Good]
        //[Int32Fast, 894 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3072 {System.Int32}; Good]
        //[Int32Fast, 920 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3073 {System.Int32}; Good]
        //[Int32Fast, 1003 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 3073 {System.Int32}; Good]
        //[Int32Fast, 1074 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 3074 {System.Int32}; Good]
        //[Int32Fast, 1140 {System.Int32}; Good]
        //[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
' from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the resolution
is used to decode
' fixed layout messages with RawData field encoding.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation,
see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.
```

```
Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub ResolveFromFile()

            ' Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
            ' we have. The file itself is at the root of the project, and we have specified that it has
to be copied to the
            ' project's output directory.
            Dim pubSubResolverDescriptor = UAPubSubResolverDescriptor.File("UADemoPublisher-
Default.uabinary")

            ' Define the PubSub connection we will work with, using its logical name in the PubSub
configuration.
            Dim pubSubConnectionDescriptor = New UAPubSubConnectionDescriptor
            pubSubConnectionDescriptor.Name = "FixedLayoutConnection"
            ' In some cases you may have to set the interface (network adapter) name that needs to be
used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. The writer group And the dataset writer are specified using their
logical names in the
            ' PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub
connection.
            Dim filter = New UASubscribeDataSetFilter("FixedLayoutGroup", "SimpleWriter")

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_ResolveFromFile
            AddHandler subscriber.ResolverAccess, AddressOf subscriber_ResolverAccess_ResolveFromFile

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, pubSubConnectionDescriptor, filter)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_ResolveFromFile(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
```

```
            End If
        End Sub

        Private Shared Sub subscriber_ResolverAccess_ResolveFromFile(ByVal sender As Object, ByVal e As
EasyUAResolverAccessEventArgs)
            ' Display resolution information.
            Console.WriteLine(e)
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    '[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 894 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 920 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1003 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1074 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 3074 {System.Int32}; Good]
    '[Int32Fast, 1140 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
// from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode
// fixed layout messages with RawData field encoding.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

type
  TSubscriberEventHandlers81 = class
    procedure OnDataSetMessage(
```

```
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
    procedure OnResolverAccess(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUAResolverAccessEventArgs);
  end;

class procedure SubscribeDataSet.ResolveFromFile;
var
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers81;
begin
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;

  // Define the PubSub resolver. We want the information be resolved from a PubSub binary configuration
file that
  // we have. The file itself is included alongside the script.
  SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString :=
'UADemoPublisher-Default.uabinary';
  SubscribeDataSetArguments.ResolverDescriptor.ResolverKind := UAPubSubResolverKind_PublisherFile;

  // Define the PubSub connection we will work with, using its logical name in the PubSub configuration.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.Name :=
'FixedLayoutConnection';
  // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
  // the statement below. Your actual interface name may differ, of course.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.ResourceAddress.InterfaceName
:= 'Ethernet';

  // Define the filter. The writer group and the dataset writer are specified using their logical names
in the
  // PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub
connection.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.Name :=
'FixedLayoutGroup';
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.Name :=
'SimpleWriter';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers81.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;
  Subscriber.OnResolverAccess := SubscriberEventHandlers.OnResolverAccess;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers81.OnDataSetMessage(
```

```
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

procedure TSubscriberEventHandlers81.OnResolverAccess(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUAResolverAccessEventArgs);
begin
  // Display resolution information.
  WriteLn(eventArgs.ToString);
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 894 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 920 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1003 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1074 {System.Int32}; Good]
```

```
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 3074 {System.Int32}; Good]
//[Int32Fast, 1140 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and specify a filter, resolving logical
parameters to physical
Rem from an OPC-UA PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode
Rem fixed layout messages with RawData field encoding.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface
name to be used.

Option Explicit

Const UAPubSubResolverKind_PublisherFile = 3

Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")

' Define the PubSub resolver. We want the information be resolved from a PubSub binary configuration file
that
' we have. The file itself is included alongside the script.
SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString =
"UADemoPublisher-Default.uabinary"
SubscribeDataSetArguments.ResolverDescriptor.ResolverKind = UAPubSubResolverKind_PublisherFile

' Define the PubSub connection we will work with, using its logical name in the PubSub configuration.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.Name =
"FixedLayoutConnection"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly
to
' the statement below. Your actual interface name may differ, of course.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor.ResourceAddress.InterfaceName
= "Ethernet"

' Define the filter. The writer group and the dataset writer are specified using their logical names in
the
' PubSub configuration. The publisher Id in the filter will be taken from the logical PubSub connection.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.Name =
"FixedLayoutGroup"
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.Name =
"SimpleWriter"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
```

```
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub

Sub Subscriber_ResolverAccess(Sender, e)
    ' Display resolution information.
    WScript.Echo e
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'[PublisherFile: UADemoPublisher-Default.uabinary] (no exception)
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 894 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 920 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1003 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1074 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 3074 {System.Int32}; Good]
'[Int32Fast, 1140 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
'
'...
```

13.2.11.10 Examples - OPC UA PubSub - Resolve parameters given published dataset name

C#

```
// This example shows how to subscribe to dataset messages, specifying just the
published dataset name, and resolving all
// the dataset subscription arguments from an OPC-UA PubSub configuration file.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void PublishedDataSet()
        {
            // Define the PubSub resolver. We want the information be resolved from a
PubSub binary configuration file that
            // we have. The file itself is at the root of the project, and we have
specified that it has to be copied to the
            // project's output directory.
            var pubSubResolverDescriptor =
UAPubSubResolverDescriptor.File("UADemoPublisher-Default.uabinary");

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            Console.WriteLine("Subscribing...");
            // Specify the published dataset name, and let all other subscription
arguments be resolved automatically.
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, "AllTypes-Dynamic",
(sender, args) =>
            {
                // Display the dataset.
                if (args.Succeeded)
                {
                    // An event with null DataSetData just indicates a successful
connection.
                    if (!(args.DataSetData is null))
                    {
                        Console.WriteLine();
```

```
                        Console.WriteLine($"Dataset data: {args.DataSetData}");
                        foreach (KeyValuePair<string, UADataSetFieldData> pair in
args.DataSetData.FieldDataDictionary)
                            Console.WriteLine(pair);
                    }
                }
                else
                {
                    Console.WriteLine();
                    Console.WriteLine($"*** Failure: {args.ErrorMessageBrief}");
                }
            });

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
        //[BoolToggle, False {System.Boolean}; Good]
        //[Byte, 137 {System.Byte}; Good]
        //[Int16, 10377 {System.Int16}; Good]
        //[Int32, 43145 {System.Int32}; Good]
        //[Int64, 43145 {System.Int64}; Good]
        //[SByte, 9 {System.Int16}; Good]
        //[UInt16, 43145 {System.Int32}; Good]
        //[UInt32, 43145 {System.Int64}; Good]
        //[UInt64, 43145 {System.Decimal}; Good]
        //[Float, 43145 {System.Single}; Good]
        //[Double, 43145 {System.Double}; Good]
        //[String, Lima {System.String}; Good]
        //[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
        //[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
        //[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
        //[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]};
Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
        //[BoolToggle, True {System.Boolean}; Good]
        //[Byte, 138 {System.Byte}; Good]
        //[Int16, 10378 {System.Int16}; Good]
        //[Int32, 43146 {System.Int32}; Good]
        //[Int64, 43146 {System.Int64}; Good]
```

```
        //[SByte, 10 {System.Int16}; Good]
        //[UInt16, 43146 {System.Int32}; Good]
        //[UInt32, 43146 {System.Int64}; Good]
        //[UInt64, 43146 {System.Decimal}; Good]
        //[Float, 43146 {System.Single}; Good]
        //[Double, 43146 {System.Double}; Good]
        //[String, Mike {System.String}; Good]
        //[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
        //[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
        //[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]};
Good]
        //[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
        //[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
        //[BoolToggle, True {System.Boolean}; Good]
        //[Byte, 138 {System.Byte}; Good]
        //[Int16, 10378 {System.Int16}; Good]
        //[Int32, 43146 {System.Int32}; Good]
        //[Int64, 43146 {System.Int64}; Good]
        //[SByte, 10 {System.Int16}; Good]
        //[UInt16, 43146 {System.Int32}; Good]
        //[UInt32, 43146 {System.Int64}; Good]
        //[UInt64, 43146 {System.Decimal}; Good]
        //[Float, 43146 {System.Single}; Good]
        //[Double, 43146 {System.Double}; Good]
        //[String, Mike {System.String}; Good]
        //[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
        //[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
        //[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]};
Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages, specifying just the
published dataset name, and resolving all
' the dataset subscription arguments from an OPC-UA PubSub configuration file.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
the interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub PublishedDataSet()

            ' Define the PubSub resolver. We want the information be resolved from a
PubSub binary configuration file that
            ' we have. The file itself is at the root of the project, and we have
specified that it has to be copied to the
```

```
            ' project's output directory.
            Dim pubSubResolverDescriptor =
UAPubSubResolverDescriptor.File("UADemoPublisher-Default.uabinary")

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            Console.WriteLine("Subscribing...")
            ' Specify the published dataset name, And let all other subscription
arguments be resolved automatically.
            subscriber.SubscribeDataSet(pubSubResolverDescriptor, "AllTypes-Dynamic",
                Sub(sender, eventArgs)
                    '  Display the dataset.
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful
connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}",
eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String,
UADataSetFieldData) In eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                    End If
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
    '[BoolToggle, False {System.Boolean}; Good]
    '[Byte, 137 {System.Byte}; Good]
    '[Int16, 10377 {System.Int16}; Good]
    '[Int32, 43145 {System.Int32}; Good]
```

```
    '[Int64, 43145 {System.Int64}; Good]
    '[SByte, 9 {System.Int16}; Good]
    '[UInt16, 43145 {System.Int32}; Good]
    '[UInt32, 43145 {System.Int64}; Good]
    '[UInt64, 43145 {System.Decimal}; Good]
    '[Float, 43145 {System.Single}; Good]
    '[Double, 43145 {System.Double}; Good]
    '[String, Lima {System.String}; Good]
    '[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
    '[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
    '[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
    '[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]};
Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
    '[BoolToggle, True {System.Boolean}; Good]
    '[Byte, 138 {System.Byte}; Good]
    '[Int16, 10378 {System.Int16}; Good]
    '[Int32, 43146 {System.Int32}; Good]
    '[Int64, 43146 {System.Int64}; Good]
    '[SByte, 10 {System.Int16}; Good]
    '[UInt16, 43146 {System.Int32}; Good]
    '[UInt32, 43146 {System.Int64}; Good]
    '[UInt64, 43146 {System.Decimal}; Good]
    '[Float, 43146 {System.Single}; Good]
    '[Double, 43146 {System.Double}; Good]
    '[String, Mike {System.String}; Good]
    '[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
    '[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
    '[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]};
Good]
    '[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
    '[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
    '[BoolToggle, True {System.Boolean}; Good]
    '[Byte, 138 {System.Byte}; Good]
    '[Int16, 10378 {System.Int16}; Good]
    '[Int32, 43146 {System.Int32}; Good]
    '[Int64, 43146 {System.Int64}; Good]
    '[SByte, 10 {System.Int16}; Good]
    '[UInt16, 43146 {System.Int32}; Good]
    '[UInt32, 43146 {System.Int64}; Good]
    '[UInt64, 43146 {System.Decimal}; Good]
    '[Float, 43146 {System.Single}; Good]
    '[Double, 43146 {System.Double}; Good]
    '[String, Mike {System.String}; Good]
    '[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
    '[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
    '[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]};
Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages, specifying just the
published dataset name, and resolving all
// the dataset subscription arguments from an OPC-UA PubSub configuration file.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

type
  TSubscriberEventHandlers79 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.PublishedDataSet;
var
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers79;
begin
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;

  // Define the PubSub resolver. We want the information be resolved from a PubSub
binary configuration file that
  // we have. The file itself is included alongside the script.


SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString
:= 'UADemoPublisher-Default.uabinary';
  SubscribeDataSetArguments.ResolverDescriptor.ResolverKind :=
UAPubSubResolverKind_PublisherFile;

  // Specify the published dataset name, and let all other subscription arguments be
resolved automatically.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.PublishedDataSetName :=
'AllTypes-Dynamic';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers79.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);
```

```
  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers79.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator :=
eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[BoolToggle, False {System.Boolean}; Good]
//[Byte, 137 {System.Byte}; Good]
//[Int16, 10377 {System.Int16}; Good]
//[Int32, 43145 {System.Int32}; Good]
//[Int64, 43145 {System.Int64}; Good]
//[SByte, 9 {System.Int16}; Good]
//[UInt16, 43145 {System.Int32}; Good]
//[UInt32, 43145 {System.Int64}; Good]
//[UInt64, 43145 {System.Decimal}; Good]
//[Float, 43145 {System.Single}; Good]
//[Double, 43145 {System.Double}; Good]
//[String, Lima {System.String}; Good]
```

```
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
//[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
//[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[BoolToggle, True {System.Boolean}; Good]
//[Byte, 138 {System.Byte}; Good]
//[Int16, 10378 {System.Int16}; Good]
//[Int32, 43146 {System.Int32}; Good]
//[Int64, 43146 {System.Int64}; Good]
//[SByte, 10 {System.Int16}; Good]
//[UInt16, 43146 {System.Int32}; Good]
//[UInt32, 43146 {System.Int64}; Good]
//[UInt64, 43146 {System.Decimal}; Good]
//[Float, 43146 {System.Single}; Good]
//[Double, 43146 {System.Double}; Good]
//[String, Mike {System.String}; Good]
//[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
//[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
//[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Byte, 138 {System.Byte}; Good]
//[Int16, 10378 {System.Int16}; Good]
//[Int32, 43146 {System.Int32}; Good]
//[Int64, 43146 {System.Int64}; Good]
//[SByte, 10 {System.Int16}; Good]
//[UInt16, 43146 {System.Int32}; Good]
//[UInt32, 43146 {System.Int64}; Good]
//[UInt64, 43146 {System.Decimal}; Good]
//[Float, 43146 {System.Single}; Good]
//[Double, 43146 {System.Double}; Good]
//[String, Mike {System.String}; Good]
//[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
//[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
//[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages, specifying just the
published dataset name, and resolving all
Rem the dataset subscription arguments from an OPC-UA PubSub configuration file.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher
tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

Option Explicit

Const UAPubSubResolverKind_PublisherFile = 3
```

```
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")


' Define the PubSub resolver. We want the information be resolved from a PubSub binary
configuration file that
' we have. The file itself is included alongside the script.
SubscribeDataSetArguments.ResolverDescriptor.PublisherFileResourceDescriptor.UrlString
= "UADemoPublisher-Default.uabinary"
SubscribeDataSetArguments.ResolverDescriptor.ResolverKind =
UAPubSubResolverKind_PublisherFile

' Specify the published dataset name, and let all other subscription arguments be
resolved automatically.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.PublishedDataSetName =
"AllTypes-Dynamic"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub
```

```
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[BoolToggle, False {System.Boolean}; Good]
'[Byte, 137 {System.Byte}; Good]
'[Int16, 10377 {System.Int16}; Good]
'[Int32, 43145 {System.Int32}; Good]
'[Int64, 43145 {System.Int64}; Good]
'[SByte, 9 {System.Int16}; Good]
'[UInt16, 43145 {System.Int32}; Good]
'[UInt32, 43145 {System.Int64}; Good]
'[UInt64, 43145 {System.Decimal}; Good]
'[Float, 43145 {System.Single}; Good]
'[Double, 43145 {System.Double}; Good]
'[String, Lima {System.String}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'[Guid, 45a99b50-e265-41f2-adea-d0bcedc3ff4b {System.Guid}; Good]
'[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
'[UInt32Array, [10] {43145, 43146, 43147, 43148, 43149, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[BoolToggle, True {System.Boolean}; Good]
'[Byte, 138 {System.Byte}; Good]
'[Int16, 10378 {System.Int16}; Good]
'[Int32, 43146 {System.Int32}; Good]
'[Int64, 43146 {System.Int64}; Good]
'[SByte, 10 {System.Int16}; Good]
'[UInt16, 43146 {System.Int32}; Good]
'[UInt32, 43146 {System.Int64}; Good]
'[UInt64, 43146 {System.Decimal}; Good]
'[Float, 43146 {System.Single}; Good]
'[Double, 43146 {System.Double}; Good]
'[String, Mike {System.String}; Good]
'[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
'[DateTime, 10/3/2019 7:15:34 AM {System.DateTime}; Good]
'[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[DateTime, 10/3/2019 7:15:35 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Byte, 138 {System.Byte}; Good]
'[Int16, 10378 {System.Int16}; Good]
'[Int32, 43146 {System.Int32}; Good]
'[Int64, 43146 {System.Int64}; Good]
'[SByte, 10 {System.Int16}; Good]
'[UInt16, 43146 {System.Int32}; Good]
'[UInt32, 43146 {System.Int64}; Good]
'[UInt64, 43146 {System.Decimal}; Good]
'[Float, 43146 {System.Single}; Good]
'[Double, 43146 {System.Double}; Good]
'[String, Mike {System.String}; Good]
```

```
'[ByteString, [20] {176, 63, 39, 37, 31, ...} {System.Byte[]}; Good]
'[Guid, a0f06d75-9896-4fa3-9724-b564359da21b {System.Guid}; Good]
'[UInt32Array, [10] {43146, 43147, 43148, 43149, 43150, ...} {System.Int64[]}; Good]
'
'...
```

13.2.11.11 Examples - OPC UA PubSub - Set message mapping parameters

C#

```
// This example shows how to set parameters specific to JSON message mapping.
//
// The following assemblies need to be referenced in your project (or otherwise made available,
together with their
// dependencies) for the MQTT transport and JSON message mapping to work.
// - OpcLabs.MqttNet
// - OpcLabs.UAPubSubJson
// - Newtonsoft.Json
// Refer to the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.Engine;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void MappingParameters()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a
string.
            // Default port with MQTT is 1883.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "mqtt://opcua-pubsub.demo-
this.com";
            // Specify the transport protocol mapping.
            pubSubConnectionDescriptor.TransportProfileUriString =
UAPubSubTransportProfileUriStrings.MqttJson;

            // Set a custom property on the PubSub connection that influences how the JSON parsing
works.
            // We are instructing the message parser to turn off the automatic recognition of
message format.
            // For more details, see http://kb.opclabs.com/OPC_UA_PubSub_JSON_mapping_package .
            pubSubConnectionDescriptor.CustomPropertyValueDictionary[new
UAQualifiedName("http://opclabs.com/OpcUA/PubSub",

"OpcLabs.UAPubSubJson.JsonReceiveMessageMapping.MessageParsingParameters.AutoRecognizeMessageFormat")]
=
                false;

            // Define the arguments for subscribing to the dataset.
```

```
            var subscribeDataSetArguments = new
UASubscribeDataSetArguments(pubSubConnectionDescriptor)
            {
                DataSetSubscriptionDescriptor =
                {
                    CommunicationParameters =
                    {
                        BrokerDataSetReaderTransportParameters = {QueueName = "opcuademo/json"},
                        // We must set the DataSetFieldContentMask when the format auto-recognition
is turned off.
                        DataSetFieldContentMask = UADataSetFieldContentMask.RawData,
                        JsonDataSetReaderMessageParameters =
                        {
                            // We must set the DataSetMessageContentMask when the format auto-
recognition is turned off.
                            DataSetMessageContentMask =
                                UAJsonDataSetMessageContentMask.DataSetWriterId |
                                UAJsonDataSetMessageContentMask.SequenceNumber |
                                UAJsonDataSetMessageContentMask.Status,
                            // We must set the NetworkMessageContentMask when the format auto-
recognition is turned off.
                            NetworkMessageContentMask =
                                UAJsonNetworkMessageContentMask.NetworkMessageHeader |
                                UAJsonNetworkMessageContentMask.DataSetMessageHeader |
                                UAJsonNetworkMessageContentMask.PublisherId
                        }
                    },
                    Filter =
                    {
                        DataSetWriterDescriptor = 1,
                    }
                }
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_MappingParameters;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_MappingParameters(object sender,
EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
```

```
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessage}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=[String]30, writer=1, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 9034 {System.Int64}; Good]
        //[Int32Fast, 1751 {System.Int64}; Good]
        //[DateTime, 1/30/2020 5:23:11 PM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=[String]30, writer=1, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 9036 {System.Int64}; Good]
        //[Int32Fast, 2526 {System.Int64}; Good]
        //[DateTime, 1/30/2020 5:23:13 PM {System.DateTime}; Good]
        //...
    }
}
```

13.2.11.12 Examples - OPC UA PubSub - Specify field names for UADP message

C#

```
// This example shows how to subscribe to dataset messages and specify field names, without having the full
metadata.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void FieldNames()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1);

            // Define the metadata, with the use of collection initializer for its fields. For UADP, the order of
```

```
field
            // metadata must correspond to the order of fields in the dataset message.
            // Since the encoding is not RawData, we do not have to specify the type information for the fields.
            var metaData = new UADataSetMetaData
            {
                new UAFieldMetaData("BoolToggle"),
                new UAFieldMetaData("Int32"),
                new UAFieldMetaData("Int32Fast"),
                new UAFieldMetaData("DateTime")
            };

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_FieldNames;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_FieldNames(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[BoolToggle, True {System.Boolean}; Good]
        //[Int32, 25 {System.Int32}; Good]
        //[Int32Fast, 928 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[Int32, 26 {System.Int32}; Good]
        //[Int32Fast, 1007 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
        //[BoolToggle, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[Int32Fast, 1113 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
        //[BoolToggle, True {System.Boolean}; Good]
```

```
        //[Int32, 26 {System.Int32}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 27 {System.Int32}; Good]
        //[Int32Fast, 1201 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[Int32Fast, 1260 {System.Int32}; Good]
        //[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
        //[BoolToggle, False {System.Boolean}; Good]
        //[Int32, 27 {System.Int32}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and specify field names, without having the full
metadata.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub FieldNames()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) Is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1)

            ' Define the metadata, with the use of collection initializer for its fields. For UADP, the order of
field
            ' metadata must correspond to the order of fields in the dataset message.
            ' Since the encoding Is Not RawData, we do Not have to specify the type information for the fields.
            Dim metaData = New UADataSetMetaData From {
                New UAFieldMetaData("BoolToggle"),
                New UAFieldMetaData("Int32"),
                New UAFieldMetaData("Int32Fast"),
                New UAFieldMetaData("DateTime")
            }

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_FieldNames

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter, metaData)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)
```

```
            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_FieldNames(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 25 {System.Int32}; Good]
    '[Int32Fast, 928 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[Int32, 26 {System.Int32}; Good]
    '[Int32Fast, 1007 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
    '[BoolToggle, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[Int32Fast, 1113 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 26 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 27 {System.Int32}; Good]
    '[Int32Fast, 1201 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[Int32Fast, 1260 {System.Int32}; Good]
    '[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 27 {System.Int32}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and specify field names, without having the full
metadata.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
```

```
  TSubscriberEventHandlers75 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.FieldNames;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Field1, Field2, Field3, Field4: _UAFieldMetaData;
  MetaData: _UADataSetMetaData;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers75;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Define the metadata, with the use of collection initializer for its fields. For UADP, the order of field
  // metadata must correspond to the order of fields in the dataset message.
  // Since the encoding is not RawData, we do not have to specify the type information for the fields.
  MetaData := CoUADataSetMetaData.Create;
  //
  Field1 := CoUAFieldMetaData.Create;
  Field1.Name := 'BoolToggle';
  MetaData.Add(Field1);
  //
  Field2 := CoUAFieldMetaData.Create;
  Field2.Name := 'Int32';
  MetaData.Add(Field2);
  //
  Field3 := CoUAFieldMetaData.Create;
  Field3.Name := 'Int32Fast';
  MetaData.Add(Field3);
  //
  Field4 := CoUAFieldMetaData.Create;
  Field4.Name := 'DateTime';
  MetaData.Add(Field4);
  //
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData := MetaData;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers75.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
```

```
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers75.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 25 {System.Int32}; Good]
//[Int32Fast, 928 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32, 26 {System.Int32}; Good]
//[Int32Fast, 1007 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1113 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 26 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//[Int32Fast, 1201 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1260 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and specify field names, without having the full
metadata.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
' Since the encoding is not RawData, we do not have to specify the type information for the fields.
Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
'
Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field1.Name = "BoolToggle"
MetaData.Add(Field1)
'
Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field2.Name = "Int32"
MetaData.Add(Field2)
'
Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field3.Name = "Int32Fast"
MetaData.Add(Field3)
'
Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field4.Name = "DateTime"
MetaData.Add(Field4)
'
Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
```

```
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub

WScript.Echo "Finished."



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 25 {System.Int32}; Good]
'[Int32Fast, 928 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32, 26 {System.Int32}; Good]
'[Int32Fast, 1007 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1113 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 26 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'[Int32Fast, 1201 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1260 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages and specify field names, without having the full
metadata.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber5 As EasyUASubscriber

Private Sub SubscribeDataSet_FieldNames_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
```

```
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the
dataset message.
    ' Since the encoding is not RawData, we do not have to specify the type information for the fields.
    Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
    '
    Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field1.Name = "BoolToggle"
    MetaData.Add (Field1)
    '
    Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field2.Name = "Int32"
    MetaData.Add (Field2)
    '
    Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field3.Name = "Int32Fast"
    MetaData.Add (Field3)
    '
    Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field4.Name = "DateTime"
    MetaData.Add (Field4)
    '
    Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

    ' Instantiate the subscriber object and hook events.
    Set Subscriber5 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber5.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber5.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber5 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber5_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
```

```
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 25 {System.Int32}; Good]
'[Int32Fast, 928 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32, 26 {System.Int32}; Good]
'[Int32Fast, 1007 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1113 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 26 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'[Int32Fast, 1201 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[Int32Fast, 1260 {System.Int32}; Good]
'[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 27 {System.Int32}; Good]
'
'...
'
```

13.2.11.13 Examples - OPC UA PubSub - Specify filter for dataset messages

C#

```
// This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
// UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Filter()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
```

```
    var filter = new UASubscribeDataSetFilter(
        UAPublisherId.CreateUInt64(31),
        UAWriterGroupDescriptor.Null,
        dataSetWriterDescriptor: 1);

    // Instantiate the subscriber object and hook events.
    var subscriber = new EasyUASubscriber();
    subscriber.DataSetMessage += subscriber_DataSetMessage_Filter;

    Console.WriteLine("Subscribing...");
    subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter);

    Console.WriteLine("Processing dataset message events for 20 seconds...");
    Thread.Sleep(20 * 1000);

    Console.WriteLine("Unsubscribing...");
    subscriber.UnsubscribeAllDataSets();

    Console.WriteLine("Waiting for 1 second...");
    // Unsubscribe operation is asynchronous, messages may still come for a short while.
    Thread.Sleep(1 * 1000);

    Console.WriteLine("Finished.");
}

static void subscriber_DataSetMessage_Filter(object sender, EasyUADataSetMessageEventArgs e)
{
    // Display the dataset.
    if (e.Succeeded)
    {
        // An event with null DataSetData just indicates a successful connection.
        if (!(e.DataSetData is null))
        {
            Console.WriteLine();
            Console.WriteLine($"Dataset data: {e.DataSetData}");
            foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                Console.WriteLine(pair);
        }
    }
    else
    {
        Console.WriteLine();
        Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
    }
}

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, True {System.Boolean}; Good]
//[#1, 7134 {System.Int32}; Good]
//[#2, 7364 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#1, 7135 {System.Int32}; Good]
//[#2, 7429 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 7495 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//[#1, 7135 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#1, 7136 {System.Int32}; Good]
//[#2, 7560 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
```

```
        //[#0, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#2, 7626 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //[#1, 7136 {System.Int32}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
' UDP UADP mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub Filter()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 64-bits) Is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1)

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_Filter

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_Filter(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
```

```
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, True {System.Boolean}; Good]
    '[#1, 7134 {System.Int32}; Good]
    '[#2, 7364 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7135 {System.Int32}; Good]
    '[#2, 7429 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7495 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7135 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7136 {System.Int32}; Good]
    '[#2, 7560 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7626 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7136 {System.Int32}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
// UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers76 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Filter;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers76;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
```

```
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers76.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers76.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
```

```
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 25 {System.Int32}; Good]
//[Int32Fast, 928 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32, 26 {System.Int32}; Good]
//[Int32Fast, 1007 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1113 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:02 AM {System.DateTime}; Good]
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 26 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//[Int32Fast, 1201 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[Int32Fast, 1260 {System.Int32}; Good]
//[DateTime, 10/3/2019 10:43:03 AM {System.DateTime}; Good]
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 27 {System.Int32}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
Rem UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets
```

```
WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages and specify a filter, on an OPC-UA PubSub connection
with
Rem UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
```

```
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber6 As EasyUASubscriber

Private Sub SubscribeDataSet_Filter_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Instantiate the subscriber object and hook events.
    Set Subscriber6 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber6.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber6.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber6 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber6_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
```

```
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
'
```

13.2.11.14 Examples - OPC UA PubSub - Specify metadata for RawData encoding

C#

```
// This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
// for their decoding directly in the code.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Configuration;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void Metadata()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
            // The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt16(30), writerGroupDescriptor: 101);

            // Define the metadata, with the use of collection initializer for its fields.
            var metaData = new UADataSetMetaData
            {
                new UAFieldMetaData("BoolToggle", UABuiltInType.Boolean),
                new UAFieldMetaData("Int32", UABuiltInType.Int32),
                new UAFieldMetaData("Int32Fast", UABuiltInType.Int32),
                new UAFieldMetaData("DateTime", UABuiltInType.DateTime)
            };

            // Define the dataset subscription, with specific communication parameters.
            // The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
            // encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
            // which is also the case here, but we are specifying the dataset offset anyway, for illustration.
            var dataSetSubscriptionDescriptor = new UADataSetSubscriptionDescriptor(
```

```
        pubSubConnectionDescriptor, filter, metaData)
    {
        CommunicationParameters = { UadpDataSetReaderMessageParameters = { DataSetOffset = 15 }}
    };

    // Instantiate the subscriber object and hook events.
    var subscriber = new EasyUASubscriber();
    subscriber.DataSetMessage += subscriber_DataSetMessage_Metadata;

    Console.WriteLine("Subscribing...");
    subscriber.SubscribeDataSet(dataSetSubscriptionDescriptor);

    Console.WriteLine("Processing dataset message events for 20 seconds...");
    Thread.Sleep(20 * 1000);

    Console.WriteLine("Unsubscribing...");
    subscriber.UnsubscribeAllDataSets();

    Console.WriteLine("Waiting for 1 second...");
    // Unsubscribe operation is asynchronous, messages may still come for a short while.
    Thread.Sleep(1 * 1000);

    Console.WriteLine("Finished.");
}

static void subscriber_DataSetMessage_Metadata(object sender, EasyUADataSetMessageEventArgs e)
{
    // Display the dataset.
    if (e.Succeeded)
    {
        // An event with null DataSetData just indicates a successful connection.
        if (!(e.DataSetData is null))
        {
            Console.WriteLine();
            Console.WriteLine($"Dataset data: {e.DataSetData}");
            foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                Console.WriteLine(pair);
        }
    }
    else
    {
        Console.WriteLine();
        Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
    }
}

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 894 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 920 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1003 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1074 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 3074 {System.Int32}; Good]
//[Int32Fast, 1140 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
//
//...
```

```
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
' for their decoding directly in the code.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Configuration
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub Metadata()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
            ' The dataset writer Id (1) must Not be specified in the filter, because it does Not appear in the message.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt16(30))
            filter.WriterGroupDescriptor.WriterGroupId = 101

            ' Define the metadata, with the use of collection initializer for its fields.
            Dim metaData = New UADataSetMetaData From {
                New UAFieldMetaData("BoolToggle", UABuiltInType.Boolean),
                New UAFieldMetaData("Int32", UABuiltInType.Int32),
                New UAFieldMetaData("Int32Fast", UABuiltInType.Int32),
                New UAFieldMetaData("DateTime", UABuiltInType.DateTime)
            }

            ' Define the dataset subscription, with specific communication parameters.
            ' The dataset offset is needed with messages that do Not contain dataset writer Ids And use RawData field
            ' encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
            ' which is also the case here, but we are specifying the dataset offset anyway, for illustration.
            Dim dataSetSubscriptionDescriptor = New UADataSetSubscriptionDescriptor(
                pubSubConnectionDescriptor, filter, metaData)
            dataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset = 15

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_Metadata

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(dataSetSubscriptionDescriptor)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_Metadata(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
```

```
            End If
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 894 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3072 {System.Int32}; Good]
    '[Int32Fast, 920 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1003 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, False {System.Boolean}; Good]
    '[Int32, 3073 {System.Int32}; Good]
    '[Int32Fast, 1074 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
    '[BoolToggle, True {System.Boolean}; Good]
    '[Int32, 3074 {System.Int32}; Good]
    '[Int32Fast, 1140 {System.Int32}; Good]
    '[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
// for their decoding directly in the code.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

type
  TSubscriberEventHandlers78 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Metadata;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  Field1, Field2, Field3, Field4: _UAFieldMetaData;
  MetaData: _UADataSetMetaData;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers78;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
  // The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetUint16Identifier(30);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.WriterGroupId := 101;

  // Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset message.
```

```
  MetaData := CoUADataSetMetaData.Create;
  //
  Field1 := CoUAFieldMetaData.Create;
  Field1.BuiltInType := UABuiltInType_Boolean;
  Field1.Name := 'BoolToggle';
  MetaData.Add(Field1);
  //
  Field2 := CoUAFieldMetaData.Create;
  Field2.BuiltInType := UABuiltInType_Int32;
  Field2.Name := 'Int32';
  MetaData.Add(Field2);
  //
  Field3 := CoUAFieldMetaData.Create;
  Field3.BuiltInType := UABuiltInType_Int32;
  Field3.Name := 'Int32Fast';
  MetaData.Add(Field3);
  //
  Field4 := CoUAFieldMetaData.Create;
  Field4.BuiltInType := UABuiltInType_DateTime;
  Field4.Name := 'DateTime';
  MetaData.Add(Field4);
  //
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData := MetaData;

  // Define the specific communication parameters for the dataset subscription.
  // The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
  // encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
  // which is also the case here, but we are specifying the dataset offset anyway, for illustration.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset
:= 15;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers78.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers78.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
```

```
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 894 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3072 {System.Int32}; Good]
//[Int32Fast, 920 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1003 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, False {System.Boolean}; Good]
//[Int32, 3073 {System.Int32}; Good]
//[Int32Fast, 1074 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
//[BoolToggle, True {System.Boolean}; Good]
//[Int32, 3074 {System.Int32}; Good]
//[Int32Fast, 1140 {System.Int32}; Good]
//[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
Rem for their decoding directly in the code.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

Option Explicit

Const UABuiltInType_Boolean = 1
Const UABuiltInType_Int32 = 6
Const UABuiltInType_DateTime = 13

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
' The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetUInt16Identifier 30
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.WriterGroupId = 101

' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset message.
Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
'
Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field1.BuiltInType = UABuiltInType_Boolean
Field1.Name = "BoolToggle"
MetaData.Add(Field1)
'
Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field2.BuiltInType = UABuiltInType_Int32
Field2.Name = "Int32"
```

```
MetaData.Add(Field2)
'
Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field3.BuiltInType = UABuiltInType_Int32
Field3.Name = "Int32Fast"
MetaData.Add(Field3)
'
Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
Field4.BuiltInType = UABuiltInType_DateTime
Field4.Name = "DateTime"
MetaData.Add(Field4)
'
Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

' Define the specific communication parameters for the dataset subscription.
' The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
' encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
' which is also the case here, but we are specifying the dataset offset anyway, for illustration.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset
= 15

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 894 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 920 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1003 {System.Int32}; Good]
```

```
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1074 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 3074 {System.Int32}; Good]
'[Int32Fast, 1140 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages with RawData field encoding, specifying the metadata necessary
Rem for their decoding directly in the code.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be used.

' The subscriber object, with events.
'Public WithEvents Subscriber7 As EasyUASubscriber

Private Sub SubscribeDataSet_Metadata_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 16-bits) is 30, and the writer group Id is 101.
    ' The dataset writer Id (1) must not be specified in the filter, because it does not appear in the message.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetUInt16Identifier 30
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.WriterGroupDescriptor.WriterGroupId = 101

    ' Define the metadata. For UADP, the order of field metadata must correspond to the order of fields in the dataset
message.
    Dim MetaData: Set MetaData = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UADataSetMetaData")
    '
    Dim Field1: Set Field1 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field1.BuiltInType = UABuiltInType_Boolean
    Field1.Name = "BoolToggle"
    MetaData.Add (Field1)
    '
    Dim Field2: Set Field2 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field2.BuiltInType = UABuiltInType_Int32
    Field2.Name = "Int32"
    MetaData.Add (Field2)
    '
    Dim Field3: Set Field3 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field3.BuiltInType = UABuiltInType_Int32
    Field3.Name = "Int32Fast"
    MetaData.Add (Field3)
    '
    Dim Field4: Set Field4 = CreateObject("OpcLabs.EasyOpc.UA.PubSub.Configuration.UAFieldMetaData")
    Field4.BuiltInType = UABuiltInType_DateTime
    Field4.Name = "DateTime"
    MetaData.Add (Field4)
    '
    Set SubscribeDataSetArguments.DataSetSubscriptionDescriptor.DataSetMetaData = MetaData

    ' Define the specific communication parameters for the dataset subscription.
    ' The dataset offset is needed with messages that do not contain dataset writer Ids and use RawData field
    ' encoding. An exception to this rule is when the dataset is the only or first in the dataset message payload,
    ' which is also the case here, but we are specifying the dataset offset anyway, for illustration.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.CommunicationParameters.UadpDataSetReaderMessageParameters.DataSetOffset
= 15

    ' Instantiate the subscriber object and hook events.
    Set Subscriber7 = New EasyUASubscriber
```

```
    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber7.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber7.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber7 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber7_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 894 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3072 {System.Int32}; Good]
'[Int32Fast, 920 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:14 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1003 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, False {System.Boolean}; Good]
'[Int32, 3073 {System.Int32}; Good]
'[Int32Fast, 1074 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:15 PM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt16)30, group=101, fields: 4
'[BoolToggle, True {System.Boolean}; Good]
'[Int32, 3074 {System.Int32}; Good]
'[Int32Fast, 1140 {System.Int32}; Good]
'[DateTime, 10/1/2019 12:21:16 PM {System.DateTime}; Good]
'
'...
'
```

13.2.11.15 Examples - OPC UA PubSub - Subscribe to a single dataset field

C#

```
// This example shows how to subscribe to a single dataset field, resolving logical
parameters to physical from an OPC-UA
// PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode fixed layout
// messages with RawData field encoding.
//
// In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to
specify the interface name to be used.

using System;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    class SubscribeDataSetField
    {
        public static void Main1()
        {
            // Define the PubSub resolver. We want the information be resolved from a
PubSub binary configuration file that
            // we have. The file itself at the root of the project, and we have
specified that it has to be copied to the
            // project's output directory.
            var pubSubResolverDescriptor =
UAPubSubResolverDescriptor.File("UADemoPublisher-Default.uabinary");

            // Define the PubSub connection we will work with, using its logical name
in the PubSub configuration.
            var pubSubConnectionDescriptor = new UAPubSubConnectionDescriptor { Name =
"FixedLayoutConnection" };
            // In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. The writer group and the dataset writer are specified
using their logical names in the
            // PubSub configuration. The publisher Id in the filter will be taken from
the logical PubSub connection.
            var filter = new UASubscribeDataSetFilter("FixedLayoutGroup",
"SimpleWriter");

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            Console.WriteLine("Subscribing...");
            int fieldHandle = subscriber.SubscribeDataSetField(
```

```
                pubSubResolverDescriptor,
                pubSubConnectionDescriptor,
                filter,
                "Int32Fast",
                (sender, args) => { Console.WriteLine(args); });

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeDataSetField(fieldHandle);

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //Success
        //Success; 1626 {System.Int32}; Good
        //Success; 1711 {System.Int32}; Good
        //Success; 1741 {System.Int32}; Good
        //Success; 1837 {System.Int32}; Good
        //Success; 1897 {System.Int32}; Good
        //Success; 1993 {System.Int32}; Good
        //Success; 2082 {System.Int32}; Good
        //Success; 2135 {System.Int32}; Good
        //Success; 2185 {System.Int32}; Good
        //Success; 2241 {System.Int32}; Good
        //Success; 2324 {System.Int32}; Good
        //Success; 2368 {System.Int32}; Good
        //Success; 2423 {System.Int32}; Good
        //Success; 2445 {System.Int32}; Good
        //Success; 2497 {System.Int32}; Good
        //Success; 2584 {System.Int32}; Good
        //Success; 2608 {System.Int32}; Good
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to a single dataset field, resolving logical
parameters to physical from an OPC-UA
' PubSub configuration file in binary format. The metadata obtained through the
resolution is used to decode fixed layout
' messages with RawData field encoding.
'
' In order to produce network messages for this example, run the UADemoPublisher tool.
For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify
```

```
the interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class SubscribeDataSetField
        Public Shared Sub Main1()

            ' Define the PubSub resolver. We want the information be resolved from a
PubSub binary configuration file that
            ' we have. The file itself is at the root of the project, and we have
specified that it has to be copied to the
            ' project's output directory.
            Dim pubSubResolverDescriptor =
UAPubSubResolverDescriptor.File("UADemoPublisher-Default.uabinary")

            ' Define the PubSub connection we will work with, using its logical name in
the PubSub configuration.
            Dim pubSubConnectionDescriptor = New UAPubSubConnectionDescriptor
            pubSubConnectionDescriptor.Name = "FixedLayoutConnection"
            ' In some cases you may have to set the interface (network adapter) name
that needs to be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the filter. The writer group And the dataset writer are specified
using their logical names in the
            ' PubSub configuration. The publisher Id in the filter will be taken from
the logical PubSub connection.
            Dim filter = New UASubscribeDataSetFilter("FixedLayoutGroup",
"SimpleWriter")

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            Console.WriteLine("Subscribing...")
            Dim fieldHandle = subscriber.SubscribeDataSetField(
                pubSubResolverDescriptor,
                pubSubConnectionDescriptor,
                filter,
                "Int32Fast",
                Sub(sender, eventArgs)
                    Console.WriteLine(eventArgs)
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a
short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
```

```
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    'Success
    'Success; 1626 {System.Int32}; Good
    'Success; 1711 {System.Int32}; Good
    'Success; 1741 {System.Int32}; Good
    'Success; 1837 {System.Int32}; Good
    'Success; 1897 {System.Int32}; Good
    'Success; 1993 {System.Int32}; Good
    'Success; 2082 {System.Int32}; Good
    'Success; 2135 {System.Int32}; Good
    'Success; 2185 {System.Int32}; Good
    'Success; 2241 {System.Int32}; Good
    'Success; 2324 {System.Int32}; Good
    'Success; 2368 {System.Int32}; Good
    'Success; 2423 {System.Int32}; Good
    'Success; 2445 {System.Int32}; Good
    'Success; 2497 {System.Int32}; Good
    'Success; 2584 {System.Int32}; Good
    'Success; 2608 {System.Int32}; Good
    '...

End Namespace
```

13.2.11.16 Examples - OPC UA PubSub - Subscribe to all dataset messages on a connection

The example below subscribes to all dataset messages on an OPC-UA PubSub connection with UDP UADP mapping. The connection is specified by its physical parameters, using the scheme name "opc.udp" and the IP address of the multicast group to listen on.

C#

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
```

```
    {
        public static void Main1()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a
string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to
be used, similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in
e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
```

```
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //...
    }
}
```

C++

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include "SubscribeDataSet.h"

namespace PubSub
{
    namespace _EasyUASubscriber
    {
        // CEasyUASubscriberEvents

        class CEasyUASubscriberEvents : public IDispEventImpl<1, CEasyUASubscriberEvents>
        {
        public:
            BEGIN_SINK_MAP(CEasyUASubscriberEvents)
                // Event handlers must have the __stdcall calling convention
                SINK_ENTRY(1, DISPID_EASYUASUBSCRIBEREVENTS_DATASETMESSAGE,
&CEasyUASubscriberEvents::DataSetMessage)
            END_SINK_MAP()

        public:
            // The handler for EasyUAClient.DataChangeNotification event
            STDMETHOD(DataSetMessage)(VARIANT varSender, _EasyUADataSetMessageEventArgs*
pEventArgs)
            {
                // Display the dataset.
                if (pEventArgs->Succeeded)
                {
                    _UADataSetDataPtr DataSetDataPtr = pEventArgs->DataSetData;
                    // An event with null DataSetData just indicates a successful connection.
                    if (DataSetDataPtr != NULL)
                    {
                        _tprintf(_T("\n"));
                        _tprintf(_T("Dataset data: %s\n"), (LPCTSTR)CW2CT(DataSetDataPtr-
>ToString));

                        IEnumVARIANTPtr EnumDictionaryEntry2Ptr = DataSetDataPtr-
```

```
>FieldDataDictionary->GetEnumerator();
                        _variant_t vDictionaryEntry2;
                        while (EnumDictionaryEntry2Ptr->Next(1, &vDictionaryEntry2, NULL) == S_OK)
                        {
                            _DictionaryEntry2Ptr DictionaryEntry2Ptr(vDictionaryEntry2);
                            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(DictionaryEntry2Ptr->ToString));
                            vDictionaryEntry2.Clear();
                        }
                    }
                }
                else
                {
                    _tprintf(_T("\n"));
                    _tprintf(_T("*** Failure: %s\n"), (LPCTSTR)CW2CT(pEventArgs-
>ErrorMessageBrief));
                }
                return S_OK;
            }
        };



        void SubscribeDataSet::Main1()
        {
            // Initialize the COM library
            CoInitializeEx(NULL, COINIT_MULTITHREADED);
            {
                // Prepare arguments for a subscription to the dataset.
                _EasyUASubscribeDataSetArgumentsPtr
SubscribeDataSetArgumentsPtr(__uuidof(EasyUASubscribeDataSetArguments));

                // Define the PubSub connection we will work with.
                _UAPubSubConnectionDescriptorPtr PubSubConnectionDescriptorPtr =
                    SubscribeDataSetArgumentsPtr->DataSetSubscriptionDescriptor-
>ConnectionDescriptor;
                PubSubConnectionDescriptorPtr->ResourceAddress->ResourceDescriptor->UrlString =
L"opc.udp://239.0.0.1";
                // In some cases you may have to set the interface (network adapter) name that
needs to be used, similarly to
                // the statement below. Your actual interface name may differ, of course.
                //PubSubConnectionDescriptorPtr->ResourceAddress->InterfaceName = L"Ethernet";

                // Instantiate the subscriber object.
                _EasyUASubscriberPtr SubscriberPtr(__uuidof(EasyUASubscriber));

                // Hook events.
                CEasyUASubscriberEvents* pSubscriberEvents = new CEasyUASubscriberEvents();
                AtlGetObjectSourceInterface(SubscriberPtr, &pSubscriberEvents->m_libid,
&pSubscriberEvents->m_iid,
                    &pSubscriberEvents->m_wMajorVerNum, &pSubscriberEvents->m_wMinorVerNum);
                pSubscriberEvents->m_iid = _uuidof(DEasyUASubscriberEvents);
                pSubscriberEvents->DispEventAdvise(SubscriberPtr, &pSubscriberEvents->m_iid);

                _tprintf(_T("Subscribing...\n"));
                _variant_t
vSubscribeDataSetArguments(SubscribeDataSetArgumentsPtr.GetInterfacePtr());
                SubscriberPtr->SubscribeDataSet(vSubscribeDataSetArguments);

                _tprintf(_T("Processing dataset message events for 20 seconds...\n"));
                Sleep(20 * 1000);

                _tprintf(_T("Unsubscribing...\n"));
                SubscriberPtr->UnsubscribeAllDataSets();

                _tprintf(_T("Waiting for 1 second...\n"));
```

```
                // Unsubscribe operation is asynchronous, messages may still come for a short
while.
                Sleep(1 * 1000);

                // Unhook events
                pSubscriberEvents->DispEventUnadvise(SubscriberPtr, &pSubscriberEvents->m_iid);

                _tprintf(_T("Finished.\n"));
            }
            // Release all interface pointers BEFORE calling CoUninitialize()
            CoUninitialize();
        }



        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        ////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
        //[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
        //
        //Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
        //[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
        //...
    }
}
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

type
  TSubscriberEventHandlers77 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.Main1;
```

```
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers77;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor :=
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers77.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers77.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
```

```
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908,
fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae,
fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

PHP

```
// This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

class SubscriberEvents {
    function DataSetMessage($Sender, $E)
    {
        // Display the dataset.
        if ($E->Succeeded) {
            // An event with null DataSetData just indicates a successful connection.
            printf("\n");
            printf("Dataset data: %s\n", $E->DataSetData);
            foreach ($E->DataSetData->FieldDataDictionary as $Pair)
                printf("%s\n", $Pair);
        }
        else {
            printf("\n");
            printf("*** Failure: %s\n", $E->ErrorMessageBrief);
        }
    }
}
```

```
// Define the PubSub connection we will work with.
$SubscribeDataSetArguments = new
COM("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments");
$ConnectionDescriptor = $SubscribeDataSetArguments->DataSetSubscriptionDescriptor-
>ConnectionDescriptor;
$ConnectionDescriptor->ResourceAddress->ResourceDescriptor->UrlString = "opc.udp://239.0.0.1";
// In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
// the statement below. Your actual interface name may differ, of course.
/$ConnectionDescriptor->ResourceAddress->InterfaceName = "Ethernet";

// Instantiate the subscriber object and hook events.
$Subscriber = new COM("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber");
$SubscriberEvents = new SubscriberEvents();
com_event_sink($Subscriber, $SubscriberEvents, "DEasyUASubscriberEvents");

printf("Subscribing...\n");
$Subscriber->SubscribeDataSet($SubscribeDataSetArguments);

printf("Processing dataset message events for 20 seconds...");
$startTime = time();
do {
    com_message_pump(1000);
} while (time() < $startTime + 20);

printf("Unsubscribing...\n");
$Subscriber->UnsubscribeAllDataSets;

printf("Waiting for 1 second...");
// Unsubscribe operation is asynchronous, messages may still come for a short while.
$startTime = time();
do {
    com_message_pump(1000);
} while (time() < $startTime + 1);

printf("Finished.\n");



// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
////Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908,
fields: 4
//[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
//
//Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae,
fields: 100
//[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
```

```
//[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
//...
```

Python

```
# This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection, and
pull events, and display
# the incoming datasets.
#
# In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
# http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

import time
import win32com.client

# Define the PubSub connection we will work with.
subscribeDataSetArguments =
win32com.client.Dispatch('OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments')

connectionDescriptor = subscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
connectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.udp://239.0.0.1'
# In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
# the statement below. Your actual interface name may differ, of course.
#connectionDescriptor.ResourceAddress.InterfaceName = 'Ethernet'

# Instantiate the subscriber object.
subscriber = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber')
# In order to use event pull, you must set a non-zero queue capacity upfront.
subscriber.PullDataSetMessageQueueCapacity = 1000

print('Subscribing...')
subscriber.SubscribeDataSet(subscribeDataSetArguments)

print('Processing dataset message events for 20 seconds...')
endTime = time.time() + 20
while time.time() < endTime:
    eventArgs = subscriber.PullDataSetMessage(2*1000)
    if eventArgs is not None:
        # Display the dataset.
        if eventArgs.Succeeded:
            # An event with null DataSetData just indicates a successful connection.
            if eventArgs.DataSetdata is not None:
                print('')
                print('Dataset data: ', eventArgs.DataSetData)
                for pair in eventArgs.DataSetData.FieldDataDictionary:
                    print(pair)
        else:
            print('')
            print('*** Failure: ', eventArgs.ErrorMessageBrief)

print('Unsubscribing...')
subscriber.UnsubscribeAllDataSets

print('Waiting for 1 second...')
# Unsubscribe operation is asynchronous, messages may still come for a short while.
endTime = time.time() + 1
while time.time() < endTime:
    pass

print('Finished.')
```

```
# Example output:
#
#Subscribing...
#Processing dataset message events for 20 seconds...
#
##Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908,
fields: 4
#[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
#
#Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae,
fields: 100
#[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
#...
```

VB.NET

```
' This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class SubscribeDataSet
        Public Shared Sub Main1()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a
string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to
be used, similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor)
```

```
            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    ''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-
4096cd1d8908, fields: 4
    '[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-
01T00:00:00.000; Good]
    '
    'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-
0979884b55ae, fields: 100
    '[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
    '...

End Namespace
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

' The subscriber object, with events.
'Public WithEvents Subscriber1 As EasyUASubscriber

Private Sub SubscribeDataSet_Main1_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments = New
EasyUASubscribeDataSetArguments
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
    ' the statement below. Your actual interface name may differ, of course.
    'ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Instantiate the subscriber object.
    Set Subscriber1 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber1.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber1.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber1 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber1_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As
EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
```

```
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908,
fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae,
fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
'
```

VBScript

```
Rem This example shows how to subscribe to all dataset messages on an OPC-UA PubSub connection with
UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For
documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the
interface name to be used.

Option Explicit

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000
```

```
WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
''Dataset data: Good; Data; publisher="32", writer=1, class=eae79794-1af7-4f96-8401-4096cd1d8908,
fields: 4
'[#0, True {System.Boolean} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 7945 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 5246 {System.Int32} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 9/30/2019 11:19:14 AM {System.DateTime} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000;
Good]
'
'Dataset data: Good; Data; publisher="32", writer=3, class=96976b7b-0db7-46c3-a715-0979884b55ae,
fields: 100
'[#0, 45 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#1, 145 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#2, 245 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#3, 345 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#4, 445 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#5, 545 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#6, 645 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#7, 745 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#8, 845 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#9, 945 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'[#10, 1045 {System.Int64} @0001-01-01T00:00:00.000 @@0001-01-01T00:00:00.000; Good]
'...
```

13.2.11.17 Examples - OPC UA PubSub - Subscribe to dataset messages from specific publisher

C#

```
// This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
// with UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void PublisherId()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit)
publisher Id 31.
            var subscribeDataSetArguments = new UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31));

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage_PublisherId;

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage_PublisherId(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
```

```
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
        //[#0, True {System.Boolean}; Good]
        //[#1, 1237 {System.Int32}; Good]
        //[#2, 2514 {System.Int32}; Good]
        //[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#0, False {System.Boolean}; Good]
        //[#1, 1239 {System.Int32}; Good]
        //[#2, 2703 {System.Int32}; Good]
        //[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
        //[#0, False {System.Boolean}; Good]
        //[#1, 215 {System.Byte}; Good]
        //[#2, 1239 {System.Int16}; Good]
        //[#3, 1239 {System.Int32}; Good]
        //[#4, 1239 {System.Int64}; Good]
        //[#5, 87 {System.Int16}; Good]
        //[#6, 1239 {System.Int32}; Good]
        //[#7, 1239 {System.Int64}; Good]
        //[#8, 1239 {System.Decimal}; Good]
        //[#9, 1239 {System.Single}; Good]
        //[#10, 1239 {System.Double}; Good]
        //[#11, Romeo {System.String}; Good]
        //[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
        //[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
        //[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
        //[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#2, 2722 {System.Int32}; Good]
        //[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
        //[#0, False {System.Boolean}; Good]
        //[#1, 1239 {System.Int32}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
        //[#0, 39 {System.Int64}; Good]
        //[#1, 139 {System.Int64}; Good]
        //[#2, 239 {System.Int64}; Good]
        //[#3, 339 {System.Int64}; Good]
        //[#4, 439 {System.Int64}; Good]
        //[#5, 539 {System.Int64}; Good]
        //[#6, 639 {System.Int64}; Good]
        //[#7, 739 {System.Int64}; Good]
        //[#8, 839 {System.Int64}; Good]
        //[#9, 939 {System.Int64}; Good]
        //[#10, 1039 {System.Int64}; Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
' with UDP UADP mapping.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.
```

```
Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub PublisherId()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

            ' Define the arguments for subscribing to the dataset, where the filter Is (unsigned 64-bit)
publisher Id 31.
            Dim subscribeDataSetArguments = New UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31))

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage_PublisherId

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(subscribeDataSetArguments)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage_PublisherId(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
    '[#0, True {System.Boolean}; Good]
    '[#1, 1237 {System.Int32}; Good]
    '[#2, 2514 {System.Int32}; Good]
    '[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
    '
```

```
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, False {System.Boolean}; Good]
    '[#1, 1239 {System.Int32}; Good]
    '[#2, 2703 {System.Int32}; Good]
    '[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
    '[#0, False {System.Boolean}; Good]
    '[#1, 215 {System.Byte}; Good]
    '[#2, 1239 {System.Int16}; Good]
    '[#3, 1239 {System.Int32}; Good]
    '[#4, 1239 {System.Int64}; Good]
    '[#5, 87 {System.Int16}; Good]
    '[#6, 1239 {System.Int32}; Good]
    '[#7, 1239 {System.Int64}; Good]
    '[#8, 1239 {System.Decimal}; Good]
    '[#9, 1239 {System.Single}; Good]
    '[#10, 1239 {System.Double}; Good]
    '[#11, Romeo {System.String}; Good]
    '[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
    '[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
    '[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
    '[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 2722 {System.Int32}; Good]
    '[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
    '[#0, False {System.Boolean}; Good]
    '[#1, 1239 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
    '[#0, 39 {System.Int64}; Good]
    '[#1, 139 {System.Int64}; Good]
    '[#2, 239 {System.Int64}; Good]
    '[#3, 339 {System.Int64}; Good]
    '[#4, 439 {System.Int64}; Good]
    '[#5, 539 {System.Int64}; Good]
    '[#6, 639 {System.Int64}; Good]
    '[#7, 739 {System.Int64}; Good]
    '[#8, 839 {System.Int64}; Good]
    '[#9, 939 {System.Int64}; Good]
    '[#10, 1039 {System.Int64}; Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
// with UDP UADP mapping.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers80 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.PublisherId;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers80;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
```

```
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers80.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers80.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
//[#0, True {System.Boolean}; Good]
```

```
//[#1, 1237 {System.Int32}; Good]
//[#2, 2514 {System.Int32}; Good]
//[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, False {System.Boolean}; Good]
//[#1, 1239 {System.Int32}; Good]
//[#2, 2703 {System.Int32}; Good]
//[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
//[#0, False {System.Boolean}; Good]
//[#1, 215 {System.Byte}; Good]
//[#2, 1239 {System.Int16}; Good]
//[#3, 1239 {System.Int32}; Good]
//[#4, 1239 {System.Int64}; Good]
//[#5, 87 {System.Int16}; Good]
//[#6, 1239 {System.Int32}; Good]
//[#7, 1239 {System.Int64}; Good]
//[#8, 1239 {System.Decimal}; Good]
//[#9, 1239 {System.Single}; Good]
//[#10, 1239 {System.Double}; Good]
//[#11, Romeo {System.String}; Good]
//[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
//[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
//[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 2722 {System.Int32}; Good]
//[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
//[#0, False {System.Boolean}; Good]
//[#1, 1239 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
//[#0, 39 {System.Int64}; Good]
//[#1, 139 {System.Int64}; Good]
//[#2, 239 {System.Int64}; Good]
//[#3, 339 {System.Int64}; Good]
//[#4, 439 {System.Int64}; Good]
//[#5, 539 {System.Int64}; Good]
//[#6, 639 {System.Int64}; Good]
//[#7, 739 {System.Int64}; Good]
//[#8, 839 {System.Int64}; Good]
//[#9, 939 {System.Int64}; Good]
//[#10, 1039 {System.Int64}; Good]
//...
```

VBScript

```
Rem This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
Rem with UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.
```

```
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 1237 {System.Int32}; Good]
'[#2, 2514 {System.Int32}; Good]
'[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'[#2, 2703 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[#0, False {System.Boolean}; Good]
'[#1, 215 {System.Byte}; Good]
'[#2, 1239 {System.Int16}; Good]
'[#3, 1239 {System.Int32}; Good]
'[#4, 1239 {System.Int64}; Good]
'[#5, 87 {System.Int16}; Good]
'[#6, 1239 {System.Int32}; Good]
'[#7, 1239 {System.Int64}; Good]
'[#8, 1239 {System.Decimal}; Good]
'[#9, 1239 {System.Single}; Good]
'[#10, 1239 {System.Double}; Good]
'[#11, Romeo {System.String}; Good]
```

```
'[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
'[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
'[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 2722 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 39 {System.Int64}; Good]
'[#1, 139 {System.Int64}; Good]
'[#2, 239 {System.Int64}; Good]
'[#3, 339 {System.Int64}; Good]
'[#4, 439 {System.Int64}; Good]
'[#5, 539 {System.Int64}; Good]
'[#6, 639 {System.Int64}; Good]
'[#7, 739 {System.Int64}; Good]
'[#8, 839 {System.Int64}; Good]
'[#9, 939 {System.Int64}; Good]
'[#10, 1039 {System.Int64}; Good]
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to all dataset messages with specific publisher Id, on an OPC-UA PubSub
connection
Rem with UDP UADP mapping.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber9 As EasyUASubscriber

Private Sub SubscribeDataSet_PublisherId_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31

    ' Instantiate the subscriber object and hook events.
    Set Subscriber9 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber9.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Subscriber9.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber9 = Nothing
```

```
    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber9_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Event; publisher=(UInt64)31, writer=51, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 1237 {System.Int32}; Good]
'[#2, 2514 {System.Int32}; Good]
'[#3, 10/1/2019 9:03:59 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'[#2, 2703 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=4, fields: 16
'[#0, False {System.Boolean}; Good]
'[#1, 215 {System.Byte}; Good]
'[#2, 1239 {System.Int16}; Good]
'[#3, 1239 {System.Int32}; Good]
'[#4, 1239 {System.Int64}; Good]
'[#5, 87 {System.Int16}; Good]
'[#6, 1239 {System.Int32}; Good]
'[#7, 1239 {System.Int64}; Good]
'[#8, 1239 {System.Decimal}; Good]
'[#9, 1239 {System.Single}; Good]
'[#10, 1239 {System.Double}; Good]
'[#11, Romeo {System.String}; Good]
'[#12, [20] {175, 186, 248, 246, 215, ...} {System.Byte[]}; Good]
'[#13, d4492ca8-35c8-4b98-8edf-6ffa5ca041ca {System.Guid}; Good]
'[#14, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#15, [10] {1239, 1240, 1241, 1242, 1243, ...} {System.Int64[]}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 2722 {System.Int32}; Good]
'[#3, 10/1/2019 9:04:01 AM {System.DateTime}; Good]
'[#0, False {System.Boolean}; Good]
'[#1, 1239 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 39 {System.Int64}; Good]
'[#1, 139 {System.Int64}; Good]
'[#2, 239 {System.Int64}; Good]
'[#3, 339 {System.Int64}; Good]
'[#4, 439 {System.Int64}; Good]
'[#5, 539 {System.Int64}; Good]
'[#6, 639 {System.Int64}; Good]
'[#7, 739 {System.Int64}; Good]
'[#8, 839 {System.Int64}; Good]
'[#9, 939 {System.Int64}; Good]
```

```
'[#10, 1039 {System.Int64}; Good]
'...
'
```

13.2.11.18 Examples - OPC UA PubSub - Unsubscribe from a dataset

C#

```
// This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
// dataset.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    class UnsubscribeDataSet
    {
        public static void Main1()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            // In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            // the statement below. Your actual interface name may differ, of course.
            //pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet";

            // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
            var filter = new UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1);

            // Instantiate the subscriber object and hook events.
            var subscriber = new EasyUASubscriber();
            subscriber.DataSetMessage += subscriber_DataSetMessage;

            Console.WriteLine("Subscribing...");
            int dataSetHandle = subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter);

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeDataSet(dataSetHandle);

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        static void subscriber_DataSetMessage(object sender, EasyUADataSetMessageEventArgs e)
        {
            // Display the dataset.
            if (e.Succeeded)
            {
                // An event with null DataSetData just indicates a successful connection.
                if (!(e.DataSetData is null))
                {
                    Console.WriteLine();
                    Console.WriteLine($"Dataset data: {e.DataSetData}");
```

```
                    foreach (KeyValuePair<string, UADataSetFieldData> pair in e.DataSetData.FieldDataDictionary)
                        Console.WriteLine(pair);
                }
            }
            else
            {
                Console.WriteLine();
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}");
            }
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#0, True {System.Boolean}; Good]
        //[#1, 7134 {System.Int32}; Good]
        //[#2, 7364 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#1, 7135 {System.Int32}; Good]
        //[#2, 7429 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#2, 7495 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //[#1, 7135 {System.Int32}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#1, 7136 {System.Int32}; Good]
        //[#2, 7560 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //
        //Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#2, 7626 {System.Int32}; Good]
        //[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
        //[#0, True {System.Boolean}; Good]
        //[#1, 7136 {System.Int32}; Good]
        //
        //...
    }
}
```

VB.NET

```
' This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
' dataset.
'
' In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
' http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Friend Class UnsubscribeDataSet
        Public Shared Sub Main1()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            ' In some cases you may have to set the interface (network adapter) name that needs to be used,
similarly to
            ' the statement below. Your actual interface name may differ, of course.
            ' pubSubConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"
```

```
            ' Define the filter. Publisher Id (unsigned 64-bits) is 31, And the dataset writer Id Is 1.
            Dim filter = New UASubscribeDataSetFilter(UAPublisherId.CreateUInt64(31),
UAWriterGroupDescriptor.Null, 1)

            ' Instantiate the subscriber object and hook events.
            Dim subscriber = New EasyUASubscriber()
            AddHandler subscriber.DataSetMessage, AddressOf subscriber_DataSetMessage

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(pubSubConnectionDescriptor, filter)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub

        Private Shared Sub subscriber_DataSetMessage(ByVal sender As Object, ByVal e As
EasyUADataSetMessageEventArgs)
            ' Display the dataset.
            If e.Succeeded Then
                ' An event with null DataSetData just indicates a successful connection.
                If e.DataSetData IsNot Nothing Then
                    Console.WriteLine()
                    Console.WriteLine($"Dataset data: {e.DataSetData}")
                    For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
e.DataSetData.FieldDataDictionary
                        Console.WriteLine(pair)
                    Next
                End If
            Else
                Console.WriteLine()
                Console.WriteLine($"*** Failure: {e.ErrorMessageBrief}")
            End If
        End Sub
    End Class



    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, True {System.Boolean}; Good]
    '[#1, 7134 {System.Int32}; Good]
    '[#2, 7364 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7135 {System.Int32}; Good]
    '[#2, 7429 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7495 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7135 {System.Int32}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#1, 7136 {System.Int32}; Good]
    '[#2, 7560 {System.Int32}; Good]
```

```
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '
    'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#2, 7626 {System.Int32}; Good]
    '[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
    '[#0, True {System.Boolean}; Good]
    '[#1, 7136 {System.Int32}; Good]
    '
    '...

End Namespace
```

Object Pascal

```
// This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
// dataset.
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.

type
  TSubscriberEventHandlers82 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure UnsubscribeDataSet.Main1;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers82;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.udp://239.0.0.1';
  // In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
  // the statement below. Your actual interface name may differ, of course.
  //ConnectionDescriptor.ResourceAddress.InterfaceName := 'Ethernet';

  // Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);
  SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId := 1;

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers82.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
```

```
end;

procedure TSubscriberEventHandlers82.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, True {System.Boolean}; Good]
//[#1, 7134 {System.Int32}; Good]
//[#2, 7364 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#1, 7135 {System.Int32}; Good]
//[#2, 7429 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 7495 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//[#1, 7135 {System.Int32}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#1, 7136 {System.Int32}; Good]
//[#2, 7560 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//
//Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#2, 7626 {System.Int32}; Good]
//[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
//[#0, True {System.Boolean}; Good]
//[#1, 7136 {System.Int32}; Good]
//
//...
```

VBScript

```
Rem This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
Rem dataset.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
' the statement below. Your actual interface name may differ, of course.
' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Dim dataSetHandle: dataSetHandle = Subscriber.SubscribeDataSet(SubscribeDataSetArguments)

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeDataSet dataSetHandle

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub



' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
```

```
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to dataset messages on an OPC-UA PubSub connection, and then unsubscribe
from that
Rem dataset.
Rem
Rem In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
Rem http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to
be used.

' The subscriber object, with events.
'Public WithEvents Subscriber11 As EasyUASubscriber

Private Sub UnsubscribeDataSet_Main1_Command_Click()
    OutputText = ""

    ' Define the PubSub connection we will work with.
    Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
    Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
    ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
    ' In some cases you may have to set the interface (network adapter) name that needs to be used, similarly to
    ' the statement below. Your actual interface name may differ, of course.
    ' ConnectionDescriptor.ResourceAddress.InterfaceName = "Ethernet"

    ' Define the filter. Publisher Id (unsigned 64-bits) is 31, and the dataset writer Id is 1.
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31
    SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.DataSetWriterDescriptor.DataSetWriterId = 1

    ' Instantiate the subscriber object and hook events.
    Set Subscriber11 = New EasyUASubscriber

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Subscriber11.SubscribeDataSet SubscribeDataSetArguments

    OutputText = OutputText & "Processing dataset message events for 20 seconds..." & vbCrLf
    Pause 20 * 1000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
```

```
    Subscriber11.UnsubscribeAllDataSets

    OutputText = OutputText & "Waiting for 1 second..." & vbCrLf
    ' Unsubscribe operation is asynchronous, messages may still come for a short while.
    Pause 1000

    Set Subscriber11 = Nothing

    OutputText = OutputText & "Finished." & vbCrLf
End Sub

Private Sub Subscriber11_DataSetMessage(ByVal sender As Variant, ByVal EventArgs As
EasyUADataSetMessageEventArgs)
    ' Display the dataset.
    If EventArgs.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (EventArgs.DataSetData Is Nothing) Then
            OutputText = OutputText & vbCrLf
            OutputText = OutputText & "Dataset data: " & EventArgs.DataSetData & vbCrLf
            Dim Pair: For Each Pair In EventArgs.DataSetData.FieldDataDictionary
                OutputText = OutputText & Pair & vbCrLf
            Next
        End If
    Else
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "*** Failure: " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
'
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean}; Good]
'[#1, 7134 {System.Int32}; Good]
'[#2, 7364 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:16 AM {System.DateTime}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7135 {System.Int32}; Good]
'[#2, 7429 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7495 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:17 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7135 {System.Int32}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#1, 7136 {System.Int32}; Good]
'[#2, 7560 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'
'Dataset data: Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#2, 7626 {System.Int32}; Good]
'[#3, 10/1/2019 10:42:18 AM {System.DateTime}; Good]
'[#0, True {System.Boolean}; Good]
'[#1, 7136 {System.Int32}; Good]
'
'...
'
```

13.2.11.19 Examples - OPC UA PubSub - Use Packet capture file

C#

```
// This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to
the message
// oriented middleware (receiving the messages from the network).
//
// The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made available, together with
its
// dependencies) for the capture files to work. Refer to the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Extensions;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void CaptureFile()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF";
            // Use packets from the specified Ethernet capture file. The file itself is at the root of the
project, and we
            // have specified that it has to be copied to the project's output directory.
            // Note that .pcap is the default file name extension, and can thus be omitted.
            pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-Ethernet.pcap");

            // Alternative setup for Ethernet with VLAN tagging:
            //UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF:2";
            //pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-EthernetVlan.pcap");

            // Alternative setup for UDP over IPv4:
            //UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            //pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP.pcap");

            // Alternative setup for UDP over IPv6:
            //UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://[ff02::1]";
            //pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP6.pcap");

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit)
publisher Id 31.
            var subscribeDataSetArguments = new UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31));

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments, (sender, args) =>
            {
                // Display the dataset.
                if (args.Succeeded)
                {
                    // An event with null DataSetData just indicates a successful connection.
                    if (!(args.DataSetData is null))
                    {
                        Console.WriteLine();
                        Console.WriteLine($"Dataset data: {args.DataSetData}");
                        foreach (KeyValuePair<string, UADataSetFieldData> pair in
args.DataSetData.FieldDataDictionary)
                            Console.WriteLine(pair);
                    }
                }
                else
                {
                    Console.WriteLine();
                    Console.WriteLine($"*** Failure: {args.ErrorMessageBrief}");
                }
            });
```

```
            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#0, True {System.Boolean} @2019-10-31T16:04:59.145,266,700,00; Good]
        //[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
        //[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
        //[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
        //
        //Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields:
100
        //[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to the
message
' oriented middleware (receiving the messages from the network).
'
' The OpcLabs.Pcap assembly needs to be referenced in your project (Or otherwise made available, together with
its
' dependencies) for the capture files to work. Refer to the documentation for more information.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub CaptureFile()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF"
            ' Use packets from the specified Ethernet capture file. The file itself is at the root of the
project, and we
            ' have specified that it has to be copied to the project's output directory.
            ' Note that .pcap is the default file name extension, and can thus be omitted.
            pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-Ethernet.pcap")

            ' Alternative setup for Ethernet with VLAN tagging
            'Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF:2"
            'pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-EthernetVlan.pcap")

            ' Alternative setup for UDP over IPv4
```

```
            'Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            'pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP.pcap")

            ' Alternative setup for UDP over IPv6
            'Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://[ff02::1]"
            'pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP6.pcap")

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            ' Define the arguments for subscribing to the dataset, where the filter Is (unsigned 64-bit)
publisher Id 31.
            Dim subscribeDataSetArguments = New UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31))

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(subscribeDataSetArguments,
                Sub(sender, eventArgs)
                    '  Display the dataset.
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}", eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief)
                    End If
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, True {System.Boolean} @2019-10-31T1604:59.145,266,700,00; Good]
    '[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
    '[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
    '[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
    '
    'Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields: 100
    '[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '...
```

```
End Namespace
```

Object Pascal

```
// This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to
the message
// oriented middleware (receiving the messages from the network).
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.
//
// The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made available, together with
its
// dependencies) for the capture files to work. Refer to the documentation for more information.

type
  TSubscriberEventHandlers72 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.CaptureFile;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers72;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.eth://FF-FF-FF-FF-FF-FF';
  // Use packets from the specified Ethernet capture file. The file itself is at the root of the project, and we
  // have specified that it has to be copied to the project's output directory.
  // Note that .pcap is the default file name extension, and can thus be omitted.
  ConnectionDescriptor.UseEthernetCaptureFile('UADemoPublisher-Ethernet.pcap');

  // Alternative setup for Ethernet with VLAN tagging:
  //ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.eth://FF-FF-FF-FF-FF-FF:2';
  //ConnectionDescriptor.UseEthernetCaptureFile('UADemoPublisher-EthernetVlan.pcap');

  // Alternative setup for UDP over IPv4:
  //ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.udp://239.0.0.1';
  //ConnectionDescriptor.UseUdpCaptureFile('UADemoPublisher-UDP.pcap');

  // Alternative setup for UDP over IPv6:
  //ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.udp://[ff02::1]';
  //ConnectionDescriptor.UseUdpCaptureFile('UADemoPublisher-UDP6.pcap');

  // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers72.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
```

```
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers72.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, True {System.Boolean} @2019-10-31T16:04:59.145,266,700,00; Good]
//[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
//[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
//[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
//
//Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields: 100
//[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//...
```

VBScript

```
Rem This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to
the message
Rem oriented middleware (receiving the messages from the network).
Rem
Rem The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made available, together with
its
```

```
Rem dependencies) for the capture files to work. Refer to the documentation for more information.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.eth://FF-FF-FF-FF-FF-FF"
' Use packets from the specified Ethernet capture file. The file itself is at the root of the project, and we
' have specified that it has to be copied to the project's output directory.
' Note that .pcap is the default file name extension, and can thus be omitted.
ConnectionDescriptor.UseEthernetCaptureFile "UADemoPublisher-Ethernet.pcap"

' Alternative setup for Ethernet with VLAN tagging:
'ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.eth://FF-FF-FF-FF-FF-FF:2"
'ConnectionDescriptor.UseEthernetCaptureFile "UADemoPublisher-EthernetVlan.pcap"

' Alternative setup for UDP over IPv4:
'ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
'ConnectionDescriptor.UseUdpCaptureFile "UADemoPublisher-UDP.pcap"

' Alternative setup for UDP over IPv6:
'ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://[ff02::1]"
'ConnectionDescriptor.UseUdpCaptureFile "UADemoPublisher-UDP6.pcap"

' Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
    End If
End Sub
```

```
' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean} @2019-10-31T16:04:59.145,266,700,00; Good]
'[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
'[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
'[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
'
'Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'...
```

13.2.12 Examples - OPC UA Specialized

13.2.12.1 Examples - OPC UA Specialized - Softing OPC UA .NET Demo Server - Repeatedly read many

Object Pascal

```
// This example shows how to repeatedly read 300 variables from Softing OPC UA .NET
Demo Server.

class procedure Softing_OpcUaNetDemoServer.RepeatedReadMany;
const
  EndpointDescriptorUrlString = 'opc.tcp://TEST-OPC-3:51510/UA/DemoServer';
  NamespaceUriString =
'nsu=http://opcfoundation.org/Quickstarts/ReferenceApplications';
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  FailureCount: Cardinal;
  SuccessCount: Cardinal;
  I, J: Cardinal;
  ReadArguments1, ReadArguments2, ReadArguments3, ReadArguments4: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  // Instantiate the client object
  Client := TEasyUAClient.Create(nil);

  //
  Arguments := VarArrayCreate([0, 299], varVariant);
```

```
  for I := 0 to 99 do
  begin
    ReadArguments1 := CoUAReadArguments.Create;
    ReadArguments1.EndpointDescriptor.UrlString := EndpointDescriptorUrlString;
    ReadArguments1.NodeDescriptor.NodeId.ExpandedText := NamespaceUriString +
      Format(' ;s=/Dynamic/All Profiles/Scalar Mass/Boolean/Boolean%.3d', [I]);

    ReadArguments2 := CoUAReadArguments.Create;
    ReadArguments2.EndpointDescriptor.UrlString := EndpointDescriptorUrlString;
    ReadArguments2.NodeDescriptor.NodeId.ExpandedText := NamespaceUriString +
      Format(' ;s=/Dynamic/All Profiles/Scalar Mass/Int16/Int16%.3d', [I]);

    ReadArguments3 := CoUAReadArguments.Create;
    ReadArguments3.EndpointDescriptor.UrlString := EndpointDescriptorUrlString;
    ReadArguments3.NodeDescriptor.NodeId.ExpandedText := NamespaceUriString +
      Format(' ;s=/Dynamic/All Profiles/Scalar Mass/Float/Float%.3d', [I]);

    Arguments[I] := ReadArguments1;
    Arguments[100 + I] := ReadArguments2;
    Arguments[200 + I] := ReadArguments3;
  end;

  WriteLn;
  WriteLn('Read the variables and wait 1 second, 60 times...');
  for J := 1 to 60 do
  begin
    TVarData(Results).VType := varArray or varVariant;
    TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

    SuccessCount := 0;
    FailureCount := 0;
    for I := 0 to 299 do
       if Results[I].Succeeded then
         SuccessCount := SuccessCount + 1
       else
         FailureCount := FailureCount + 1;
    WriteLn(Format('Success count: %d, failure count: %d', [SuccessCount,
FailureCount]));

    PumpSleep(1000);
  end;

  WriteLn;
  WriteLn('Finished.');
  FreeAndNil(Client);
end;
```

13.2.12.2 Examples - OPC UA Specialized - Softing OPC UA .NET Demo Server - Subscribe to many

Object Pascal

```
// This example shows how to subscribe to 300 monitored items from Softing OPC UA .NET
Demo Server.
```

```
type
  TClientEventHandlers1 = class
    FailureCount: Cardinal;
    SuccessCount: Cardinal;
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers1.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Process the data.
  if eventArgs.Succeeded then
    SuccessCount := SuccessCount + 1
  else
    FailureCount := FailureCount + 1;
end;

class procedure Softing_OpcUaNetDemoServer.SubscribeMany;
const
  EndpointDescriptorUrlString = 'opc.tcp://TEST-OPC-3:51510/UA/DemoServer';
  NamespaceUriString =
'nsu=http://opcfoundation.org/Quickstarts/ReferenceApplications';
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers1;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
_EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers1.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  //
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  Arguments := VarArrayCreate([0, 299], varVariant);
  for I := 0 to 99 do
  begin
    MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
    MonitoredItemArguments1.EndpointDescriptor.UrlString :=
EndpointDescriptorUrlString;
    MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText := NamespaceUriString +
      Format(' ;s=/Dynamic/All Profiles/Scalar Mass/Boolean/Boolean%.3d', [I]);
    MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;

    MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
```

```
    MonitoredItemArguments2.EndpointDescriptor.UrlString :=
EndpointDescriptorUrlString;
    MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText := NamespaceUriString +
      Format(' ;s=/Dynamic/All Profiles/Scalar Mass/Int16/Int16%.3d', [I]);
    MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;

    MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
    MonitoredItemArguments3.EndpointDescriptor.UrlString :=
EndpointDescriptorUrlString;
    MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText := NamespaceUriString +
      Format(' ;s=/Dynamic/All Profiles/Scalar Mass/Float/Float%.3d', [I]);
    MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;

    Arguments[I] := MonitoredItemArguments1;
    Arguments[100 + I] := MonitoredItemArguments2;
    Arguments[200 + I] := MonitoredItemArguments3;
  end;

  WriteLn('Subscribing...');
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray :=
PVarArray(Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Processing monitored item changed events for 60 seconds...');
  PumpSleep(60*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn;
  WriteLn(Format('Success count: %d', [ClientEventHandlers.SuccessCount]));
  WriteLn(Format('Failure count: %d', [ClientEventHandlers.FailureCount]));
  WriteLn(Format('Total count: %d', [ClientEventHandlers.SuccessCount +
ClientEventHandlers.FailureCount]));

  WriteLn;
  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

13.2.13 Examples - OPC Unified Architecture

13.2.13.1 Examples - OPC Unified Architecture - Browsing for all kinds of nodes

C#

```
// Shows how to obtain references of all kinds to nodes of all classes, from the
```

```
"Server" node in the address space.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class Browse
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain nodes under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.Browse(
                    endpointDescriptor,
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerArray
```

```
        //nodeElement.NodeId: Server_ServerArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
        //
        //nodeElement.DisplayName: NamespaceArray
        //nodeElement.NodeId: Server_NamespaceArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
        //
        //nodeElement.DisplayName: ServerStatus
        //nodeElement.NodeId: Server_ServerStatus
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
        //
        //nodeElement.DisplayName: ServiceLevel
        //nodeElement.NodeId: Server_ServiceLevel
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
        //
        //nodeElement.DisplayName: Auditing
        //nodeElement.NodeId: Server_Auditing
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
        //
        //nodeElement.DisplayName: ServerCapabilities
        //nodeElement.NodeId: Server_ServerCapabilities
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
        //
        //nodeElement.DisplayName: ServerDiagnostics
        //nodeElement.NodeId: Server_ServerDiagnostics
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
        //
        //nodeElement.DisplayName: VendorServerInfo
        //nodeElement.NodeId: Server_VendorServerInfo
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
        //
        //nodeElement.DisplayName: ServerRedundancy
        //nodeElement.NodeId: Server_ServerRedundancy
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
        //
        //nodeElement.DisplayName: Namespaces
        //nodeElement.NodeId: Server_Namespaces
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
        //
        //nodeElement.DisplayName: GetMonitoredItems
        //nodeElement.NodeId: Server_GetMonitoredItems
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11492
        //
        //nodeElement.DisplayName: Data
        //nodeElement.NodeId: nsu=http://test.org/UA/Data/ ;ns=2;i=10157
        //nodeElement.NodeId.ExpandedText: nsu=http://test.org/UA/Data/ ;ns=2;i=10157
        //
        //nodeElement.DisplayName: Boilers
        //nodeElement.NodeId: nsu=http://opcfoundation.org/UA/Boiler/ ;ns=4;i=1240
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/Boiler/
;ns=4;i=1240
        //
        //nodeElement.DisplayName: ServerType
        //nodeElement.NodeId: ServerType
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2004
    }
}
```

VB.NET

```
' Shows how to obtain references of all kinds to nodes of all classes, from the
"Server" node in the address space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class Browse
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain nodes under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.Browse( _
                    endpointDescriptor, _
                    UAObjectIds.Server, _
                    New UABrowseParameters(UANodeClass.All, New UANodeId()
{UAReferenceTypeIds.References}))
                ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain nodes under a given node of the OPC-UA address
space.
Rem For each node, it displays its browse name and node ID.
```

```
Option Explicit

Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"

Dim NodeDescriptor: Set NodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UANodeDescriptor")
Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
BrowsePathParser.DefaultNamespaceUriString = "http://test.org/UA/Data/"
NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar")

Dim BrowseParameters: Set BrowseParameters =
CreateObject("OpcLabs.EasyOpc.UA.UABrowseParameters")
BrowseParameters.StandardName = "AllForwardReferences"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.Browse(EndpointDescriptor, NodeDescriptor,
BrowseParameters)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo NodeElement.BrowseName & ": " & NodeElement.NodeId
Next
```

13.2.13.2 Examples - OPC Unified Architecture - Browsing for data nodes

C#

```
// This example shows how to obtain "data nodes" (objects, variables and properties)
under the "Objects" node in the address
// space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseDataNodes
```

```
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain data nodes under "Objects" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseDataNodes(endpointDescriptor);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: Server
        //nodeElement.NodeId: Server
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2253
        //
        //nodeElement.DisplayName: Data
        //nodeElement.NodeId: nsu = http://test.org/UA/Data/ ;ns=2;i=10157
        //nodeElement.NodeId.ExpandedText: nsu = http://test.org/UA/Data/ ;ns=2;i=10157
        //
        //nodeElement.DisplayName: Boilers
        //nodeElement.NodeId: nsu = http://opcfoundation.org/UA/Boiler/ ;ns=4;i=1240
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/Boiler/
;ns=4;i=1240
        //
        //nodeElement.DisplayName: MemoryBuffers
        //nodeElement.NodeId: nsu = http://samples.org/UA/memorybuffer ;ns=7;i=1025
        //nodeElement.NodeId.ExpandedText: nsu = http://samples.org/UA/memorybuffer
;ns=7;i=1025
    }
```

```
}
```

VB.NET

```
' This example shows how to obtain "data nodes" (objects, variables and properties)
under the "Objects" node in the address
' space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseDataNodes
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain data nodes under "Objects" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseDataNodes(endpointDescriptor)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2253
            'nodeElement.DisplayName: Server
            '
            'nodeElement.NodeId: nsu=http://test.org/UA/Data/;i=10157
            'nodeElement.DisplayName: Data
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/Boiler/;i=1240
            'nodeElement.DisplayName: Boilers
            '
            'nodeElement.NodeId: nsu=http://samples.org/UA/memorybuffer;i=1025
            'nodeElement.DisplayName: MemoryBuffers
```

```
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
// For each node, it displays its browse name and node ID.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "BrowseDataNodes.h"

namespace _EasyUAClient
{
    void BrowseDataNodes::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            _UANodeElementCollectionPtr NodeElementsPtr = ClientPtr->BrowseDataNodes(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10791");

            // Display results
            IEnumVARIANTPtr EnumNodeElementPtr = NodeElementsPtr->GetEnumerator();
            _variant_t vNodeElement;
            while (EnumNodeElementPtr->Next(1, &vNodeElement, NULL) == S_OK)
            {
                _UANodeElementPtr NodeElementPtr(vNodeElement);
                _tprintf(_T("%s: "), (LPCTSTR)CW2CT(NodeElementPtr->BrowseName-
>ToString));
                _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(NodeElementPtr->NodeId->ToString));
                vNodeElement.Clear();
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to obtain all data nodes (objects and variables)
// under a given node of the OPC-UA address space. For each node, it displays
// its browse name and node ID.

class procedure BrowseDataNodes.Main;
var
  Client: EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  NodeElement: _UANodeElement;
```

```
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  NodeElements := Client.BrowseDataNodes(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10791');

  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn(NodeElement.BrowseName.ToString, ': ', NodeElement.NodeId.ToString);
  end;
end;
```

Object Pascal

```
// This example shows how to obtain all data nodes (objects and variables)
// under a given node of the OPC-UA address space. For each node, it displays
// its browse name and node ID.

class procedure BrowseDataNodes.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  try
    NodeElements := Client.BrowseDataNodes(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10791');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn(NodeElement.BrowseName.ToString, ': ', NodeElement.NodeId.ToString);
  end;
end;
```

PHP

```
// This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
// For each node, it displays its browse name and node ID.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $NodeElements = $Client->BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("%s: %s\n", $NodeElement->BrowseName, $NodeElement->NodeId);
}
```

PowerScript

```
// This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
// For each node, it displays its browse name and node ID.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Perform the operation
OLEObject nodeElements
TRY
    nodeElements = client.BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
Int i
FOR i = 0 TO nodeElements.Count - 1
    OLEObject nodeElement
    nodeElement = nodeElements.Item[i]
```

```
    mle_outputtext.Text = mle_outputtext.Text + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "nodeElement.NodeId: " +
nodeElement.NodeId.DisplayString + "~r~n"
    mle_outputtext.Text = mle_outputtext.Text + "nodeElement.DisplayName: " +
nodeElement.DisplayName + "~r~n"
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to obtain all data nodes (objects and variables) under a given
node of the OPC-UA address space.
# For each node, it displays its browse name and node ID.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Perform the operation
nodeElements = client.BrowseDataNodes('http://opcua.demo-
this.com:51211/UA/SampleServer', 'nsu=http://test.org/UA/Data/;i=10791')

# Display results
for nodeElement in nodeElements:
    print(nodeElement.BrowseName, ': ', nodeElement.NodeId)
```

Visual Basic (VB 6.)

```
Rem This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
Rem For each node, it displays its browse name and node ID.

Private Sub BrowseDataNodes_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Dim NodeElements As UANodeElementCollection
    Set NodeElements = Client.BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    Dim NodeElement: For Each NodeElement In NodeElements
        OutputText = OutputText & NodeElement.BrowseName & ": " & NodeElement.NodeId &
vbCrLf
```

```
    Next
End Sub
```

VBScript

```
Rem This example shows how to obtain all data nodes (objects and variables) under a
given node of the OPC-UA address space.
Rem For each node, it displays its browse name and node ID.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim NodeElements: Set NodeElements = Client.BrowseDataNodes("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10791")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo NodeElement.BrowseName & ": " & NodeElement.NodeId
Next
```

13.2.13.3 Examples - OPC Unified Architecture - Browsing for data variables

C#

```
// This example shows how to obtain data variables under the "Server" node in the
address space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseDataVariables
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
```

```
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain variables under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
UAObjectIds.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerStatus
        //nodeElement.NodeId: Server_ServerStatus
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
    }
}
```

Object Pascal

```
// This example shows how to obtain data variables under the "Server" node
// in the address space.

class procedure BrowseDataVariables.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
  ServerNodeId: _UANodeId;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
```

```
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain variables under "Server" node
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  try
    NodeElements := Client.BrowseDataVariables(EndpointDescriptor,
ServerNodeId.ExpandedText);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn;
    WriteLn('nodeElement.NodeId: ', NodeElement.NodeId.ToString);
    WriteLn('nodeElement.NodeId.ExpandedText: ', NodeElement.NodeId.ExpandedText);
    WriteLn('nodeElement.DisplayName: ', NodeElement.DisplayName);
  end;

  // Example output:
  //
  //nodeElement.NodeId: Server_ServerStatus
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
  //nodeElement.DisplayName: ServerStatus
end;
```

PHP

```
// This example shows how to obtain data variables under the "Server" node
// in the address space.

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain variables under "Server" node
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";
```

```
try
{
    $NodeElements = $Client->BrowseDataVariables($EndpointDescriptor, $ServerNodeId-
>ExpandedText);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("\n");
    printf("nodeElement.NodeId: %s\n", $NodeElement->NodeId);
    printf("nodeElement.NodeId.ExpandedText: %s\n", $NodeElement->NodeId-
>ExpandedText);
    printf("nodeElement.DisplayName: %s\n", $NodeElement->DisplayName);
}

// Example output:
//
//nodeElement.NodeId: Server_ServerStatus
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
//nodeElement.DisplayName: ServerStatus
```

VB.NET

```
' This example shows how to obtain data variables under the "Server" node in the
address space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseDataVariables
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain variables under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
UAObjectIds.Server)
```

```
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2256
            'nodeElement.DisplayName: ServerStatus
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain data variables under the "Server" node in the
address space.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain variables under "Server" node
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
On Error Resume Next
Dim NodeElementCollection: Set NodeElementCollection =
Client.BrowseDataVariables(endpointDescriptor, ServerNodeId.ExpandedText)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElementCollection
    WScript.Echo
    WScript.Echo "nodeElement.NodeId: " & NodeElement.NodeId
    WScript.Echo "nodeElement.NodeId.ExpandedText: " & NodeElement.NodeId.ExpandedText
    WScript.Echo "nodeElement.DisplayName: " & NodeElement.DisplayName
Next
```

```
' Example output:
'
'nodeElement.NodeId: Server_ServerStatus
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2256
'nodeElement.DisplayName: ServerStatus
```

13.2.13.4 Examples - OPC Unified Architecture - Browsing for methods

C#

```
// This example shows how to obtain all method nodes under a given node of the OPC-UA
address space.
// For each node, it displays its browse name and node ID.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseMethods
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain methods under the specified node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseMethods(endpointDescriptor,
"nsu=http://test.org/UA/Data/ ;i=10755");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
                Console.WriteLine($"{nodeElement.BrowseName}: {nodeElement.NodeId}");
```

```
        }

        // Example output:
        //ScalarMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10756
        //ScalarMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10759
        //ScalarMethod3: nsu = http://test.org/UA/Data/ ;ns=2;i=10762
        //ArrayMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10765
        //ArrayMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10768
        //ArrayMethod3: nsu = http://test.org/UA/Data/ ;ns=2;i=10771
        //UserScalarMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10774
        //UserScalarMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10777
        //UserArrayMethod1: nsu = http://test.org/UA/Data/ ;ns=2;i=10780
        //UserArrayMethod2: nsu = http://test.org/UA/Data/ ;ns=2;i=10783
    }
}
```

PHP

```
// This example shows how to obtain all method nodes under a given node of the OPC-UA
address space.
// For each node, it displays its browse name and node ID.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $NodeElements = $Client->BrowseMethods("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10755");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}


// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("%s: %s\n", $NodeElement->BrowseName, $NodeElement->NodeId);
}
```

VBScript

```
Rem This example shows how to obtain all method nodes under a given node of the OPC-UA
address space.
Rem For each node, it displays its browse name and node ID.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
```

```
Dim NodeElements: Set NodeElements = Client.BrowseMethods("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/ ;i=10755")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElements
    WScript.Echo NodeElement.BrowseName & ": " & NodeElement.NodeId
Next


' Example output:
'ScalarMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10756
'ScalarMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10759
'ScalarMethod3: nsu=http://test.org/UA/Data/ ;ns=2;i=10762
'ArrayMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10765
'ArrayMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10768
'ArrayMethod3: nsu=http://test.org/UA/Data/ ;ns=2;i=10771
'UserScalarMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10774
'UserScalarMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10777
'UserArrayMethod1: nsu=http://test.org/UA/Data/ ;ns=2;i=10780
'UserArrayMethod2: nsu=http://test.org/UA/Data/ ;ns=2;i=10783
```

13.2.13.5 Examples - OPC Unified Architecture - Browsing for objects

C#

```
// This example shows how to obtain objects under the "Server" node in the address
space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseObjects
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
```

```
            var client = new EasyUAClient();

            // Obtain objects under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseObjects(endpointDescriptor,
UAObjectIds.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerCapabilities
        //nodeElement.NodeId: Server_ServerCapabilities
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2268
        //
        //nodeElement.DisplayName: ServerDiagnostics
        //nodeElement.NodeId: Server_ServerDiagnostics
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2274
        //
        //nodeElement.DisplayName: VendorServerInfo
        //nodeElement.NodeId: Server_VendorServerInfo
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2295
        //
        //nodeElement.DisplayName: ServerRedundancy
        //nodeElement.NodeId: Server_ServerRedundancy
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=2296
        //
        //nodeElement.DisplayName: Namespaces
        //nodeElement.NodeId: Server_Namespaces
        //nodeElement.NodeId.ExpandedText: nsu = http://opcfoundation.org/UA/ ;i=11715
    }
}
```

Object Pascal

```
// This example shows how to obtain objects under the "Server" node
// in the address space.
```

```
class procedure BrowseObjects.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
  ServerNodeId: _UANodeId;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain objects under "Server" node
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  try
    NodeElements := Client.BrowseObjects(EndpointDescriptor,
ServerNodeId.ExpandedText);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn;
    WriteLn('nodeElement.NodeId: ', NodeElement.NodeId.ToString);
    WriteLn('nodeElement.NodeId.ExpandedText: ', NodeElement.NodeId.ExpandedText);
    WriteLn('nodeElement.DisplayName: ', NodeElement.DisplayName);
  end;

  // Example output:
  //
  //nodeElement.NodeId: Server_ServerCapabilities
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
  //nodeElement.DisplayName: ServerCapabilities
  //
  //nodeElement.NodeId: Server_ServerDiagnostics
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
  //nodeElement.DisplayName: ServerDiagnostics
  //
  //nodeElement.NodeId: Server_VendorServerInfo
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
  //nodeElement.DisplayName: VendorServerInfo
  //
```

```
  //nodeElement.NodeId: Server_ServerRedundancy
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
  //nodeElement.DisplayName: ServerRedundancy
  //
  //nodeElement.NodeId: Server_Namespaces
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
  //nodeElement.DisplayName: Namespaces

end;
```

PHP

```
// This example shows how to obtain objects under the "Server" node
// in the address space.

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain variables under "Server" node
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";

try
{
    $NodeElements = $Client->BrowseObjects($EndpointDescriptor, $ServerNodeId-
>ExpandedText);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("\n");
    printf("nodeElement.NodeId: %s\n", $NodeElement->NodeId);
    printf("nodeElement.NodeId.ExpandedText: %s\n", $NodeElement->NodeId-
>ExpandedText);
    printf("nodeElement.DisplayName: %s\n", $NodeElement->DisplayName);
}

// Example output:
//
//nodeElement.NodeId: Server_ServerCapabilities
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
//nodeElement.DisplayName: ServerCapabilities
//
//nodeElement.NodeId: Server_ServerDiagnostics
```

```
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
//nodeElement.DisplayName: ServerDiagnostics
//
//nodeElement.NodeId: Server_VendorServerInfo
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
//nodeElement.DisplayName: VendorServerInfo
//
//nodeElement.NodeId: Server_ServerRedundancy
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
//nodeElement.DisplayName: ServerRedundancy
//
//nodeElement.NodeId: Server_Namespaces
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
//nodeElement.DisplayName: Namespaces
```

VB.NET

```
' This example shows how to obtain objects under the "Server" node in the address
space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseObjects
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain objects under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseObjects(endpointDescriptor,
UAObjectIds.Server)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
```

```
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2268
            'nodeElement.DisplayName: ServerCapabilities
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2274
            'nodeElement.DisplayName: ServerDiagnostics
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2295
            'nodeElement.DisplayName: VendorServerInfo
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2296
            'nodeElement.DisplayName: ServerRedundancy
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain objects under the "Server" node in the address
space.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain objects under "Server" node
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
On Error Resume Next
Dim NodeElementCollection: Set NodeElementCollection =
Client.BrowseObjects(endpointDescriptor, ServerNodeId.ExpandedText)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElementCollection
    WScript.Echo
    WScript.Echo "nodeElement.NodeId: " & NodeElement.NodeId
    WScript.Echo "nodeElement.NodeId.ExpandedText: " & NodeElement.NodeId.ExpandedText
    WScript.Echo "nodeElement.DisplayName: " & NodeElement.DisplayName
Next

' Example output:
'
```

```
'nodeElement.NodeId: Server_ServerCapabilities
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2268
'nodeElement.DisplayName: ServerCapabilities
'
'nodeElement.NodeId: Server_ServerDiagnostics
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2274
'nodeElement.DisplayName: ServerDiagnostics
'
'nodeElement.NodeId: Server_VendorServerInfo
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2295
'nodeElement.DisplayName: VendorServerInfo
'
'nodeElement.NodeId: Server_ServerRedundancy
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2296
'nodeElement.DisplayName: ServerRedundancy
'
'nodeElement.NodeId: Server_Namespaces
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=11715
'nodeElement.DisplayName: Namespaces
```

13.2.13.6 Examples - OPC Unified Architecture - Browsing for properties

C#

```
// This example shows how to obtain properties under the "Server" node in the address
space.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class BrowseProperties
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain properties under "Server" node
            UANodeElementCollection nodeElementCollection;
            try
            {
```

```
                nodeElementCollection = client.BrowseProperties(endpointDescriptor,
UAObjectIds.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UANodeElement nodeElement in nodeElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName);
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId);
                Console.WriteLine("nodeElement.NodeId.ExpandedText: {0}",
nodeElement.NodeId.ExpandedText);
            }
        }

        // Example output:
        //
        //nodeElement.DisplayName: ServerArray
        //nodeElement.NodeId: Server_ServerArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
        //
        //nodeElement.DisplayName: NamespaceArray
        //nodeElement.NodeId: Server_NamespaceArray
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
        //
        //nodeElement.DisplayName: ServiceLevel
        //nodeElement.NodeId: Server_ServiceLevel
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
        //
        //nodeElement.DisplayName: Auditing
        //nodeElement.NodeId: Server_Auditing
        //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
    }
}
```

Object Pascal

```
// This example shows how to obtain properties under the "Server" node
// in the address space.

class procedure BrowseProperties.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  EndpointDescriptor: string;
  NodeElement: _UANodeElement;
  NodeElementEnumerator: IEnumVariant;
  NodeElements: _UANodeElementCollection;
  ServerNodeId: _UANodeId;
```

```
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain properties under "Server" node
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  try
    NodeElements := Client.BrowseProperties(EndpointDescriptor,
ServerNodeId.ExpandedText);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  NodeElementEnumerator := NodeElements.GetEnumerator;
  while (NodeElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    NodeElement := IUnknown(Element) as _UANodeElement;
    WriteLn;
    WriteLn('nodeElement.NodeId: ', NodeElement.NodeId.ToString);
    WriteLn('nodeElement.NodeId.ExpandedText: ', NodeElement.NodeId.ExpandedText);
    WriteLn('nodeElement.DisplayName: ', NodeElement.DisplayName);
  end;

  // Example output:
  //
  //nodeElement.NodeId: Server_ServerArray
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
  //nodeElement.DisplayName: ServerArray
  //
  //nodeElement.NodeId: Server_NamespaceArray
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
  //nodeElement.DisplayName: NamespaceArray
  //
  //nodeElement.NodeId: Server_ServiceLevel
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
  //nodeElement.DisplayName: ServiceLevel
  //
  //nodeElement.NodeId: Server_Auditing
  //nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
  //nodeElement.DisplayName: Auditing

end;
```

PHP

```
// This example shows how to obtain properties under the "Server" node
// in the address space.
```

```
$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain variables under "Server" node
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ServerNodeId->StandardName = "Server";

try
{
    $NodeElements = $Client->BrowseProperties($EndpointDescriptor, $ServerNodeId-
>ExpandedText);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
foreach ($NodeElements as $NodeElement)
{
    printf("\n");
    printf("nodeElement.NodeId: %s\n", $NodeElement->NodeId);
    printf("nodeElement.NodeId.ExpandedText: %s\n", $NodeElement->NodeId-
>ExpandedText);
    printf("nodeElement.DisplayName: %s\n", $NodeElement->DisplayName);
}

// Example output:
//
//nodeElement.NodeId: Server_ServerArray
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
//nodeElement.DisplayName: ServerArray
//
//nodeElement.NodeId: Server_NamespaceArray
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
//nodeElement.DisplayName: NamespaceArray
//
//nodeElement.NodeId: Server_ServiceLevel
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
//nodeElement.DisplayName: ServiceLevel
//
//nodeElement.NodeId: Server_Auditing
//nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
//nodeElement.DisplayName: Auditing
```

VB.NET

```
' This example shows how to obtain properties under the "Server" node in the address
space.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class BrowseProperties
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain properties under "Server" node
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseProperties(endpointDescriptor,
UAObjectIds.Server)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each nodeElement As UANodeElement In nodeElementCollection
                Console.WriteLine()
                Console.WriteLine("nodeElement.NodeId: {0}", nodeElement.NodeId)
                Console.WriteLine("nodeElement.DisplayName: {0}",
nodeElement.DisplayName)
            Next nodeElement

            ' Example output:
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2254
            'nodeElement.DisplayName: ServerArray
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2255
            'nodeElement.DisplayName: NamespaceArray
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2267
            'nodeElement.DisplayName: ServiceLevel
            '
            'nodeElement.NodeId: nsu=http://opcfoundation.org/UA/;i=2994
            'nodeElement.DisplayName: Auditing            }
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain properties under the "Server" node in the address
space.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain properties under "Server" node
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
On Error Resume Next
Dim NodeElementCollection: Set NodeElementCollection =
Client.BrowseProperties(endpointDescriptor, ServerNodeId.ExpandedText)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim NodeElement: For Each NodeElement In NodeElementCollection
    WScript.Echo
    WScript.Echo "nodeElement.NodeId: " & NodeElement.NodeId
    WScript.Echo "nodeElement.NodeId.ExpandedText: " & NodeElement.NodeId.ExpandedText
    WScript.Echo "nodeElement.DisplayName: " & NodeElement.DisplayName
Next

' Example output:
'
'nodeElement.NodeId: Server_ServerArray
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2254
'nodeElement.DisplayName: ServerArray
'
'nodeElement.NodeId: Server_NamespaceArray
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2255
'nodeElement.DisplayName: NamespaceArray
'
'nodeElement.NodeId: Server_ServiceLevel
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2267
'nodeElement.DisplayName: ServiceLevel
'
'nodeElement.NodeId: Server_Auditing
'nodeElement.NodeId.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2994
'nodeElement.DisplayName: Auditing
```

13.2.13.7 Examples - OPC Unified Architecture - Call a single method

C#

```
// This example shows how to call a single method, and pass arguments to and from it.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class CallMethod
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            object[] inputs =
            {
                false,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            };

            TypeCode[] typeCodes =
            {
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double
            };

            // Instantiate the client object
            var client = new EasyUAClient();
```

```
            // Perform the operation
            object[] outputs;
            try
            {
                outputs = client.CallMethod(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10755",
                    "nsu=http://test.org/UA/Data/;i=10756",
                    inputs,
                    typeCodes);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            for (int i = 0; i < outputs.Length; i++)
                Console.WriteLine("outputs[{0}]: {1}", i, outputs[i]);

            // Example output:
            //outputs[0]: False
            //outputs[1]: 1
            //outputs[2]: 2
            //outputs[3]: 3
            //outputs[4]: 4
            //outputs[5]: 5
            //outputs[6]: 6
            //outputs[7]: 7
            //outputs[8]: 8
            //outputs[9]: 9
            //outputs[10]: 10
        }
    }
}
```

VB.NET

```
' This example shows how to call a single method, and pass arguments to and from it.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class CallMethod
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Dim inputs() As Object =
```

```
            { _
                False,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            }

            Dim typeCodes() As TypeCode =
            { _
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double
            }

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Perform the operation
            Dim outputs() As Object
            Try
                outputs = client.CallMethod( _
                    endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;i=10755", _
                    "nsu=http://test.org/UA/Data/;i=10756", _
                    inputs, _
                    typeCodes)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For i As Integer = 0 To outputs.Length - 1
                Console.WriteLine("outputs[{0}]: {1}", i, outputs(i))
            Next i

            ' Example output:
            'outputs[0]: False
            'outputs[1]: 1
            'outputs[2]: 2
            'outputs[3]: 3
```

```
            'outputs[4]: 4
            'outputs[5]: 5
            'outputs[6]: 6
            'outputs[7]: 7
            'outputs[8]: 8
            'outputs[9]: 9
            'outputs[10]: 10
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to call a single method, and pass arguments to and from it.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "CallMethod.h"

namespace _EasyUAClient
{
    void CallMethod::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            CComSafeArray<VARIANT> inputs(11);
            inputs.SetAt(0, _variant_t(false));
            inputs.SetAt(1, _variant_t(1));
            inputs.SetAt(2, _variant_t(2));
            inputs.SetAt(3, _variant_t(3));
            inputs.SetAt(4, _variant_t(4));
            inputs.SetAt(5, _variant_t(5));
            inputs.SetAt(6, _variant_t(6));
            inputs.SetAt(7, _variant_t(7));
            inputs.SetAt(8, _variant_t(8));
            inputs.SetAt(9, _variant_t(9));
            inputs.SetAt(10, _variant_t(10));
            CComVariant vInputs(inputs);

            CComSafeArray<VARIANT> typeCodes(11);
            typeCodes.SetAt(0, _variant_t(TypeCode_Boolean));
            typeCodes.SetAt(1, _variant_t(TypeCode_SByte));
            typeCodes.SetAt(2, _variant_t(TypeCode_Byte));
            typeCodes.SetAt(3, _variant_t(TypeCode_Int16));
            typeCodes.SetAt(4, _variant_t(TypeCode_UInt16));
            typeCodes.SetAt(5, _variant_t(TypeCode_Int32));
            typeCodes.SetAt(6, _variant_t(TypeCode_UInt32));
            typeCodes.SetAt(7, _variant_t(TypeCode_Int64));
            typeCodes.SetAt(8, _variant_t(TypeCode_UInt64));
            typeCodes.SetAt(9, _variant_t(TypeCode_Single));
            typeCodes.SetAt(10, _variant_t(TypeCode_Double));
            CComVariant vTypeCodes(typeCodes);

            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));
```

```
            // Perform the operation
            CComSafeArray<VARIANT> outputs;
            outputs.Attach(ClientPtr->CallMethod(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10755",
                L"nsu=http://test.org/UA/Data/;i=10756",
                &vInputs,
                &vTypeCodes));

            // Display results
            for (int i = outputs.GetLowerBound(); i <= outputs.GetUpperBound(); i++)
            {
                _variant_t output(outputs[i]);
                _variant_t vString;
                vString.ChangeType(VT_BSTR, &output);
                _tprintf(_T("outputs(%d): %s\n"), i, (LPCTSTR)CW2CT((_bstr_t)vString));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to call a single method, and pass arguments to and
// from it.

class procedure CallMethod.Main;
var
  Client: EasyUAClient;
  I: Cardinal;
  Inputs: OleVariant;
  Outputs: OleVariant;
  TypeCodes: OleVariant;
begin
  Inputs := VarArrayCreate([0, 10], varVariant);
  Inputs[0] := False;
  Inputs[1] := 1;
  Inputs[2] := 2;
  Inputs[3] := 3;
  Inputs[4] := 4;
  Inputs[5] := 5;
  Inputs[6] := 6;
  Inputs[7] := 7;
  Inputs[8] := 8;
  Inputs[9] := 9;
  Inputs[10] := 10;

  TypeCodes := VarArrayCreate([0, 10], varVariant);
  TypeCodes[0] := TypeCode_Boolean;
  TypeCodes[1] := TypeCode_SByte;
  TypeCodes[2] := TypeCode_Byte;
  TypeCodes[3] := TypeCode_Int16;
  TypeCodes[4] := TypeCode_UInt16;
  TypeCodes[5] := TypeCode_Int32;
  TypeCodes[6] := TypeCode_UInt32;
```

```
  TypeCodes[7] := TypeCode_Int64;
  TypeCodes[8] := TypeCode_UInt64;
  TypeCodes[9] := TypeCode_Single;
  TypeCodes[10] := TypeCode_Double;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Outputs).VType := varArray or varVariant;
  TVarData(Outputs).VArray := PVarArray(Client.CallMethod(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10755',
    'nsu=http://test.org/UA/Data/;i=10756',
    PSafeArray(TVarData(Inputs).VArray),
    PSafeArray(TVarData(TypeCodes).VArray)));

  // Display results
  for I := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
      try
        WriteLn('outputs(', I, '): ', Outputs[I]);
      except
        on EVariantError do WriteLn('*** Error displaying the value');
      end;
end;
```

Object Pascal

```
// This example shows how to call a single method, and pass arguments to and
// from it.

class procedure CallMethod.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  Inputs: OleVariant;
  Outputs: OleVariant;
  TypeCodes: OleVariant;
begin
  Inputs := VarArrayCreate([0, 10], varVariant);
  Inputs[0] := False;
  Inputs[1] := 1;
  Inputs[2] := 2;
  Inputs[3] := 3;
  Inputs[4] := 4;
  Inputs[5] := 5;
  Inputs[6] := 6;
  Inputs[7] := 7;
  Inputs[8] := 8;
  Inputs[9] := 9;
  Inputs[10] := 10;

  TypeCodes := VarArrayCreate([0, 10], varVariant);
  TypeCodes[0] := TypeCode_Boolean;
  TypeCodes[1] := TypeCode_SByte;
  TypeCodes[2] := TypeCode_Byte;
  TypeCodes[3] := TypeCode_Int16;
```

```
  TypeCodes[4] := TypeCode_UInt16;
  TypeCodes[5] := TypeCode_Int32;
  TypeCodes[6] := TypeCode_UInt32;
  TypeCodes[7] := TypeCode_Int64;
  TypeCodes[8] := TypeCode_UInt64;
  TypeCodes[9] := TypeCode_Single;
  TypeCodes[10] := TypeCode_Double;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  try
    TVarData(Outputs).VType := varArray or varVariant;
    TVarData(Outputs).VArray := PVarArray(Client.CallMethod(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10755',
      'nsu=http://test.org/UA/Data/;i=10756',
      Inputs,
      TypeCodes));
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  for I := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
      WriteLn('outputs(', I, '): ', Outputs[I]);
end;
```

PHP

```
// This example shows how to call a single method, and pass arguments to and from it.

$inputs[0] = false;
$inputs[1] = 1;
$inputs[2] = 2;
$inputs[3] = 3;
$inputs[4] = 4;
$inputs[5] = 5;
$inputs[6] = 6;
$inputs[7] = 7;
$inputs[8] = 8;
$inputs[9] = 9;
$inputs[10] = 10;

$typeCodes[0] = 3;    // TypeCode.Boolean
$typeCodes[1] = 5;    // TypeCode.SByte
$typeCodes[2] = 6;    // TypeCode.Byte
$typeCodes[3] = 7;    // TypeCode.Int16
$typeCodes[4] = 8;    // TypeCode.UInt16
$typeCodes[5] = 9;    // TypeCode.Int32
$typeCodes[6] = 10;   // TypeCode.UInt32
$typeCodes[7] = 11;   // TypeCode.Int64
```

```
$typeCodes[8] = 12;   // TypeCode.UInt64
$typeCodes[9] = 13;   // TypeCode.Single
$typeCodes[10] = 14;  // TypeCode.Double

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $outputs = $Client->CallMethod(
        "http://opcua.demo-this.com:51211/UA/SampleServer",
        "nsu=http://test.org/UA/Data/;i=10755",
        "nsu=http://test.org/UA/Data/;i=10756",
        $inputs,
        $typeCodes);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display results
for ($i = 0; $i < count($outputs); $i++)
{
    printf("outputs[%d]: %s\n", $i, $outputs[$i]);
}
```

Python

```
# This example shows how to call a single method, and pass arguments to and from it.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

#
inputs = [False, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
typeCodes = [
    3,  # TypeCode.Boolean
    5,  # TypeCode.SByte
    6,  # TypeCode.Byte
    7,  # TypeCode.Int16
    8,  # TypeCode.UInt16
    9,  # TypeCode.Int32
    10, # TypeCode.UInt32
    11, # TypeCode.Int64
    12, # TypeCode.UInt64
    13, # TypeCode.Single
    14  # TypeCode.Double
    ]


# Perform the operation
outputs = client.CallMethod(
```

```
    'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10755',
    'nsu=http://test.org/UA/Data/;i=10756',
    inputs,
    typeCodes)
# outputs[0] contains the actual results; outputs[1] and outputs[2] are copies of
inputs and typeCodes.


# Display results
for i, value in enumerate(outputs[0]):
     print('outputs[', i, ']: ', value)

# Example output:
#outputs[0]: False
#outputs[1]: 1
#outputs[2]: 2
#outputs[3]: 3
#outputs[4]: 4
#outputs[5]: 5
#outputs[6]: 6
#outputs[7]: 7
#outputs[8]: 8
#outputs[9]: 9.0
#outputs[10]: 10.0
```

Visual Basic (VB 6.)

```
Rem This example shows how to call a single method, and pass arguments to and from it.

Private Sub CallMethod_Main_Command_Click()
    OutputText = ""

    Dim inputs(10)
    inputs(0) = False
    inputs(1) = 1
    inputs(2) = 2
    inputs(3) = 3
    inputs(4) = 4
    inputs(5) = 5
    inputs(6) = 6
    inputs(7) = 7
    inputs(8) = 8
    inputs(9) = 9
    inputs(10) = 10

    Dim typeCodes(10)
    typeCodes(0) = 3    ' TypeCode.Boolean
    typeCodes(1) = 5    ' TypeCode.SByte
    typeCodes(2) = 6    ' TypeCode.Byte
    typeCodes(3) = 7    ' TypeCode.Int16
    typeCodes(4) = 8    ' TypeCode.UInt16
    typeCodes(5) = 9    ' TypeCode.Int32
    typeCodes(6) = 10   ' TypeCode.UInt32
    typeCodes(7) = 11   ' TypeCode.Int64
    typeCodes(8) = 12   ' TypeCode.UInt64
```

```
    typeCodes(9) = 13   ' TypeCode.Single
    typeCodes(10) = 14  ' TypeCode.Double

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Dim outputs As Variant
    outputs = Client.CallMethod( _
        "http://opcua.demo-this.com:51211/UA/SampleServer", _
        "nsu=http://test.org/UA/Data/;i=10755", _
        "nsu=http://test.org/UA/Data/;i=10756", _
        inputs, _
        typeCodes)
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    Dim i: For i = LBound(outputs) To UBound(outputs)
        On Error Resume Next
        OutputText = OutputText & "outputs(" & i & "): " & outputs(i) & vbCrLf
        If Err <> 0 Then OutputText = OutputText & "*** Error" & vbCrLf ' occurrs with
types not recognized by VB6
        On Error GoTo 0
    Next
End Sub
```

VBScript

```
Rem This example shows how to call a single method, and pass arguments to and from it.

Option Explicit

Dim inputs(10)
inputs(0) = False
inputs(1) = 1
inputs(2) = 2
inputs(3) = 3
inputs(4) = 4
inputs(5) = 5
inputs(6) = 6
inputs(7) = 7
inputs(8) = 8
inputs(9) = 9
inputs(10) = 10

Dim typeCodes(10)
typeCodes(0) = 3    ' TypeCode.Boolean
typeCodes(1) = 5    ' TypeCode.SByte
typeCodes(2) = 6    ' TypeCode.Byte
typeCodes(3) = 7    ' TypeCode.Int16
typeCodes(4) = 8    ' TypeCode.UInt16
```

```
typeCodes(5) = 9     ' TypeCode.Int32
typeCodes(6) = 10    ' TypeCode.UInt32
typeCodes(7) = 11    ' TypeCode.Int64
typeCodes(8) = 12    ' TypeCode.UInt64
typeCodes(9) = 13    ' TypeCode.Single
typeCodes(10) = 14   ' TypeCode.Double

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim outputs: outputs = Client.CallMethod( _
    "http://opcua.demo-this.com:51211/UA/SampleServer", _
    "nsu=http://test.org/UA/Data/;i=10755", _
    "nsu=http://test.org/UA/Data/;i=10756", _
    inputs, _
    typeCodes)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim i: For i = LBound(outputs) To UBound(outputs)
    On Error Resume Next
    WScript.Echo "outputs(" & i & "): " & outputs(i)
    If Err <> 0 Then WScript.Echo "*** Error"    ' occurrs with types not recognized by
VBScript
    On Error Goto 0
Next
```

13.2.13.8 Examples - OPC Unified Architecture - Call multiple methods

C#

```
// This example shows how to call multiple methods, and pass arguments to and
// from them.

using System;
using System.Diagnostics;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class CallMultipleMethods
    {
        public static void Main1()
```

```
        {

            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            UANodeDescriptor nodeDescriptor = "nsu=http://test.org/UA/Data/;i=10755";

            object[] inputs1 =
            {
                false,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            };

            TypeCode[] typeCodes1 =
            {
                TypeCode.Boolean,
                TypeCode.SByte,
                TypeCode.Byte,
                TypeCode.Int16,
                TypeCode.UInt16,
                TypeCode.Int32,
                TypeCode.UInt32,
                TypeCode.Int64,
                TypeCode.UInt64,
                TypeCode.Single,
                TypeCode.Double
            };

            object[] inputs2 =
            {
                false,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10,
                "eleven"
            };

            TypeCode[] typeCodes2 =
            {
```

```
    TypeCode.Boolean,
    TypeCode.SByte,
    TypeCode.Byte,
    TypeCode.Int16,
    TypeCode.UInt16,
    TypeCode.Int32,
    TypeCode.UInt32,
    TypeCode.Int64,
    TypeCode.UInt64,
    TypeCode.Single,
    TypeCode.Double,
    TypeCode.String
};

// Instantiate the client object
var client = new EasyUAClient();

// Perform the operation
ValueArrayResult[] results = client.CallMultipleMethods(new[]
{
    new UACallArguments(endpointDescriptor, nodeDescriptor,
        "nsu=http://test.org/UA/Data/;i=10756", inputs1, typeCodes1),
    new UACallArguments(endpointDescriptor, nodeDescriptor,
        "nsu=http://test.org/UA/Data/;i=10774", inputs2, typeCodes2)
});

// Display results
for (int i = 0; i < results.Length; i++)
{
    Console.WriteLine();
    Console.WriteLine("results[{0}]:", i);

    ValueArrayResult result = results[i];
    if (result.Succeeded)
    {
        object[] outputs = result.ValueArray;
        Debug.Assert(outputs != null);

        for (int j = 0; j < outputs.Length; j++)
            Console.WriteLine("    outputs[{0}]: {1}", j, outputs[j]);
    }
    else
        Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief);
}

// Example output:
//results[0]:
//outputs[0]: False
//outputs[1]: 1
//outputs[2]: 2
//outputs[3]: 3
//outputs[4]: 4
//outputs[5]: 5
//outputs[6]: 6
//outputs[7]: 7
//outputs[8]: 8
//outputs[9]: 9
```

```
            //outputs[10]: 10

            //results[1]:
            //outputs[0]: False
            //outputs[1]: 1
            //outputs[2]: 2
            //outputs[3]: 3
            //outputs[4]: 4
            //outputs[5]: 5
            //outputs[6]: 6
            //outputs[7]: 7
            //outputs[8]: 8
            //outputs[9]: 9
            //outputs[10]: 10
            //outputs[11]: eleven
        }
    }
}
```

VB.NET

```
' This example shows how to call multiple methods, and pass arguments to and
' from them.

Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class CallMultipleMethods
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"
            Dim nodeDescriptor As UANodeDescriptor =
"nsu=http://test.org/UA/Data/;i=10755"

            Dim inputs1() As Object =
            { _
                False,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10
            }

            Dim typeCodes1() As TypeCode =
            { _
```

```
            TypeCode.Boolean,
            TypeCode.SByte,
            TypeCode.Byte,
            TypeCode.Int16,
            TypeCode.UInt16,
            TypeCode.Int32,
            TypeCode.UInt32,
            TypeCode.Int64,
            TypeCode.UInt64,
            TypeCode.Single,
            TypeCode.Double
        }

        Dim inputs2() As Object =
        { _
            False,
            1,
            2,
            3,
            4,
            5,
            6,
            7,
            8,
            9,
            10,
            "eleven"
        }

        Dim typeCodes2() As TypeCode =
        { _
            TypeCode.Boolean,
            TypeCode.SByte,
            TypeCode.Byte,
            TypeCode.Int16,
            TypeCode.UInt16,
            TypeCode.Int32,
            TypeCode.UInt32,
            TypeCode.Int64,
            TypeCode.UInt64,
            TypeCode.Single,
            TypeCode.Double,
            TypeCode.String
        }

        ' Instantiate the client object
        Dim client = New EasyUAClient()

        ' Perform the operation
        Dim results() As ValueArrayResult = client.CallMultipleMethods(New
UACallArguments() _
    { _
        New UACallArguments(endpointDescriptor, nodeDescriptor, _
            "nsu=http://test.org/UA/Data/;i=10756", inputs1, typeCodes1), _
        New UACallArguments(endpointDescriptor, nodeDescriptor, _
            "nsu=http://test.org/UA/Data/;i=10774", inputs2, typeCodes2) _
    } _
```

```
            )

            ' Display results
            For i As Integer = 0 To results.Length - 1
                Console.WriteLine()
                Console.WriteLine("results[{0}]:", i)

                Dim result As ValueArrayResult = results(i)
                If result.Succeeded Then
                    Dim outputs As Object() = result.ValueArray
                    Debug.Assert(outputs IsNot Nothing)

                    For j As Integer = 0 To outputs.Length - 1
                        Console.WriteLine("    outputs[{0}]: {1}", j, outputs(j))
                    Next j
                Else
                    Console.WriteLine("*** Failure: {0}", result.ErrorMessageBrief)
                End If
            Next i

            ' Example output:
            'outputs[0]: False
            'outputs[1]: 1
            'outputs[2]: 2
            'outputs[3]: 3
            'outputs[4]: 4
            'outputs[5]: 5
            'outputs[6]: 6
            'outputs[7]: 7
            'outputs[8]: 8
            'outputs[9]: 9
            'outputs[10]: 10

            'results[1]:
            'outputs[0]: False
            'outputs[1]: 1
            'outputs[2]: 2
            'outputs[3]: 3
            'outputs[4]: 4
            'outputs[5]: 5
            'outputs[6]: 6
            'outputs[7]: 7
            'outputs[8]: 8
            'outputs[9]: 9
            'outputs[10]: 10
            'outputs[11]: eleven
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to call multiple methods, and pass arguments to and from
them.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
```

```
#include "CallMultipleMethods.h"

namespace _EasyUAClient
{
    void CallMultipleMethods::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            CComSafeArray<VARIANT> inputs1(11);
            inputs1.SetAt(0, _variant_t(false));
            inputs1.SetAt(1, _variant_t(1));
            inputs1.SetAt(2, _variant_t(2));
            inputs1.SetAt(3, _variant_t(3));
            inputs1.SetAt(4, _variant_t(4));
            inputs1.SetAt(5, _variant_t(5));
            inputs1.SetAt(6, _variant_t(6));
            inputs1.SetAt(7, _variant_t(7));
            inputs1.SetAt(8, _variant_t(8));
            inputs1.SetAt(9, _variant_t(9));
            inputs1.SetAt(10, _variant_t(10));

            CComSafeArray<VARIANT> typeCodes1(11);
            typeCodes1.SetAt(0, _variant_t(TypeCode_Boolean));
            typeCodes1.SetAt(1, _variant_t(TypeCode_SByte));
            typeCodes1.SetAt(2, _variant_t(TypeCode_Byte));
            typeCodes1.SetAt(3, _variant_t(TypeCode_Int16));
            typeCodes1.SetAt(4, _variant_t(TypeCode_UInt16));
            typeCodes1.SetAt(5, _variant_t(TypeCode_Int32));
            typeCodes1.SetAt(6, _variant_t(TypeCode_UInt32));
            typeCodes1.SetAt(7, _variant_t(TypeCode_Int64));
            typeCodes1.SetAt(8, _variant_t(TypeCode_UInt64));
            typeCodes1.SetAt(9, _variant_t(TypeCode_Single));
            typeCodes1.SetAt(10, _variant_t(TypeCode_Double));

            _UACallArgumentsPtr CallArguments1Ptr(__uuidof(UACallArguments));
            CallArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            CallArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10755";
            CallArguments1Ptr->MethodNodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10756";
            CallArguments1Ptr->InputArguments = inputs1;
            CallArguments1Ptr->InputTypeCodes = typeCodes1;

            CComSafeArray<VARIANT> inputs2(12);
            inputs2.SetAt(0, _variant_t(false));
            inputs2.SetAt(1, _variant_t(1));
            inputs2.SetAt(2, _variant_t(2));
            inputs2.SetAt(3, _variant_t(3));
            inputs2.SetAt(4, _variant_t(4));
            inputs2.SetAt(5, _variant_t(5));
            inputs2.SetAt(6, _variant_t(6));
            inputs2.SetAt(7, _variant_t(7));
            inputs2.SetAt(8, _variant_t(8));
            inputs2.SetAt(9, _variant_t(9));
            inputs2.SetAt(10, _variant_t(10));
```

```
            inputs2.SetAt(11, _variant_t(L"eleven"));

            CComSafeArray<VARIANT> typeCodes2(12);
            typeCodes2.SetAt(0, _variant_t(TypeCode_Boolean));
            typeCodes2.SetAt(1, _variant_t(TypeCode_SByte));
            typeCodes2.SetAt(2, _variant_t(TypeCode_Byte));
            typeCodes2.SetAt(3, _variant_t(TypeCode_Int16));
            typeCodes2.SetAt(4, _variant_t(TypeCode_UInt16));
            typeCodes2.SetAt(5, _variant_t(TypeCode_Int32));
            typeCodes2.SetAt(6, _variant_t(TypeCode_UInt32));
            typeCodes2.SetAt(7, _variant_t(TypeCode_Int64));
            typeCodes2.SetAt(8, _variant_t(TypeCode_UInt64));
            typeCodes2.SetAt(9, _variant_t(TypeCode_Single));
            typeCodes2.SetAt(10, _variant_t(TypeCode_Double));
            typeCodes2.SetAt(11, _variant_t(TypeCode_String));

            _UACallArgumentsPtr CallArguments2Ptr(__uuidof(UACallArguments));
            CallArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            CallArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10755";
            CallArguments2Ptr->MethodNodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10774";
            CallArguments2Ptr->InputArguments = inputs2;
            CallArguments2Ptr->InputTypeCodes = typeCodes2;

            CComSafeArray<VARIANT> arguments(2);
            arguments.SetAt(0, _variant_t((IDispatch*)CallArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)CallArguments2Ptr));
            CComVariant vArguments(arguments);

            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->CallMultipleMethods(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _tprintf(_T("\n"));
                _tprintf(_T("results(%d):\n"), i);
                _ValueArrayResultPtr ResultPtr = results[i];

                if (ResultPtr->Exception == NULL)
                {
                    CComSafeArray<VARIANT> outputs = ResultPtr->ValueArray;
                    for (int j = outputs.GetLowerBound(); j <= outputs.GetUpperBound();
j++)
                    {
                        _variant_t output(outputs[j]);
                        _variant_t vString;
                        vString.ChangeType(VT_BSTR, &output);
                        _tprintf(_T("    outputs(%d): %s\n"), i,
(LPCTSTR)CW2CT((_bstr_t)vString));
                    }
```

```
                }
                else
                    _tprintf(_T("*** Error: %s\n"), (LPCTSTR)CW2CT(ResultPtr-
>Exception->ToString));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to call multiple methods, and pass arguments to and
// from them.

class procedure CallMultipleMethods.Main;
var
  Arguments: OleVariant;
  CallArguments1, CallArguments2: _UACallArguments;
  Client: EasyUAClient;
  I, J: Cardinal;
  Inputs1, Inputs2: OleVariant;
  Outputs: OleVariant;
  Result: _ValueArrayResult;
  Results: OleVariant;
  TypeCodes1, TypeCodes2: OleVariant;
begin
  Inputs1 := VarArrayCreate([0, 10], varVariant);
  Inputs1[0] := False;
  Inputs1[1] := 1;
  Inputs1[2] := 2;
  Inputs1[3] := 3;
  Inputs1[4] := 4;
  Inputs1[5] := 5;
  Inputs1[6] := 6;
  Inputs1[7] := 7;
  Inputs1[8] := 8;
  Inputs1[9] := 9;
  Inputs1[10] := 10;

  TypeCodes1 := VarArrayCreate([0, 10], varVariant);
  TypeCodes1[0] := TypeCode_Boolean;
  TypeCodes1[1] := TypeCode_SByte;
  TypeCodes1[2] := TypeCode_Byte;
  TypeCodes1[3] := TypeCode_Int16;
  TypeCodes1[4] := TypeCode_UInt16;
  TypeCodes1[5] := TypeCode_Int32;
  TypeCodes1[6] := TypeCode_UInt32;
  TypeCodes1[7] := TypeCode_Int64;
  TypeCodes1[8] := TypeCode_UInt64;
  TypeCodes1[9] := TypeCode_Single;
  TypeCodes1[10] := TypeCode_Double;

  CallArguments1 := CoUACallArguments.Create;
  CallArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
```

```
  CallArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments1.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10756';
  CallArguments1.InputArguments := PSafeArray(TVarData(Inputs1).VArray);
  CallArguments1.InputTypeCodes := PSafeArray(TVarData(TypeCodes1).VArray);

  Inputs2 := VarArrayCreate([0, 11], varVariant);
  Inputs2[0] := False;
  Inputs2[1] := 1;
  Inputs2[2] := 2;
  Inputs2[3] := 3;
  Inputs2[4] := 4;
  Inputs2[5] := 5;
  Inputs2[6] := 6;
  Inputs2[7] := 7;
  Inputs2[8] := 8;
  Inputs2[9] := 9;
  Inputs2[10] := 10;
  Inputs2[11] := 'eleven';

  TypeCodes2 := VarArrayCreate([0, 11], varVariant);
  TypeCodes2[0] := TypeCode_Boolean;
  TypeCodes2[1] := TypeCode_SByte;
  TypeCodes2[2] := TypeCode_Byte;
  TypeCodes2[3] := TypeCode_Int16;
  TypeCodes2[4] := TypeCode_UInt16;
  TypeCodes2[5] := TypeCode_Int32;
  TypeCodes2[6] := TypeCode_UInt32;
  TypeCodes2[7] := TypeCode_Int64;
  TypeCodes2[8] := TypeCode_UInt64;
  TypeCodes2[9] := TypeCode_Single;
  TypeCodes2[10] := TypeCode_Double;
  TypeCodes2[11] := TypeCode_String;

  CallArguments2 := CoUACallArguments.Create;
  CallArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments2.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10774';
  CallArguments2.InputArguments := PSafeArray(TVarData(Inputs2).VArray);
  CallArguments2.InputTypeCodes := PSafeArray(TVarData(TypeCodes2).VArray);

  Arguments := VarArrayCreate([0, 1], varVariant);
  Arguments[0] := CallArguments1;
  Arguments[1] := CallArguments2;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.CallMultipleMethods(
    PSafeArray(TVarData(Arguments).VArray)));
```

```
  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteLn;
    WriteLn('results(', I, '):');
    Result := IInterface(Results[I]) as _ValueArrayResult;

    if Result.Exception = nil then
      begin
        TVarData(Outputs).VType := varArray or varVariant;
        TVarData(Outputs).VArray := PVarArray(Result.ValueArray);
        for J := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
          try
            WriteLn('    ', 'outputs(', J, '): ', Outputs[J]);
          except
            on EVariantError do WriteLn('*** Error displaying the value');
          end;
      end
    else
      WriteLn('*** Error: ', Result.Exception.ToString);
  end;
end;
```

Object Pascal

```
// This example shows how to call multiple methods, and pass arguments to and
// from them.

class procedure CallMultipleMethods.Main;
var
  Arguments: OleVariant;
  CallArguments1, CallArguments2: _UACallArguments;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I, J: Cardinal;
  Inputs1, Inputs2: OleVariant;
  Outputs: OleVariant;
  Result: _ValueArrayResult;
  Results: OleVariant;
  TypeCodes1, TypeCodes2: OleVariant;
begin
  Inputs1 := VarArrayCreate([0, 10], varVariant);
  Inputs1[0] := False;
  Inputs1[1] := 1;
  Inputs1[2] := 2;
  Inputs1[3] := 3;
  Inputs1[4] := 4;
  Inputs1[5] := 5;
  Inputs1[6] := 6;
  Inputs1[7] := 7;
  Inputs1[8] := 8;
  Inputs1[9] := 9;
  Inputs1[10] := 10;

  TypeCodes1 := VarArrayCreate([0, 10], varVariant);
  TypeCodes1[0] := TypeCode_Boolean;
  TypeCodes1[1] := TypeCode_SByte;
  TypeCodes1[2] := TypeCode_Byte;
```

```
  TypeCodes1[3] := TypeCode_Int16;
  TypeCodes1[4] := TypeCode_UInt16;
  TypeCodes1[5] := TypeCode_Int32;
  TypeCodes1[6] := TypeCode_UInt32;
  TypeCodes1[7] := TypeCode_Int64;
  TypeCodes1[8] := TypeCode_UInt64;
  TypeCodes1[9] := TypeCode_Single;
  TypeCodes1[10] := TypeCode_Double;

  CallArguments1 := CoUACallArguments.Create;
  CallArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments1.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10756';
  CallArguments1.InputArguments := PSafeArray(TVarData(Inputs1).VArray);
  CallArguments1.InputTypeCodes := PSafeArray(TVarData(TypeCodes1).VArray);

  Inputs2 := VarArrayCreate([0, 11], varVariant);
  Inputs2[0] := False;
  Inputs2[1] := 1;
  Inputs2[2] := 2;
  Inputs2[3] := 3;
  Inputs2[4] := 4;
  Inputs2[5] := 5;
  Inputs2[6] := 6;
  Inputs2[7] := 7;
  Inputs2[8] := 8;
  Inputs2[9] := 9;
  Inputs2[10] := 10;
  Inputs2[11] := 'eleven';

  TypeCodes2 := VarArrayCreate([0, 11], varVariant);
  TypeCodes2[0] := TypeCode_Boolean;
  TypeCodes2[1] := TypeCode_SByte;
  TypeCodes2[2] := TypeCode_Byte;
  TypeCodes2[3] := TypeCode_Int16;
  TypeCodes2[4] := TypeCode_UInt16;
  TypeCodes2[5] := TypeCode_Int32;
  TypeCodes2[6] := TypeCode_UInt32;
  TypeCodes2[7] := TypeCode_Int64;
  TypeCodes2[8] := TypeCode_UInt64;
  TypeCodes2[9] := TypeCode_Single;
  TypeCodes2[10] := TypeCode_Double;
  TypeCodes2[11] := TypeCode_String;

  CallArguments2 := CoUACallArguments.Create;
  CallArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  CallArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10755';
  CallArguments2.MethodNodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10774';
  CallArguments2.InputArguments := PSafeArray(TVarData(Inputs2).VArray);
  CallArguments2.InputTypeCodes := PSafeArray(TVarData(TypeCodes2).VArray);
```

```
  Arguments := VarArrayCreate([0, 1], varVariant);
  Arguments[0] := CallArguments1;
  Arguments[1] := CallArguments2;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.CallMultipleMethods(
    Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteLn;
    WriteLn('results(', I, '):');
    Result := IInterface(Results[I]) as _ValueArrayResult;

    if Result.Exception = nil then
      begin
        TVarData(Outputs).VType := varArray or varVariant;
        TVarData(Outputs).VArray := PVarArray(Result.ValueArray);
        for J := VarArrayLowBound(Outputs, 1) to VarArrayHighBound(Outputs, 1) do
          WriteLn('    ', 'outputs(', J, '): ', Outputs[J]);
      end
    else
      WriteLn('*** Failure: ', Result.Exception.ToString);
  end;
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to call multiple methods, and pass arguments to and from
them.

Private Sub CallMultipleMethods_Main_Command_Click()
    OutputText = ""

    Dim inputs1(10)
    inputs1(0) = False
    inputs1(1) = 1
    inputs1(2) = 2
    inputs1(3) = 3
    inputs1(4) = 4
    inputs1(5) = 5
    inputs1(6) = 6
    inputs1(7) = 7
    inputs1(8) = 8
    inputs1(9) = 9
    inputs1(10) = 10

    Dim typeCodes1(10)
    typeCodes1(0) = 3     ' TypeCode.Boolean
    typeCodes1(1) = 5     ' TypeCode.SByte
    typeCodes1(2) = 6     ' TypeCode.Byte
    typeCodes1(3) = 7     ' TypeCode.Int16
```

```
    typeCodes1(4) = 8    ' TypeCode.UInt16
    typeCodes1(5) = 9    ' TypeCode.Int32
    typeCodes1(6) = 10   ' TypeCode.UInt32
    typeCodes1(7) = 11   ' TypeCode.Int64
    typeCodes1(8) = 12   ' TypeCode.UInt64
    typeCodes1(9) = 13   ' TypeCode.Single
    typeCodes1(10) = 14  ' TypeCode.Double

    Dim CallArguments1 As New UACallArguments
    CallArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    CallArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10755"
    CallArguments1.MethodNodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10756"
    ' Use SetXXXX methods instead of array-type property setters in Visual Basic 6.0
    CallArguments1.SetInputArguments inputs1
    CallArguments1.SetInputTypeCodes typeCodes1

    Dim inputs2(11)
    inputs2(0) = False
    inputs2(1) = 1
    inputs2(2) = 2
    inputs2(3) = 3
    inputs2(4) = 4
    inputs2(5) = 5
    inputs2(6) = 6
    inputs2(7) = 7
    inputs2(8) = 8
    inputs2(9) = 9
    inputs2(10) = 10
    inputs2(11) = "eleven"

    Dim typeCodes2(11)
    typeCodes2(0) = 3    ' TypeCode.Boolean
    typeCodes2(1) = 5    ' TypeCode.SByte
    typeCodes2(2) = 6    ' TypeCode.Byte
    typeCodes2(3) = 7    ' TypeCode.Int16
    typeCodes2(4) = 8    ' TypeCode.UInt16
    typeCodes2(5) = 9    ' TypeCode.Int32
    typeCodes2(6) = 10   ' TypeCode.UInt32
    typeCodes2(7) = 11   ' TypeCode.Int64
    typeCodes2(8) = 12   ' TypeCode.UInt64
    typeCodes2(9) = 13   ' TypeCode.Single
    typeCodes2(10) = 14  ' TypeCode.Double
    typeCodes2(11) = 18  ' TypeCode.String

    Dim CallArguments2 As New UACallArguments
    CallArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    CallArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10755"
    CallArguments2.MethodNodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10774"
    ' Use SetXXXX methods instead of array-type property setters in Visual Basic 6.0
    CallArguments2.SetInputArguments inputs2
    CallArguments2.SetInputTypeCodes typeCodes2
```

```
    Dim arguments(1) As Variant
    Set arguments(0) = CallArguments1
    Set arguments(1) = CallArguments2

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    Dim results As Variant
    results = Client.CallMultipleMethods(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        OutputText = OutputText & vbCrLf
        OutputText = OutputText & "results(" & i & "):" & vbCrLf
        Dim Result As ValueArrayResult: Set Result = results(i)

        If Result.Exception Is Nothing Then
            Dim outputs As Variant: outputs = Result.ValueArray
            Dim j: For j = LBound(outputs) To UBound(outputs)
                On Error Resume Next
                OutputText = OutputText & Space(4) & "outputs(" & j & "): " &
outputs(j) & vbCrLf
                If Err <> 0 Then OutputText = OutputText & Space(4) & "*** Error" &
vbCrLf ' occurrs with types not recognized by VB6
                On Error GoTo 0
            Next
        Else
            OutputText = OutputText & "*** Error: " & Result.Exception & vbCrLf
        End If
    Next
End Sub
```

VBScript

```
Rem This example shows how to call multiple methods, and pass arguments to and from
them.

Option Explicit

Dim inputs1(10)
inputs1(0) = False
inputs1(1) = 1
inputs1(2) = 2
inputs1(3) = 3
inputs1(4) = 4
inputs1(5) = 5
inputs1(6) = 6
inputs1(7) = 7
inputs1(8) = 8
inputs1(9) = 9
inputs1(10) = 10

Dim typeCodes1(10)
typeCodes1(0) = 3     ' TypeCode.Boolean
typeCodes1(1) = 5     ' TypeCode.SByte
```

```
typeCodes1(2) = 6    ' TypeCode.Byte
typeCodes1(3) = 7    ' TypeCode.Int16
typeCodes1(4) = 8    ' TypeCode.UInt16
typeCodes1(5) = 9    ' TypeCode.Int32
typeCodes1(6) = 10   ' TypeCode.UInt32
typeCodes1(7) = 11   ' TypeCode.Int64
typeCodes1(8) = 12   ' TypeCode.UInt64
typeCodes1(9) = 13   ' TypeCode.Single
typeCodes1(10) = 14  ' TypeCode.Double

Dim CallArguments1: Set CallArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UACallArguments")
CallArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
CallArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10755"
CallArguments1.MethodNodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10756"
CallArguments1.InputArguments = inputs1
CallArguments1.InputTypeCodes = typeCodes1

Dim inputs2(11)
inputs2(0) = False
inputs2(1) = 1
inputs2(2) = 2
inputs2(3) = 3
inputs2(4) = 4
inputs2(5) = 5
inputs2(6) = 6
inputs2(7) = 7
inputs2(8) = 8
inputs2(9) = 9
inputs2(10) = 10
inputs2(11) = "eleven"

Dim typeCodes2(11)
typeCodes2(0) = 3    ' TypeCode.Boolean
typeCodes2(1) = 5    ' TypeCode.SByte
typeCodes2(2) = 6    ' TypeCode.Byte
typeCodes2(3) = 7    ' TypeCode.Int16
typeCodes2(4) = 8    ' TypeCode.UInt16
typeCodes2(5) = 9    ' TypeCode.Int32
typeCodes2(6) = 10   ' TypeCode.UInt32
typeCodes2(7) = 11   ' TypeCode.Int64
typeCodes2(8) = 12   ' TypeCode.UInt64
typeCodes2(9) = 13   ' TypeCode.Single
typeCodes2(10) = 14  ' TypeCode.Double
typeCodes2(11) = 18  ' TypeCode.String

Dim CallArguments2: Set CallArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UACallArguments")
CallArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
CallArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10755"
CallArguments2.MethodNodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10774"
```

```
CallArguments2.InputArguments = inputs2
CallArguments2.InputTypeCodes = typeCodes2

Dim arguments(1)
Set arguments(0) = CallArguments1
Set arguments(1) = CallArguments2

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
Dim results: results = Client.CallMultipleMethods(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    WScript.Echo
    WScript.Echo "results(" & i & "):"
    Dim Result: Set Result = results(i)

    If Result.Exception Is Nothing Then
        Dim outputs: outputs = Result.ValueArray
        Dim j: For j = LBound(outputs) To UBound(outputs)
            On Error Resume Next
            WScript.Echo Space(4) & "outputs(" & j & "): " & outputs(j)
            If Err <> 0 Then WScript.Echo Space(4) & "*** Error"   ' occurrs with types
not recognized by VBScript
            On Error Goto 0
        Next
    Else
        WScript.Echo "*** Error: " & Result.Exception
    End If
Next
```

13.2.13.9 Examples - OPC Unified Architecture - Callback using a lambda

C#

```
// This example shows how to subscribe to changes of a single monitored item, and
display the value of the item with each change
// using a callback method that is provided as lambda expression.

using System;
using OpcLabs.EasyOpc.UA;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeDataChange
    {
        public static void CallbackLambda()
        {
            UAEndpointDescriptor endpointDescriptor =
```

```
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            Console.WriteLine("Subscribing...");
            // The callback is a lambda expression the displays the value
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000,
                (sender, eventArgs) =>
                {
                    if (eventArgs.Succeeded)
                        Console.WriteLine("Value: {0}", eventArgs.AttributeData.Value);
                    else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                });

            Console.WriteLine("Processing data change events for 10 seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 2 seconds...");
            System.Threading.Thread.Sleep(2 * 1000);
        }
    }
}
```

F#

```
// This example shows how to subscribe to changes of a single monitored item, and
display the value of the item with each change
// using a callback method that is provided as a function delegate.

module _EasyUAClient.SubscribeDataChange

open OpcLabs.EasyOpc.UA
open System
open System.Threading

let CallbackFunction =

    let endpointDescriptor =
        new UAEndpointDescriptor("opc.tcp://opcua.demo-this.com:51210/UA/SampleServer")
        // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
        // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

    // Instantiate the client object
    let client = new EasyUAClient()

    Console.WriteLine("Subscribing...");
    // The callback is a delegate that displays the value
```

```
    let handle =
        client.SubscribeDataChange(
            endpointDescriptor,
            new UANodeDescriptor("nsu=http://test.org/UA/Data/;i=10853"),
            1000,
            new EasyUADataChangeNotificationEventHandler(
                fun sender eventArgs ->
                    if eventArgs.Succeeded then Console.WriteLine("Value: {0}",
eventArgs.AttributeData.Value)
                    else Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)))

    Console.WriteLine("Processing data change events for 10 seconds...")
    Thread.Sleep(10 * 1000)

    Console.WriteLine("Unsubscribing...")
    client.UnsubscribeAllMonitoredItems()

    Console.WriteLine("Waiting for 2 seconds...")
    Thread.Sleep(2 * 1000)
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item, and
display the value of the item with each change
' using a callback method that is provided as lambda expression.

Imports OpcLabs.EasyOpc.UA

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeDataChange
        Public Shared Sub CallbackLambda()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the value
            client.SubscribeDataChange( _
                endpointDescriptor, _
                "nsu=http://test.org/UA/Data/;i=10853", _
                1000, _
                Sub(sender, eventArgs)
                    If eventArgs.Succeeded Then
                        Console.WriteLine("Value: {0}", eventArgs.AttributeData.Value)
                    Else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                    End If
                End Sub)
```

```
            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

13.2.13.10 Examples - OPC Unified Architecture - Certificate in the platform-specific certificate store

C#

```
// This example demonstrates how to place the client certificate in the platform-specific (Windows,
Linux, ...) certificate
// store.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UAApplicationParameters
{
    class ApplicationCertificateStore
    {
        public static void PlatformSpecific()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Set the application certificate store path, which determines the location of the client
certificate.
            // Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore =
"CurrentUser\\My";

            // Do something - invoke an OPC read, to trigger creation of the certificate.
            var client = new EasyUAClient();
            try
            {
                client.ReadValue(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException().Message);
            }

            // The certificate will be located or created in the specified platform-specific
certificate store.
            // On Windows, when viewed by the certmgr.msc tool, it will be under
```

```
            // Certificates - Current User -> Personal -> Certificates.

            Console.WriteLine("Done.");
        }
    }
}
```

Object Pascal

```
// This example demonstrates how to place the client certificate
// in the platform-specific (Windows, Linux, ...) certificate store.

class procedure ApplicationCertificateStore.PlatformSpecific;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfiguration.Connect;

  // Set the application certificate store path, which determines the location of the client
certificate.
  // Note that this only works once in each host process.

ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore
:=
    'CurrentUser\My';

  // Do something - invoke an OPC read, to trigger creation of the certificate.
  Client := CoEasyUAClient.Create;
  try
    Value := Client.ReadValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
    end;
  end;

  // The certificate will be located or created in the specified platform-specific certificate store.
  // On Windows, when viewed by the certmgr.msc tool, it will be under
  // Certificates - Current User -> Personal -> Certificates.

  WriteLn('Done.');

  FreeAndNil(ClientConfiguration);
end;
```

PHP

```
// This example demonstrates how to place the client certificate
// in the platform-specific (Windows, Linux, ...) certificate store.


// The configuration object allows access to static behavior.
$ClientConfiguration = new COM("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration");

// Set the application certificate store path, which determines the location of the client
certificate.
// Note that this only works once in each host process.
$ClientConfiguration->SharedParameters->EngineParameters->ApplicationParameters-
>ApplicationCertificateStore =
```

```
    "CurrentUser\My";

// Do something - invoke an OPC read, to trigger creation of the certificate.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}

// The certificate will be located or created in the specified platform-specific certificate store.
// On Windows, when viewed by the certmgr.msc tool, it will be under
// Certificates - Current User -> Personal -> Certificates.

printf("Done.\n");
```

VB.NET

```
' This example demonstrates how to place the client certificate in the platform-specific (Windows,
Linux, ...) certificate store.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._UAApplicationParameters
    Friend Class ApplicationCertificateStore
        Public Shared Sub PlatformSpecific()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Set the application certificate store path, which determines the location of the client
certificate.
            ' Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore =
"CurrentUser\\My"

            ' Do something - invoke an OPC read, to trigger creation of the certificate.
            Dim client = New EasyUAClient()
            Try
                client.ReadValue(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}", uaException.GetBaseException.Message)
            End Try

            ' The certificate will be located or created in the specified platform-specific
certificate store.
            ' On Windows, when viewed by the certmgr.msc tool, it will be under
            ' Certificates - Current User -> Personal -> Certificates.

            Console.WriteLine("Done.")
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example demonstrates how to place the client certificate in the platform-specific (Windows,
```

```
Linux, ...) certificate
Rem store.
Rem Note: COM is only available on Windows.

Option Explicit

' The configuration object allows access to static behavior.
Dim ClientConfiguration: Set ClientConfiguration =
CreateObject("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration")
WScript.ConnectObject ClientConfiguration, "ClientConfiguration_"

' Set the application certificate store path, which determines the location of the client certificate.
' Note that this only works once in each host process.
ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationCertificateStore
= "CurrentUser\My"

' Do something - invoke an OPC read, to trigger some loggable entries.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' The certificate will be located or created in the specified platform-specific certificate store.
' On Windows, when viewed by the certmgr.msc tool, it will be under
' Certificates - Current User -> Personal -> Certificates.

WScript.Echo "Done."
```

13.2.13.11 Examples - OPC Unified Architecture - Change sampling rate of a single subscription

C#

```
// This example shows how to change the sampling rate of an existing monitored item
subscription.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class ChangeMonitoredItemSubscription
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int handle = client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Changing subscription...");
            client.ChangeMonitoredItemSubscription(handle, 100);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

Object Pascal

```
// This example shows how to change the sampling rate of an existing monitored
// item subscription.

type
  TClientEventHandlers110 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers110.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
```

```
      WriteLn(eventArgs.AttributeData.ToString)
  else
      WriteLn('*** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure ChangeMonitoredItemSubscription.Main;
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers110;
  Handle: Cardinal;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers110.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  Handle := Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  Client.ChangeMonitoredItemSubscription(Handle, 100);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how to change the sampling rate of an existing monitored item
subscription.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class ChangeMonitoredItemSubscription
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
```

```
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handle As Integer = _
                client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Changing subscription...")
            client.ChangeMonitoredItemSubscription(handle, 100)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to change the sampling rate of an existing monitored item
subscription.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeDataChange( _
    "http://opcua.demo-this.com:51211/UA/SampleServer", _
    "nsu=http://test.org/UA/Data/;i=10853", _
```

```
    1000)

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Changing subscription..."
Client.ChangeMonitoredItemSubscription handle, 100

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

13.2.13.12 Examples - OPC Unified Architecture - Change sampling rate of multiple subscriptions

C#

```
// This example shows how change the sampling rate of multiple existing monitored item
subscriptions.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class ChangeMultipleMonitoredItemSubscriptions
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
```

```
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Changing subscriptions...");
            client.ChangeMultipleMonitoredItemSubscriptions(handleArray, 100);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

Object Pascal

```
// This example shows how change the sampling rate of multiple existing monitored item
subscriptions.


type
  TClientEventHandlers111 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers111.OnDataChangeNotification(
```

```
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure ChangeMultipleMonitoredItemSubscriptions.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers111;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  OldMonitoringParameters, NewMonitoringParameters: _UAMonitoringParameters;
  SubscriptionChangeArguments: OleVariant;
  SubscriptionChangeArguments1, SubscriptionChangeArguments2,
SubscriptionChangeArguments3:
    _EasyUASubscriptionChangeArguments;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers111.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  OldMonitoringParameters := CoUAMonitoringParameters.Create;
  OldMonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := OldMonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := OldMonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := OldMonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
```

```
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Changing subscriptions...');
  NewMonitoringParameters := CoUAMonitoringParameters.Create;
  NewMonitoringParameters.SamplingInterval := 100;
  SubscriptionChangeArguments1 := CoEasyUASubscriptionChangeArguments.Create;
  SubscriptionChangeArguments1.Handle := Cardinal(HandleArray[0]);
  SubscriptionChangeArguments1.MonitoringParameters := NewMonitoringParameters;
  SubscriptionChangeArguments2 := CoEasyUASubscriptionChangeArguments.Create;
  SubscriptionChangeArguments2.Handle := Cardinal(HandleArray[1]);
  SubscriptionChangeArguments2.MonitoringParameters := NewMonitoringParameters;
  SubscriptionChangeArguments3 := CoEasyUASubscriptionChangeArguments.Create;
  SubscriptionChangeArguments3.Handle := Cardinal(HandleArray[2]);
  SubscriptionChangeArguments3.MonitoringParameters := NewMonitoringParameters;
  SubscriptionChangeArguments := VarArrayCreate([0, 2], varVariant);
  SubscriptionChangeArguments[0] := SubscriptionChangeArguments1;
  SubscriptionChangeArguments[1] := SubscriptionChangeArguments2;
  SubscriptionChangeArguments[2] := SubscriptionChangeArguments3;

  Client.ChangeMultipleMonitoredItemSubscriptions(SubscriptionChangeArguments);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how change the sampling rate of multiple existing monitored item
subscriptions.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel
```

```
Namespace UADocExamples._EasyUAClient
    Friend Class ChangeMultipleMonitoredItemSubscriptions
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                })

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Changing subscriptions...")
            client.ChangeMultipleMonitoredItemSubscriptions(handleArray, 100)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
```

```
    End Class
End Namespace
```

VBScript

```
Rem This example shows how change the sampling rate of multiple existing monitored item
subscriptions.

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim OldMonitoringParameters: Set OldMonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
OldMonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = OldMonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = OldMonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = OldMonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Changing subscriptions..."
Dim NewMonitoringParameters: Set NewMonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
NewMonitoringParameters.SamplingInterval = 100
Dim SubscriptionChangeArguments1: Set SubscriptionChangeArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUASubscriptionChangeArguments")
SubscriptionChangeArguments1.Handle = handleArray(0)
```

```
Set SubscriptionChangeArguments1.MonitoringParameters = NewMonitoringParameters
Dim SubscriptionChangeArguments2: Set SubscriptionChangeArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUASubscriptionChangeArguments")
SubscriptionChangeArguments2.Handle = handleArray(1)
Set SubscriptionChangeArguments2.MonitoringParameters = NewMonitoringParameters
Dim SubscriptionChangeArguments3: Set SubscriptionChangeArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUASubscriptionChangeArguments")
SubscriptionChangeArguments3.Handle = handleArray(2)
Set SubscriptionChangeArguments3.MonitoringParameters = NewMonitoringParameters
Dim subscriptionChangeArguments(2)
Set subscriptionChangeArguments(0) = SubscriptionChangeArguments1
Set subscriptionChangeArguments(1) = SubscriptionChangeArguments2
Set subscriptionChangeArguments(2) = SubscriptionChangeArguments3
Client.ChangeMultipleMonitoredItemSubscriptions subscriptionChangeArguments

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

13.2.13.13 Examples - OPC Unified Architecture - Create a node ID

Following examples are intended to show you many of the options that exist for constructing the node IDs.

C#

```
// This example shows different ways of constructing OPC UA node IDs.

using System;
using OpcLabs.BaseLib;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.AddressSpace.Parsing;
using OpcLabs.EasyOpc.UA.AddressSpace.Parsing.Extensions;
using OpcLabs.EasyOpc.UA.AddressSpace.Standard;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples._UANodeId
{
    class _Construction
    {
        // A node ID specifies a namespace (either by an URI or by an index), and an
identifier.
        // The identifier can be numeric (an integer), string, GUID, or opaque.
        public static void Main1()
        {

            // A node ID can be specified in string form (so-called expanded text).
            // The code below specifies a namespace URI (nsu=...), and an integer
identifier (i=...).
            UANodeId nodeId1 = new UANodeId("nsu=http://test.org/UA/Data/;i=10853");
            Console.WriteLine(nodeId1);


            // Similarly, with a string identifier (s=...).
            UANodeId nodeId2 = new
UANodeId("nsu=http://test.org/UA/Data/;s=someIdentifier");
            Console.WriteLine(nodeId2);


            // Actually, "s=" can be omitted (not recommended, though)
            UANodeId nodeId3 = new
UANodeId("nsu=http://test.org/UA/Data/;someIdentifier");
            Console.WriteLine(nodeId3);
            // Notice that the output is normalized - the "s=" is added again.


            // Similarly, with a GUID identifier (g=...)
            UANodeId nodeId4 = new UANodeId("nsu=http://test.org/UA/Data/;g=BAEAF004-
1E43-4A06-9EF0-E52010D5CD10");
            Console.WriteLine(nodeId4);
            // Notice that the output is normalized - uppercase letters in the GUI are
converted to lowercase, etc.


            // Similarly, with an opaque identifier (b=..., in Base64 encoding).
            UANodeId nodeId5 = new UANodeId("nsu=http://test.org/UA/Data/;b=AP8=");
            Console.WriteLine(nodeId5);


            // Namespace index can be used instead of namespace URI. The server is
allowed to change the namespace
            // indices between sessions (except for namespace 0), and for this reason,
you should avoid the use of
            // namespace indices, and rather use the namespace URIs whenever possible.
            UANodeId nodeId6 = new UANodeId("ns=2;i=10853");
            Console.WriteLine(nodeId6);


            // Namespace index can be also specified together with namespace URI. This
is still safe, but may be
            // a bit quicker to perform, because the client can just verify the
namespace URI instead of looking
            // it up.
```

```
            UANodeId nodeId7 = new
UANodeId("nsu=http://test.org/UA/Data/;ns=2;i=10853");
            Console.WriteLine(nodeId7);


            // When neither namespace URI nor namespace index are given, the node ID is
assumed to be in namespace
            // with index 0 and URI "http://opcfoundation.org/UA/", which is reserved
by OPC UA standard. There are
            // many standard nodes that live in this reserved namespace, but no nodes
specific to your servers will
            // be in the reserved namespace, and hence the need to specify the
namespace with server-specific nodes.
            UANodeId nodeId8 = new UANodeId("i=2254");
            Console.WriteLine(nodeId8);


            // If you attempt to pass in a string that does not conform to the syntax
rules,
            // a UANodeIdFormatException is thrown.
            try
            {
                UANodeId nodeId9 = new
UANodeId("nsu=http://test.org/UA/Data/;i=notAnInteger");
                Console.WriteLine(nodeId9);
            }
            catch (UANodeIdFormatException nodeIdFormatException)
            {
                Console.WriteLine($"*** Failure {nodeIdFormatException.Message}");
            }


            // There is a parser object that can be used to parse the expanded texts of
node IDs.
            UANodeIdParser nodeIdParser10 = new UANodeIdParser();
            UANodeId nodeId10 =
nodeIdParser10.Parse("nsu=http://test.org/UA/Data/;i=10853");
            Console.WriteLine(nodeId10);


            // The parser can be used if you want to parse the expanded text of the
node ID but do not want
            // exceptions be thrown.
            UANodeIdParser nodeIdParser11 = new UANodeIdParser();
            IStringParsingError stringParsingError =
                nodeIdParser11.TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger",
out UANodeId nodeId11);
            if (stringParsingError == null)
                Console.WriteLine(nodeId11);
            else
                Console.WriteLine($"*** Failure: {stringParsingError.Message}");


            // You can also use the parser if you have node IDs where you want the
default namespace be different
            // from the standard "http://opcfoundation.org/UA/".
            UANodeIdParser nodeIdParser12 = new
```

```
UANodeIdParser("http://test.org/UA/Data/");
            UANodeId nodeId12 = nodeIdParser12.Parse("i=10853");
            Console.WriteLine(nodeId12);


            // The namespace URI string (or the namespace index, or both) and the
identifier can be passed to the
            // constructor separately.
            UANodeId nodeId13 = new UANodeId("http://test.org/UA/Data/", 10853);
            Console.WriteLine(nodeId13);


            // You can create a "null" node ID. Such node ID does not actually identify
any valid node in OPC UA, but
            // is useful as a placeholder or as a starting point for further
modifications of its properties.
            UANodeId nodeId14 = new UANodeId();
            Console.WriteLine(nodeId14);


            // Properties of a node ID can be modified individually. The advantage of
this approach is that you do
            // not have to care about syntax of the node ID expanded text.
            UANodeId nodeId15 = new UANodeId();
            nodeId15.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId15.Identifier = 10853;
            Console.WriteLine(nodeId15);


            // The same as above, but using an object initializer list.
            UANodeId nodeId16 = new UANodeId
            {
                NamespaceUriString = "http://test.org/UA/Data/",
                Identifier = 10853
            };
            Console.WriteLine(nodeId16);


            // If you know the type of the identifier upfront, it is safer to use typed
properties that correspond
            // to specific types of identifier. Here, with an integer identifier.
            UANodeId nodeId17 = new UANodeId();
            nodeId17.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId17.NumericIdentifier = 10853;
            Console.WriteLine(nodeId17);


            // Similarly, with a string identifier.
            UANodeId nodeId18 = new UANodeId();
            nodeId18.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId18.StringIdentifier = "someIdentifier";
            Console.WriteLine(nodeId18);


            // Similarly, with a GUID identifier.
            UANodeId nodeId19 = new UANodeId();
            nodeId19.NamespaceUriString = "http://test.org/UA/Data/";
```

```
            nodeId19.GuidIdentifier = Guid.Parse("BAEAF004-1E43-4A06-9EF0-
E52010D5CD10");
            Console.WriteLine(nodeId19);


            // If you have GUID in its string form, the node ID object can parse it for
you.
            UANodeId nodeId20 = new UANodeId();
            nodeId20.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId20.GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10";
            Console.WriteLine(nodeId20);


            // And, with an opaque identifier.
            UANodeId nodeId21 = new UANodeId();
            nodeId21.NamespaceUriString = "http://test.org/UA/Data/";
            nodeId21.OpaqueIdentifier = new byte[] {0x00, 0xFF};
            Console.WriteLine(nodeId21);


            // Assigning an expanded text to a node ID parses the value being assigned
and sets all corresponding
            // properties accordingly.
            UANodeId nodeId22 = new UANodeId();
            nodeId22.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";
            Console.WriteLine(nodeId22);


            // There is an implicit conversion from a string (representing the expanded
text) to a node ID.
            // You can therefore use the expanded text (string) in place of any node ID
object directly.
            UANodeId nodeId23 = "nsu=http://test.org/UA/Data/;i=10853";
            Console.WriteLine(nodeId23);


            // There is a copy constructor as well, creating a clone of an existing
node ID.
            UANodeId nodeId24a = new UANodeId("nsu=http://test.org/UA/Data/;i=10853");
            Console.WriteLine(nodeId24a);
            UANodeId nodeId24b = new UANodeId(nodeId24a);
            Console.WriteLine(nodeId24b);


            // We have provided static classes with properties that correspond to all
standard nodes specified by
            // OPC UA. You can simply refer to these node IDs in your code.
            // The class names are UADataTypeIds, UAMethodIds, UAObjectIds,
UAObjectTypeIds, UAReferenceTypeIds,
            // UAVariableIds and UAVariableTypeIds.
            UANodeId nodeId25 = UAObjectIds.TypesFolder;
            Console.WriteLine(nodeId25);
            // When the UANodeId equals to one of the standard nodes, it is output in
the shortened form - as the standard
            // name only.
```

```
            // You can also refer to any standard node using its name (in a string
form).
            // Note that assigning a non-existing standard name is not allowed, and
throws ArgumentException.
            UANodeId nodeId26 = new UANodeId();
            nodeId26.StandardName = "TypesFolder";
            Console.WriteLine(nodeId26);


            // When you browse for nodes in the OPC UA server, every returned node
element contains a node ID that
            // you can use further.
            var client27 = new EasyUAClient();
            try
            {
                UANodeElementCollection nodeElementCollection27 = client27.Browse(
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
                if (nodeElementCollection27.Count != 0)
                {
                    UANodeId nodeId27 = nodeElementCollection27[0].NodeId;
                    Console.WriteLine(nodeId27);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("Failure: {0}",
uaException.GetBaseException().Message);
            }


            // As above, but using a constructor that takes a node element as an input.
            var client28 = new EasyUAClient();
            try
            {
                UANodeElementCollection nodeElementCollection28 = client28.Browse(
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
                if (nodeElementCollection28.Count != 0)
                {
                    UANodeId nodeId28 = new UANodeId(nodeElementCollection28[0]);
                    Console.WriteLine(nodeId28);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("Failure: {0}",
uaException.GetBaseException().Message);
            }


            // Or, there is an explicit conversion from a node element as well.
            var client29 = new EasyUAClient();
```

```
            try
            {
                UANodeElementCollection nodeElementCollection29 = client29.Browse(
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                    UAObjectIds.Server,
                    new UABrowseParameters(UANodeClass.All, new[] {
UAReferenceTypeIds.References }));
                if (nodeElementCollection29.Count != 0)
                {
                    UANodeId nodeId29 = (UANodeId) nodeElementCollection29[0];
                    Console.WriteLine(nodeId29);
                }
            }
            catch (UAException uaException)
            {
                Console.WriteLine("Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

Object Pascal

```
// This example shows different ways of constructing OPC UA node IDs.

class procedure _Construction.Main;
var
  BrowseParameters: _UABrowseParameters;
  Client27: _EasyUAClient;
  EndpointDescriptor: _UAEndpointDescriptor;
  NodeId1, NodeId2, NodeId3, NodeId4, NodeId5, NodeId6, NodeId7, NodeId8, NodeId9,
NodeId10,
  NodeId11, NodeId12, NodeId14, NodeId15, NodeId17, NodeId18,
  NodeId20, NodeId21, NodeId26, NodeId27,
  ServerNodeId, ReferencesNodeId: OpcLabs_EasyOpcUA_TLB._UANodeId;
  NodeId11Result: OleVariant;
  NodeIdParser10, NodeIdParser11, NodeIdParser12: _UANodeIdParser;
  NodeElements27: _UANodeElementCollection;
  OpaqueIdentifier21: Variant;
  ServerNodeDescriptor: _UANodeDescriptor;
  StringParsingError: _StringParsingError;
begin
  // A node ID specifies a namespace (either by an URI or by an index), and an
identifier.
  // The identifier can be numeric (an integer), string, GUID, or opaque.

  // A node ID can be specified in string form (so-called expanded text).
  // The code below specifies a namespace URI (nsu=...), and an integer identifier
(i=...).
  // Assigning an expanded text to a node ID parses the value being assigned and sets
all corresponding
  // properties accordingly.
  NodeId1 := CoUANodeId.Create;
  NodeId1.ExpandedText := 'nsu=http://test.org/UA/Data/;i=10853';
  WriteLn(NodeId1.ToString);
```

```
  // Similarly, with a string identifier (s=...).
  NodeId2 := CoUANodeId.Create;
  NodeId2.ExpandedText := 'nsu=http://test.org/UA/Data/;s=someIdentifier';
  WriteLn(NodeId2.ToString);


  // Actually, "s=" can be omitted (not recommended, though)
  NodeId3 := CoUANodeId.Create;
  NodeId3.ExpandedText := 'nsu=http://test.org/UA/Data/;someIdentifier';
  WriteLn(NodeId3.ToString);
  // Notice that the output is normalized - the "s=" is added again.


  // Similarly, with a GUID identifier (g=...)
  NodeId4 := CoUANodeId.Create;
  NodeId4.ExpandedText := 'nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-4A06-9EF0-
E52010D5CD10';
  WriteLn(NodeId4.ToString);
  // Notice that the output is normalized - uppercase letters in the GUI are converted
to lowercase, etc.


  // Similarly, with an opaque identifier (b=..., in Base64 encoding).
  NodeId5 := CoUANodeId.Create;
  NodeId5.ExpandedText := 'nsu=http://test.org/UA/Data/;b=AP8=';
  WriteLn(NodeId5.ToString);


  // Namespace index can be used instead of namespace URI. The server is allowed to
change the namespace
  // indices between sessions (except for namespace 0), and for this reason, you should
avoid the use of
  // namespace indices, and rather use the namespace URIs whenever possible.
  NodeId6 := CoUANodeId.Create;
  NodeId6.ExpandedText := 'ns=2;i=10853';
  WriteLn(NodeId6.ToString);


  // Namespace index can be also specified together with namespace URI. This is still
safe, but may be
  // a bit quicker to perform, because the client can just verify the namespace URI
instead of looking
  // it up.
  NodeId7 := CoUANodeId.Create;
  NodeId7.ExpandedText := 'nsu=http://test.org/UA/Data/;ns=2;i=10853';
  WriteLn(NodeId7.ToString);


  // When neither namespace URI nor namespace index are given, the node ID is assumed
to be in namespace
  // with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by OPC UA
standard. There are
  // many standard nodes that live in this reserved namespace, but no nodes specific to
your servers will
  // be in the reserved namespace, and hence the need to specify the namespace with
server-specific nodes.
```

```
  NodeId8 := CoUANodeId.Create;
  NodeId8.ExpandedText := 'i=2254';
  WriteLn(NodeId8.ToString);


  // If you attempt to pass in a string that does not conform to the syntax rules,
  // a UANodeIdFormatException is thrown.
  NodeId9 := CoUANodeId.Create;
  try
    NodeId9.ExpandedText := 'nsu=http://test.org/UA/Data/;i=notAnInteger';
    WriteLn(NodeId9.ToString);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.BaseException.Message]));
    end;
  end;


  // There is a parser object that can be used to parse the expanded texts of node IDs.
  NodeIdParser10 := CoUANodeIdParser.Create;
  NodeId10 := NodeIdParser10.Parse('nsu=http://test.org/UA/Data/;i=10853', False);
  WriteLn(NodeId10.ToString);


  // The parser can be used if you want to parse the expanded text of the node ID but
do not want
  // exceptions be thrown.
  NodeIdParser11 := CoUANodeIdParser.Create;
  StringParsingError :=
NodeIdParser11.TryParse('nsu=http://test.org/UA/Data/;i=notAnInteger', False,
NodeId11Result);
  if StringParsingError = nil then
    begin
      NodeId11 := IUnknown(NodeId11Result) as OpcLabs_EasyOpcUA_TLB._UANodeId;
      WriteLn(NodeId11.ToString);
    end
  else
    WriteLn(Format('*** Failure: %s', [StringParsingError.Message]));


  // You can also use the parser if you have node IDs where you want the default
namespace be different
  // from the standard "http://opcfoundation.org/UA/".
  NodeIdParser12 := CoUANodeIdParser.Create;
  NodeIdParser12.DefaultNamespaceUriString := 'http://test.org/UA/Data/';
  NodeId12 := NodeIdParser12.Parse('i=10853', False);
  WriteLn(NodeId12.ToString);


  // You can create a "null" node ID. Such node ID does not actually identify any valid
node in OPC UA, but
  // is useful as a placeholder or as a starting point for further modifications of its
properties.
  NodeId14 := CoUANodeId.Create;
  WriteLn(NodeId14.ToString);
```

```
  // Properties of a node ID can be modified individually. The advantage of this
approach is that you do
  // not have to care about syntax of the node ID expanded text.
  NodeId15 := CoUANodeId.Create;
  NodeId15.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId15.Identifier := 10853;
  WriteLn(NodeId15.ToString);


  // If you know the type of the identifier upfront, it is safer to use typed
properties that correspond
  // to specific types of identifier. Here, with an integer identifier.
  NodeId17 := CoUANodeId.Create;
  NodeId17.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId17.NumericIdentifier := 10853;
  WriteLn(NodeId17.ToString);


  // Similarly, with a string identifier.
  NodeId18 := CoUANodeId.Create;
  NodeId18.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId18.StringIdentifier := 'someIdentifier';
  WriteLn(NodeId18.ToString);


  // If you have GUID in its string form, the node ID object can parse it for you.
  NodeId20 := CoUANodeId.Create;
  NodeId20.NamespaceUriString := 'http://test.org/UA/Data/';
  NodeId20.GuidIdentifierString := 'BAEAF004-1E43-4A06-9EF0-E52010D5CD10';
  WriteLn(NodeId20.ToString);


  // And, with an opaque identifier.
  NodeId21 := CoUANodeId.Create;
  NodeId21.NamespaceUriString := 'http://test.org/UA/Data/';
//  OpaqueIdentifier21 := VarArrayCreate ([0, 1], varByte);
  OpaqueIdentifier21 := VarArrayCreate ([0, 1], varVariant);
  OpaqueIdentifier21[0] := $00;
  OpaqueIdentifier21[1] := $FF;
  NodeId21.OpaqueIdentifier := PSafeArray (TVarData (OpaqueIdentifier21).VArray);
  WriteLn(NodeId21.ToString);


  // We have built-in a list of all standard nodes specified by OPC UA. You can simply
refer to these node IDs in your code.
  // You can refer to any standard node using its name (in a string form).
  // Note that assigning a non-existing standard name is not allowed, and throws
ArgumentException.
  NodeId26 := CoUANodeId.Create;
  NodeId26.StandardName := 'TypesFolder';
  WriteLn(NodeId26.ToString);
  // When the UANodeId equals to one of the standard nodes, it is output in the
shortened form - as the standard name only.


  // When you browse for nodes in the OPC UA server, every returned node element
```

```
contains a node ID that
  // you can use further.
  Client27 := CoEasyUAClient.Create;
  EndpointDescriptor := CoUAEndpointDescriptor.Create;
  EndpointDescriptor.UrlString := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // Browse from the Server node.
  ServerNodeId := CoUANodeId.Create;
  ServerNodeId.StandardName := 'Server';
  ServerNodeDescriptor := CoUANodeDescriptor.Create;
  ServerNodeDescriptor.NodeId := ServerNodeId;
  // Browse all References.
  ReferencesNodeId := CoUANodeId.Create;
  ReferencesNodeId.StandardName := 'References';

  BrowseParameters := CoUABrowseParameters.Create;
  BrowseParameters.NodeClasses := UANodeClass_All;  // this is the default, anyway
  BrowseParameters.ReferenceTypeIds.Add(ReferencesNodeId);

  try
    NodeElements27 := Client27.Browse(EndpointDescriptor, ServerNodeDescriptor,
BrowseParameters);
    if NodeElements27.Count <> 0 then
    begin
      NodeId27 := NodeElements27[0].NodeId;
      WriteLn(NodeId27.ToString);
    end;
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.BaseException.Message]));
    end;
  end;
end;
```

PHP

```
// This example shows different ways of constructing OPC UA node IDs.

// A node ID specifies a namespace (either by an URI or by an index), and an
identifier.
// The identifier can be numeric (an integer), string, GUID, or opaque.

const UANodeClass_All = 255;


// A node ID can be specified in string form (so-called expanded text).
// The code below specifies a namespace URI (nsu=...), and an integer identifier
(i=...).
// Assigning an expanded text to a node ID parses the value being assigned and sets all
corresponding
// properties accordingly.
$NodeId1 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId1->ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";
printf("%s\n", $NodeId1);


// Similarly, with a string identifier (s=...).
```

```
$NodeId2 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId2->ExpandedText = "nsu=http://test.org/UA/Data/;s=someIdentifier";
printf("%s\n", $NodeId2);


// Actually, "s=" can be omitted (not recommended, though)
$NodeId3 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId3->ExpandedText = "nsu=http://test.org/UA/Data/;someIdentifier";
printf("%s\n", $NodeId3);
// Notice that the output is normalized - the "s=" is added again.


// Similarly, with a GUID identifier (g=...)
$NodeId4 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId4->ExpandedText = "nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-4A06-9EF0-
E52010D5CD10";
printf("%s\n", $NodeId4);
// Notice that the output is normalized - uppercase letters in the GUI are converted to
lowercase, etc.


// Similarly, with an opaque identifier (b=..., in Base64 encoding).
$NodeId5 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId5->ExpandedText = "nsu=http://test.org/UA/Data/;b=AP8=";
printf("%s\n", $NodeId5);


// Namespace index can be used instead of namespace URI. The server is allowed to
change the namespace
// indices between sessions (except for namespace 0), and for this reason, you should
avoid the use of
// namespace indices, and rather use the namespace URIs whenever possible.
$NodeId6 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId6->ExpandedText = "ns=2;i=10853";
printf("%s\n", $NodeId6);


// Namespace index can be also specified together with namespace URI. This is still
safe, but may be
// a bit quicker to perform, because the client can just verify the namespace URI
instead of looking
// it up.
$NodeId7 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId7->ExpandedText = "nsu=http://test.org/UA/Data/;ns=2;i=10853";
printf("%s\n", $NodeId7);


// When neither namespace URI nor namespace index are given, the node ID is assumed to
be in namespace
// with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by OPC UA
standard. There are
// many standard nodes that live in this reserved namespace, but no nodes specific to
your servers will
// be in the reserved namespace, and hence the need to specify the namespace with
server-specific nodes.
$NodeId8 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId8->ExpandedText = "i=2254";
```

```
printf("%s\n", $NodeId8);


// If you attempt to pass in a string that does not conform to the syntax rules,
// a UANodeIdFormatException is thrown.
$NodeId9 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
try
{
    $NodeId9->ExpandedText = "nsu=http://test.org/UA/Data/;i=notAnInteger";
    printf("%s\n", $NodeId9);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}


// There is a parser object that can be used to parse the expanded texts of node IDs.
$NodeIdParser10 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser");
$NodeId10 = $NodeIdParser10->Parse("nsu=http://test.org/UA/Data/;i=10853", False);
printf("%s\n", $NodeId10);


// The parser can be used if you want to parse the expanded text of the node ID but do
not want
// exceptions be thrown.
$NodeId11 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeIdParser11 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser");
$StringParsingError = $NodeIdParser11-
>TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger", False, $NodeId11);
if (is_null($StringParsingError))
    printf("%s\n", $NodeId11);
else
    printf("*** Failure: %s\n", $StringParsingError->Message);


// You can also use the parser if you have node IDs where you want the default
namespace be different
// from the standard "http://opcfoundation.org/UA/".
$NodeIdParser12 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser");
$NodeIdParser12->DefaultNamespaceUriString = "http://test.org/UA/Data/";
$NodeId12 = $NodeIdParser12->Parse("i=10853", False);
printf("%s\n", $NodeId12);


// You can create a "null" node ID. Such node ID does not actually identify any valid
node in OPC UA, but
// is useful as a placeholder or as a starting point for further modifications of its
properties.
$NodeId14 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
printf("%s\n", $NodeId14);


// Properties of a node ID can be modified individually. The advantage of this approach
is that you do
// not have to care about syntax of the node ID expanded text.
$NodeId15 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
```

```
$NodeId15->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId15->Identifier = 10853;
printf("%s\n", $NodeId15);


// If you know the type of the identifier upfront, it is safer to use typed properties
that correspond
// to specific types of identifier. Here, with an integer identifier.
$NodeId17 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId17->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId17->NumericIdentifier = 10853;
printf("%s\n", $NodeId17);


// Similarly, with a string identifier.
$NodeId18 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId18->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId18->StringIdentifier = "someIdentifier";
printf("%s\n", $NodeId18);


// If you have GUID in its string form, the node ID object can parse it for you.
$NodeId20 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId20->NamespaceUriString = "http://test.org/UA/Data/";
$NodeId20->GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10";
printf("%s\n", $NodeId20);


// And, with an opaque identifier.
$NodeId21 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId21->NamespaceUriString = "http://test.org/UA/Data/";
$OpaqueIdentifier21[0] = 0x00;
$OpaqueIdentifier21[1] = 0xFF;
$NodeId21->OpaqueIdentifier = $OpaqueIdentifier21;
printf("%s\n", $NodeId21);


// We have built-in a list of all standard nodes specified by OPC UA. You can simply
refer to these node IDs in your code.
// You can refer to any standard node using its name (in a string form).
// Note that assigning a non-existing standard name is not allowed, and throws
ArgumentException.
$NodeId26 = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$NodeId26->StandardName = "TypesFolder";
printf("%s\n", $NodeId26);
// When the UANodeId equals to one of the standard nodes, it is output in the shortened
form - as the standard name only.


// When you browse for nodes in the OPC UA server, every returned node element contains
a node ID that
// you can use further.
$Client27 = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// Browse from the Server node.
$ServerNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
```

```
$ServerNodeId->StandardName = "Server";
$ServerNodeDescriptor = new COM("OpcLabs.EasyOpc.UA.UANodeDescriptor");
$ServerNodeDescriptor->NodeId = $ServerNodeId;
// Browse all References.
$ReferencesNodeId = new COM("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId");
$ReferencesNodeId->StandardName = "References";

$BrowseParameters = new COM("OpcLabs.EasyOpc.UA.UABrowseParameters");
$BrowseParameters->NodeClasses = UANodeClass_All;  // this is the default, anyway
$BrowseParameters->ReferenceTypeIds->Add($ReferencesNodeId);

try
{
    $NodeElements27 = $Client27->Browse($EndpointDescriptor, $ServerNodeDescriptor,
$BrowseParameters);
    if ($NodeElements27->Count != 0) {
        $NodeId27 = $NodeElements27[0]->NodeId;
        printf("%s\n", $NodeId27);
    }
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}
```

VB.NET

```
' This example shows different ways of constructing OPC UA node IDs.

Imports System
Imports OpcLabs.BaseLib
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.AddressSpace.Parsing
Imports OpcLabs.EasyOpc.UA.AddressSpace.Parsing.Extensions
Imports OpcLabs.EasyOpc.UA.AddressSpace.Standard

Namespace UADocExamples._UANodeId
    Friend Class _Construction
        Public Shared Sub Main1()

            ' A node ID can be specified in string form (so-called expanded text).
            ' The code below specifies a namespace URI (nsu=...), and an integer
identifier (i=...).
            Dim nodeId1 = New UANodeId("nsu=http://test.org/UA/Data/;i=10853")
            Console.WriteLine(nodeId1)


            ' Similarly, with a string identifier (s=...).
            Dim nodeId2 = New UANodeId("nsu=http://test.org/UA/Data/;s=someIdentifier")
            Console.WriteLine(nodeId2)


            ' Actually, "s=" can be omitted (not recommended, though)
            Dim nodeId3 = New UANodeId("nsu=http://test.org/UA/Data/;someIdentifier")
            Console.WriteLine(nodeId3)
```

```
            ' Similarly, with a GUID identifier (g=...)
            Dim nodeId4 = New UANodeId("nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-
4A06-9EF0-E52010D5CD10")
            Console.WriteLine(nodeId4)


            ' Similarly, with an opaque identifier (b=..., in Base64 encoding).
            Dim nodeId5 = New UANodeId("nsu=http://test.org/UA/Data/;b=AP8=")
            Console.WriteLine(nodeId5)


            ' Namespace index can be used instead of namespace URI. The server is
allowed to change the namespace
            ' indices between sessions, and for this reason, you should avoid the use
of namespace indices, and
            ' rather use the namespace URIs whenever possible.
            Dim nodeId6 = New UANodeId("ns=2;i=10853")
            Console.WriteLine(nodeId6)


            ' Namespace index can be also specified together with namespace URI. This
is still safe, but may be
            ' a bit quicker to perform, because the client can just verify the
namespace URI instead of looking
            ' it up.
            Dim nodeId7 = New UANodeId("nsu=http://test.org/UA/Data/;ns=2;i=10853")
            Console.WriteLine(nodeId7)


            ' When neither namespace URI nor namespace index are given, the node ID is
assumed to be in namespace
            ' with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by
OPC UA standard. There are
            ' many standard nodes that live in this reserved namespace, but no nodes
specific to your servers will
            ' be in the reserved namespace, and hence the need to specify the namespace
with server-specific nodes.
            Dim nodeId8 = New UANodeId("i=2254")
            Console.WriteLine(nodeId8)


            ' If you attempt to pass in a string that does not conform to the syntax
rules,
            ' a UANodeIdFormatException is thrown.
            Try
                Dim nodeId9 = New
UANodeId("nsu=http://test.org/UA/Data/;i=notAnInteger")
                Console.WriteLine(nodeId9)
            Catch nodeIdFormatException As UANodeIdFormatException
                Console.WriteLine(nodeIdFormatException.Message)
            End Try


            ' There is a parser object that can be used to parse the expanded textx of
node IDs.
```

```
            Dim nodeIdParser10 = New UANodeIdParser()
            Dim nodeId10 = nodeIdParser10.Parse("nsu=http://test.org/UA/Data/;i=10853")
            Console.WriteLine(nodeId10)


            ' The parser can be used if you want to parse the expanded text of the node
ID but do not want
            ' exceptions be thrown.
            Dim nodeIdParser11 = New UANodeIdParser()
            Dim nodeId11 As UANodeId = Nothing
            Dim stringParsingError As IStringParsingError =

nodeIdParser11.TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger", nodeId11)
            If stringParsingError Is Nothing Then
                Console.WriteLine(nodeId11)
            Else
                Console.WriteLine(stringParsingError.Message)
            End If


            ' You can also use the parser if you have node IDs where you want the
default namespace be different
            ' from the standard "http://opcfoundation.org/UA/".
            Dim nodeIdParser12 = New UANodeIdParser("http://test.org/UA/Data/")
            Dim nodeId12 = nodeIdParser12.Parse("i=10853")
            Console.WriteLine(nodeId12)


            ' The namespace URI string (or the namespace index, or both) and the
identifier can be passed to the
            ' constructor separately.
            Dim nodeId13 = New UANodeId("http://test.org/UA/Data/", 10853)
            Console.WriteLine(nodeId13)


            ' You can create a "null" node ID. Such node ID does not actually identify
any valid node in OPC UA, but
            ' is useful as a placeholder or as a starting point for further
modifications of its properties.
            Dim nodeId14 = New UANodeId()
            Console.WriteLine(nodeId14)


            ' Properties of a node ID can be modified individually. The advantage of
this approach is that you do
            ' not have to care about syntax of the node ID expanded text.
            Dim nodeId15 = New UANodeId()
            nodeId15.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId15.Identifier = 10853
            Console.WriteLine(nodeId15)


            ' The same as above, but using an object initializer list.
            Dim nodeId16 = New UANodeId With
            {
                .NamespaceUriString = "http://test.org/UA/Data/",
                .Identifier = 10853
```

```
            }
            Console.WriteLine(nodeId16)


            ' If you know the type of the identifier upfront, it is safer to use typed
properties that correspond
            ' to specific types of identifier. Here, with an integer identifier.
            Dim nodeId17 = New UANodeId()
            nodeId17.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId17.NumericIdentifier = 10853
            Console.WriteLine(nodeId17)


            ' Similarly, with a string identifier.
            Dim nodeId18 = New UANodeId()
            nodeId18.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId18.StringIdentifier = "someIdentifier"
            Console.WriteLine(nodeId18)


            ' Similarly, with a GUID identifier.
            Dim nodeId19 = New UANodeId()
            nodeId19.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId19.GuidIdentifier = Guid.Parse("BAEAF004-1E43-4A06-9EF0-
E52010D5CD10")
            Console.WriteLine(nodeId19)


            ' If you have GUID in its string form, the node ID object can parse it for
you.
            Dim nodeId20 = New UANodeId()
            nodeId20.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId20.GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10"
            Console.WriteLine(nodeId20)


            ' And, with an opaque identifier.
            Dim nodeId21 = New UANodeId()
            nodeId21.NamespaceUriString = "http://test.org/UA/Data/"
            nodeId21.OpaqueIdentifier = {&H0, &HFF}
            Console.WriteLine(nodeId21)


            ' Assigning an expanded text to a node ID parses the value being assigned
and sets all corresponding
            ' properties accordingly.
            Dim nodeId22 = New UANodeId()
            nodeId22.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"
            Console.WriteLine(nodeId22)


            ' There is an implicit conversion from a string (representing the expanded
text) to a node ID.
            ' You can therefore use the expanded text (string) in place of any node ID
object directly.
            Dim nodeId23 = "nsu=http://test.org/UA/Data/;i=10853"
            Console.WriteLine(nodeId23)
```

```
            ' There is a copy constructor as well, creating a clone of an existing node
ID.
            Dim nodeId24a = New UANodeId("nsu=http://test.org/UA/Data/;i=10853")
            Console.WriteLine(nodeId24a)
            Dim nodeId24b = New UANodeId(nodeId24a)
            Console.WriteLine(nodeId24b)


            ' We have provided static classes with properties that correspond to all
standard nodes specified by
            ' OPC UA. You can simply refer to these node IDs in your code.
            ' The class names are UADataTypeIds, UAMethodIds, UAObjectIds,
UAObjectTypeIds, UAReferenceTypeIds,
            ' UAVariableIds and UAVariableTypeIds.
            Dim nodeId25 = UAObjectIds.TypesFolder
            Console.WriteLine(nodeId25)


            ' You can also refer to any standard node using its name (in a string
form).
            ' Note that assigning a non-existing standard name is not allowed, and
throws ArgumentException.
            Dim nodeId26 = New UANodeId()
            nodeId26.StandardName = "TypesFolder"
            Console.WriteLine(nodeId26)


            ' When you browse for nodes in the OPC UA server, every returned node
element contains a node ID that
            ' you can use further.
            Dim client27 = New EasyUAClient()
            Dim nodeElementCollection27 = client27.Browse(
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                UAObjectIds.Server,
                New UABrowseParameters(UANodeClass.All,
{UAReferenceTypeIds.References}))
            Dim nodeId27 = nodeElementCollection27(0).NodeId
            Console.WriteLine(nodeId27)


            ' As above, but using a constructor that takes a node element as an input.
            Dim client28 = New EasyUAClient()
            Dim nodeElementCollection28 = client28.Browse(
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
                UAObjectIds.Server,
                New UABrowseParameters(UANodeClass.All,
{UAReferenceTypeIds.References}))
            Dim nodeId28 = New UANodeId(nodeElementCollection28(0))
            Console.WriteLine(nodeId28)


            ' Or, there is an explicit conversion from a node element as well.
            Dim client29 = New EasyUAClient()
            Dim nodeElementCollection29 = client29.Browse(
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
```

```
                UAObjectIds.Server,
                New UABrowseParameters(UANodeClass.All,
{UAReferenceTypeIds.References}))
            Dim nodeId29 = CType(nodeElementCollection29(0), UANodeId)
            Console.WriteLine(nodeId29)
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows different ways of constructing OPC UA node IDs.

Option Explicit

' A node ID specifies a namespace (either by an URI or by an index), and an identifier.
' The identifier can be numeric (an integer), string, GUID, or opaque.

Const UANodeClass_All = 255


' A node ID can be specified in string form (so-called expanded text).
' The code below specifies a namespace URI (nsu=...), and an integer identifier
(i=...).
' Assigning an expanded text to a node ID parses the value being assigned and sets all
corresponding
' properties accordingly.
Dim NodeId1: Set NodeId1 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId1.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"
WScript.Echo NodeId1


' Similarly, with a string identifier (s=...).
Dim NodeId2: Set NodeId2 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId2.ExpandedText = "nsu=http://test.org/UA/Data/;s=someIdentifier"
WScript.Echo NodeId2


' Actually, "s=" can be omitted (not recommended, though)
Dim NodeId3: Set NodeId3 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId3.ExpandedText = "nsu=http://test.org/UA/Data/;someIdentifier"
WScript.Echo NodeId3
' Notice that the output is normalized - the "s=" is added again.


' Similarly, with a GUID identifier (g=...)
Dim NodeId4: Set NodeId4 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId4.ExpandedText = "nsu=http://test.org/UA/Data/;g=BAEAF004-1E43-4A06-9EF0-
E52010D5CD10"
WScript.Echo NodeId4
' Notice that the output is normalized - uppercase letters in the GUI are converted to
lowercase, etc.


' Similarly, with an opaque identifier (b=..., in Base64 encoding).
Dim NodeId5: Set NodeId5 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId5.ExpandedText = "nsu=http://test.org/UA/Data/;b=AP8="
```

```
WScript.Echo NodeId5


' Namespace index can be used instead of namespace URI. The server is allowed to change
the namespace
' indices between sessions (except for namespace 0), and for this reason, you should
avoid the use of
' namespace indices, and rather use the namespace URIs whenever possible.
Dim NodeId6: Set NodeId6 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId6.ExpandedText = "ns=2;i=10853"
WScript.Echo NodeId6


' Namespace index can be also specified together with namespace URI. This is still
safe, but may be
' a bit quicker to perform, because the client can just verify the namespace URI
instead of looking
' it up.
Dim NodeId7: Set NodeId7 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId7.ExpandedText = "nsu=http://test.org/UA/Data/;ns=2;i=10853"
WScript.Echo NodeId7


' When neither namespace URI nor namespace index are given, the node ID is assumed to
be in namespace
' with index 0 and URI "http://opcfoundation.org/UA/", which is reserved by OPC UA
standard. There are
' many standard nodes that live in this reserved namespace, but no nodes specific to
your servers will
' be in the reserved namespace, and hence the need to specify the namespace with
server-specific nodes.
Dim NodeId8: Set NodeId8 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId8.ExpandedText = "i=2254"
WScript.Echo NodeId8


' If you attempt to pass in a string that does not conform to the syntax rules,
' a UANodeIdFormatException is thrown.
Dim NodeId9: Set NodeId9 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
On Error Resume Next
NodeId9.ExpandedText = "nsu=http://test.org/UA/Data/;i=notAnInteger"
If Err.Number = 0 Then
    WScript.Echo NodeId9
Else
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
End If
On Error Goto 0


' There is a parser object that can be used to parse the expanded texts of node IDs.
Dim NodeIdParser10: Set NodeIdParser10 =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser")
Dim NodeId10: Set NodeId10 =
NodeIdParser10.Parse("nsu=http://test.org/UA/Data/;i=10853", False)
WScript.Echo NodeId10
```

```
' The parser can be used if you want to parse the expanded text of the node ID but do
not want
' exceptions be thrown.
Dim NodeIdParser11: Set NodeIdParser11 =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser")
Dim NodeId11
Dim StringParsingError: Set StringParsingError =
NodeIdParser11.TryParse("nsu=http://test.org/UA/Data/;i=notAnInteger", False, NodeId11)
If StringParsingError Is Nothing Then
    WScript.Echo NodeId11
Else
    WScript.Echo "*** Failure: " & StringParsingError.Message
End If


' You can also use the parser if you have node IDs where you want the default namespace
be different
' from the standard "http://opcfoundation.org/UA/".
Dim NodeIdParser12: Set NodeIdParser12 =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.Parsing.UANodeIdParser")
NodeIdParser12.DefaultNamespaceUriString = "http://test.org/UA/Data/"
Dim NodeId12: Set NodeId12 = NodeIdParser12.Parse("i=10853", False)
WScript.Echo NodeId12


' You can create a "null" node ID. Such node ID does not actually identify any valid
node in OPC UA, but
' is useful as a placeholder or as a starting point for further modifications of its
properties.
Dim NodeId14: Set NodeId14 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
WScript.Echo NodeId14


' Properties of a node ID can be modified individually. The advantage of this approach
is that you do
' not have to care about syntax of the node ID expanded text.
Dim NodeId15: Set NodeId15 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId15.NamespaceUriString = "http://test.org/UA/Data/"
NodeId15.Identifier = 10853
WScript.Echo NodeId15


' If you know the type of the identifier upfront, it is safer to use typed properties
that correspond
' to specific types of identifier. Here, with an integer identifier.
Dim NodeId17: Set NodeId17 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId17.NamespaceUriString = "http://test.org/UA/Data/"
NodeId17.NumericIdentifier = 10853
WScript.Echo NodeId17


' Similarly, with a string identifier.
Dim NodeId18: Set NodeId18 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId18.NamespaceUriString = "http://test.org/UA/Data/"
NodeId18.StringIdentifier = "someIdentifier"
WScript.Echo NodeId18
```

```
' If you have GUID in its string form, the node ID object can parse it for you.
Dim NodeId20: Set NodeId20 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId20.NamespaceUriString = "http://test.org/UA/Data/"
NodeId20.GuidIdentifierString = "BAEAF004-1E43-4A06-9EF0-E52010D5CD10"
WScript.Echo NodeId20


' And, with an opaque identifier.
Dim NodeId21: Set NodeId21 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId21.NamespaceUriString = "http://test.org/UA/Data/"
NodeId21.OpaqueIdentifier = Array(&H00, &HFF)
WScript.Echo NodeId21


' We have built-in a list of all standard nodes specified by OPC UA. You can simply
refer to these node IDs in your code.
' You can refer to any standard node using its name (in a string form).
' Note that assigning a non-existing standard name is not allowed, and throws
ArgumentException.
Dim NodeId26: Set NodeId26 = CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
NodeId26.StandardName = "TypesFolder"
WScript.Echo NodeId26
' When the UANodeId equals to one of the standard nodes, it is output in the shortened
form - as the standard name only.


' When you browse for nodes in the OPC UA server, every returned node element contains
a node ID that
' you can use further.
Dim Client27: Set Client27 = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
Dim EndpointDescriptor: Set EndpointDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UAEndpointDescriptor")
EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
' Browse from the Server node.
Dim ServerNodeId: Set ServerNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ServerNodeId.StandardName = "Server"
Dim ServerNodeDescriptor: Set ServerNodeDescriptor =
CreateObject("OpcLabs.EasyOpc.UA.UANodeDescriptor")
ServerNodeDescriptor.NodeId = ServerNodeId
' Browse all References.
Dim ReferencesNodeId: Set ReferencesNodeId =
CreateObject("OpcLabs.EasyOpc.UA.AddressSpace.UANodeId")
ReferencesNodeId.StandardName = "References"
'
Dim BrowseParameters: Set BrowseParameters =
CreateObject("OpcLabs.EasyOpc.UA.UABrowseParameters")
BrowseParameters.NodeClasses = UANodeClass_All  ' this is the default, anyway
BrowseParameters.ReferenceTypeIds.Add ReferencesNodeId
'
On Error Resume Next
Dim NodeElementCollection27: Set NodeElementCollection27 = Client27.Browse( _
    EndpointDescriptor, ServerNodeDescriptor, BrowseParameters)
If Err.Number = 0 Then
    If NodeElementCollection27.Count <> 0 Then
        Dim NodeId27: Set NodeId27 = NodeElementCollection27(0).NodeId
```

```
        WScript.Echo NodeId27
    End If
Else
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
End If
On Error Goto 0
```

13.2.13.14 Examples - OPC Unified Architecture - Discover servers from GDS, flat

C#

```
// This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
// The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class DiscoverGlobalServers
    {
        public static void Main1()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection =
client.DiscoverGlobalServers("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI string: {0}",
```

```
discoveryElement.DiscoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
            }
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
// The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

class procedure DiscoverGlobalServers.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of application elements
  try
    DiscoveryElements := Client.DiscoverGlobalServers('opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn;
    WriteLn('Server name: ', DiscoveryElement.ServerName);
    WriteLn('Discovery URI string: ', DiscoveryElement.DiscoveryUriString);
    WriteLn('Server capabilities: ', DiscoveryElement.ServerCapabilities.ToString);
  end;
end;
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
' The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.
```

```
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverGlobalServers
        Public Shared Sub Main1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection =
client.DiscoverGlobalServers("opc.tcp://opcua.demo-
this.com:58810/GlobalDiscoveryServer")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString)
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

13.2.13.15 Examples - OPC Unified Architecture - Discover servers from GDS, hierarchical

C#

```
// This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
// The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
```

```
    partial class DiscoverGlobalServers
    {
        public static void Hierarchical()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverGlobalServers(
                    "opc.tcp://opcua.demo-this.com:58810/GlobalDiscoveryServer",
                    flat:false);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI strings:");
                foreach (string discoveryUriString in
discoveryElement.DiscoveryUriStrings)
                    Console.WriteLine("  {0}", discoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString);
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers from the Global
Discovery Server (GDS).
' The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverGlobalServers
        Public Shared Sub Hierarchical()
            ' Instantiate the client object
```

```
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverGlobalServers( _
                    "opc.tcp://opcua.demo-this.com:58810/GlobalDiscoveryServer",
                    flat:=False)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI strings:")
                For Each discoveryUriString As String In
discoveryElement.DiscoveryUriStrings
                    Console.WriteLine("  {0}", discoveryUriString)
                Next discoveryUriString
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString)
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

13.2.13.16 Examples - OPC Unified Architecture - Discover servers on a host

C#

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on a given machine.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class DiscoverLocalServers
    {
        public static void Overload1()
        {
```

```
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of server elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverLocalServers("opcua.demo-
this.com");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
                Console.WriteLine("discoveryElementCollection[\"
{0}\"].ApplicationUriString: {1}",
                    discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString);

            // Example output:
            // discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
            // discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        }
    }
}
```

VB.NET

```
' This example shows how to obtain application URLs of all OPC Unified Architecture
servers on a given machine.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class DiscoverLocalServers
        Public Shared Sub Overload1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of server elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverLocalServers("opcua.demo-
this.com")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
```

```
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine("discoveryElementCollection[""
{0}""].ApplicationUriString: {1}", _
                                  discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString)
            Next discoveryElement

            ' Example output:
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
            'discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "DiscoverLocalServers.h"

namespace _EasyUAClient
{
    void DiscoverLocalServers::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Obtain collection of server elements
            _UADiscoveryElementCollectionPtr DiscoveryElementsPtr = ClientPtr-
>DiscoverLocalServers(L"opcua.demo-this.com");

            // Display results
            IEnumVARIANTPtr EnumDiscoveryElementPtr = DiscoveryElementsPtr-
>GetEnumerator();
            _variant_t vDiscoveryElement;
            while (EnumDiscoveryElementPtr->Next(1, &vDiscoveryElement, NULL) == S_OK)
            {
                _UADiscoveryElementPtr DiscoveryElementPtr(vDiscoveryElement);
                _tprintf(_T("DiscoveryElementCollection[\"%s\"]: "),
(LPCTSTR)CW2CT(DiscoveryElementPtr->DiscoveryUriString));
                _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(DiscoveryElementPtr-
>ApplicationUriString));
                vDiscoveryElement.Clear();
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
```

```
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

class procedure DiscoverLocalServers.Main;
var
  Client: EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of server elements
  DiscoveryElements := Client.DiscoverLocalServers('opcua.demo-this.com');

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn(
      'DiscoveryElements["',
      DiscoveryElement.DiscoveryUriString,
      '".ApplicationUriString: ',
      DiscoveryElement.ApplicationUriString);
  end;
end;
```

Object Pascal

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

class procedure DiscoverLocalServers.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of server elements
  try
    DiscoveryElements := Client.DiscoverLocalServers('opcua.demo-this.com');
```

```
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn(
      'DiscoveryElements["',
      DiscoveryElement.DiscoveryUriString,
      '".ApplicationUriString: ',
      DiscoveryElement.ApplicationUriString);
  end;
end;
```

PHP

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain collection of server elements
try
{
    $DiscoveryElementCollection = $Client->DiscoverLocalServers("opcua.demo-this.com");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}

// Display results
foreach ($DiscoveryElementCollection as $DiscoveryElement)
{
    printf("DiscoveryElementCollection[\"%s\"].ApplicationUriString: %s\n",
        $DiscoveryElement->DiscoveryUriString, $DiscoveryElement-
>ApplicationUriString);
}
```

Python

```
# This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')
```

```
# Obtain collection of server elements
discoveryElementCollection = client.DiscoverLocalServers('opcua.demo-this.com')

# Display results
for discoveryElement in discoveryElementCollection:
    print('DiscoveryElementCollection["', discoveryElement.DiscoveryUriString,
'"].ApplicationUriString: ',
          discoveryElement.ApplicationUriString)
```

Visual Basic (VB 6.)

```
Rem This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

Private Sub DiscoverLocalServers_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Obtain collection of server elements
    On Error Resume Next
    Dim DiscoveryElementCollection As UADiscoveryElementCollection
    Set DiscoveryElementCollection = Client.DiscoverLocalServers("opcua.demo-this.com")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    Dim DiscoveryElement As UADiscoveryElement: For Each DiscoveryElement In
DiscoveryElementCollection
        OutputText = OutputText & "DiscoveryElementCollection[""" &
DiscoveryElement.DiscoveryUriString & """].ApplicationUriString: " & _
            DiscoveryElement.applicationUriString & vbCrLf
    Next
End Sub
```

VBScript

```
Rem This example shows how to obtain application URLs of all OPC Unified Architecture
servers on the specified host.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain collection of server elements
On Error Resume Next
Dim DiscoveryElementCollection: Set DiscoveryElementCollection =
Client.DiscoverLocalServers("opcua.demo-this.com")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
```

```
End If
On Error Goto 0

' Display results
Dim DiscoveryElement: For Each DiscoveryElement In DiscoveryElementCollection
    WScript.Echo "DiscoveryElementCollection[""" & DiscoveryElement.DiscoveryUriString
& """].ApplicationUriString: " & _
        DiscoveryElement.ApplicationUriString
Next
```

13.2.13.17 Examples - OPC Unified Architecture - Discover servers on a network, flat

C#

```
// This example shows how to obtain information about OPC UA servers available on the
network.
// The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class DiscoverNetworkServers
    {
        public static void Main1()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
```

```
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString);
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString);
            }
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain information about OPC UA servers available on the
network.

class procedure DiscoverNetworkServers.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of application elements
  try
    DiscoveryElements := Client.DiscoverNetworkServers(Unassigned, 'opcua.demo-
this.com');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn;
    WriteLn('Server name: ', DiscoveryElement.ServerName);
    WriteLn('Discovery URI string: ', DiscoveryElement.DiscoveryUriString);
    WriteLn('Server capabilities: ', DiscoveryElement.ServerCapabilities.ToString);
  end;
end;
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers available on the
network.
' The result is flat, i.e. each discovery URL is returned in separate element, with
possible repetition of the servers.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverNetworkServers
        Public Shared Sub Main1()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI string: {0}",
discoveryElement.DiscoveryUriString)
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
                Console.WriteLine("Application URI string: {0}",
discoveryElement.ApplicationUriString)
                Console.WriteLine("Product URI string: {0}",
discoveryElement.ProductUriString)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to obtain information about OPC UA servers available on the
network.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "DiscoverNetworkServers.h"

namespace _EasyUAClient
{
    void DiscoverNetworkServers::Main()
    {
        // Initialize the COM library
```

```
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            _StringCollectionPtr ServerElementFilterPtr(__uuidof(StringCollection));
            _variant_t vServerElementFilter(ServerElementFilterPtr.GetInterfacePtr());

            // Obtain collection of application elements
            _UADiscoveryElementCollectionPtr DiscoveryElementsPtr =
                ClientPtr->DiscoverNetworkServers(vServerElementFilter, L"opcua.demo-
this.com");

            // Display results
            IEnumVARIANTPtr EnumDiscoveryElementPtr = DiscoveryElementsPtr-
>GetEnumerator();
            _variant_t vDiscoveryElement;
            while (EnumDiscoveryElementPtr->Next(1, &vDiscoveryElement, NULL) == S_OK)
            {
                _UADiscoveryElementPtr DiscoveryElementPtr(vDiscoveryElement);
                _tprintf(_T("\n"));
                _tprintf(_T("Application type: %d\n"), DiscoveryElementPtr-
>ApplicationType);
                _tprintf(_T("Server name: %s\n"), (LPCTSTR)CW2CT(DiscoveryElementPtr-
>ServerName));
                _tprintf(_T("Discovery URI string: %s\n"),
(LPCTSTR)CW2CT(DiscoveryElementPtr->DiscoveryUriString));
                _tprintf(_T("Server capabilities: "));
                IEnumVARIANTPtr EnumServerCapabilityPtr = DiscoveryElementPtr-
>ServerCapabilities->GetEnumerator();
                _variant_t vServerCapability = NULL;
                while (EnumServerCapabilityPtr->Next(1, &vServerCapability, NULL) ==
S_OK)
                    _tprintf(_T("%s "), (LPCTSTR)CW2CT(_bstr_t(vServerCapability)));
                _tprintf(_T("\n"));
                vDiscoveryElement.Clear();
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

VBScript

```
Rem This example shows how to obtain information about OPC UA servers available on the
network.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain collection of application elements
On Error Resume Next
Dim DiscoveryElementCollection: Set DiscoveryElementCollection = _
    Client.DiscoverNetworkServers(Nothing, "opcua.demo-this.com")
```

```
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim DiscoveryElement: For Each DiscoveryElement In DiscoveryElementCollection
    WScript.Echo
    WScript.Echo "Server name: " & DiscoveryElement.ServerName
    WScript.Echo "Discovery URI string: " & DiscoveryElement.DiscoveryUriString
    Dim s: s = ""
    Dim serverCapability: For Each serverCapability In
DiscoveryElement.ServerCapabilities
        s = s & serverCapability & " "
    Next
    WScript.Echo "Server capabilities: " & s
Next
```

13.2.13.18 Examples - OPC Unified Architecture - Discover servers on a network, hierarchical

C#

```
// This example shows how to obtain information about OPC UA servers available on the
network.
// The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class DiscoverNetworkServers
    {
        public static void Hierarchical()
        {
            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com", flat:false);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
```

```
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
            {
                Console.WriteLine();
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName);
                Console.WriteLine("Discovery URI strings:");
                foreach (string discoveryUriString in
discoveryElement.DiscoveryUriStrings)
                    Console.WriteLine("  {0}", discoveryUriString);
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain information about OPC UA servers available on the
network.
' The result is hierarchical, i.e. each server is returned in one element, and the
element contains all its discovery URLs.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class DiscoverNetworkServers
        Public Shared Sub Hierarchical()
            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection = client.DiscoverNetworkServers("opcua.demo-
this.com", flat:=False)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine()
                Console.WriteLine("Server name: {0}", discoveryElement.ServerName)
                Console.WriteLine("Discovery URI strings:")
                For Each discoveryUriString As String In
discoveryElement.DiscoveryUriStrings
                    Console.WriteLine("  {0}", discoveryUriString)
                Next discoveryUriString
```

```
                Console.WriteLine("Server capabilities: {0}",
discoveryElement.ServerCapabilities)
            Next discoveryElement
        End Sub
    End Class
End Namespace
```

13.2.13.19 Examples - OPC Unified Architecture - Event logging

C#

```
// This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

using System;
using OpcLabs.BaseLib.Instrumentation;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class LogEntry
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Hook static events
            EasyUAClient.LogEntry += EasyUAClientOnLogEntry;

            try
            {
                // Do something - invoke an OPC read, to trigger some loggable entries.
                var client = new EasyUAClient();
                try
                {
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                    return;
                }

                Console.WriteLine("Processing log entry events for 1 minute...");
                System.Threading.Thread.Sleep(60 * 1000);
```

```
                Console.WriteLine("Done.");
            }
            finally
            {
                // Unhook static events
                EasyUAClient.LogEntry -= EasyUAClientOnLogEntry;
            }
        }

        // Event handler for the LogEntry event. It simply prints out the event.
        private static void EasyUAClientOnLogEntry(object sender, LogEntryEventArgs
logEntryEventArgs)
        {
            Console.WriteLine(logEntryEventArgs);
        }
    }
}
```

VB.NET

```
' This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

Imports OpcLabs.BaseLib.Instrumentation
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class LogEntry
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Hook static events
            AddHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry

            Try
                ' Do something - invoke an OPC read, to trigger some loggable entries.
                Dim client = New EasyUAClient()
                Try
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Exit Sub
                End Try

                Console.WriteLine("Processing log entry events for 1 minute...")
                Threading.Thread.Sleep(60 * 1000)

                Console.WriteLine("Done.")
            Finally
```

```
                ' Unhook static events
                RemoveHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry
            End Try

        End Sub


        ' Event handler for the LogEntry event. It simply prints out the event.
        Private Shared Sub EasyUAClientOnLogEntry(ByVal sender As Object, ByVal
logEntryEventArgs As LogEntryEventArgs)
            Console.WriteLine(logEntryEventArgs)
        End Sub
    End Class
End Namespace
```

C++

```
// This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include "LogEntry.h"

namespace _EasyUAClientConfiguration
{
    // CEasyUAClientConfigurationEvents

    class CEasyUAClientConfigurationEvents : public IDispEventImpl<1,
CEasyUAClientConfigurationEvents>
    {
    public:
    BEGIN_SINK_MAP(CEasyUAClientConfigurationEvents)
        // Event handlers must have the __stdcall calling convention
        SINK_ENTRY(1, 1 /*DISPID_EASYUACLIENTCONFIGURATIONEVENTS_LOGENTRY*/,
&CEasyUAClientConfigurationEvents::LogEntry)
    END_SINK_MAP()

    public:
        // Event handler for the LogEntry event. It simply prints out the event.
        STDMETHOD(LogEntry)(VARIANT varSender, _LogEntryEventArgs* pEventArgs)
        {
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(pEventArgs->ToString));
            return S_OK;
        }
    };



    void LogEntry::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // The configuration object allows access to static behavior - here, the
shared LogEntry event.
            _EasyUAClientConfigurationPtr
```

```
ClientConfigurationPtr(__uuidof(EasyUAClientConfiguration));

            // Hook events
            CEasyUAClientConfigurationEvents* pClientConfigurationEvents = new
CEasyUAClientConfigurationEvents();
            AtlGetObjectSourceInterface(ClientConfigurationPtr,
&pClientConfigurationEvents->m_libid,
                &pClientConfigurationEvents->m_iid,
                &pClientConfigurationEvents->m_wMajorVerNum,
&pClientConfigurationEvents->m_wMinorVerNum);
            pClientConfigurationEvents->m_iid =
_uuidof(DEasyUAClientConfigurationEvents);
            pClientConfigurationEvents->DispEventAdvise(ClientConfigurationPtr,
&pClientConfigurationEvents->m_iid);

            // Do something - invoke an OPC read, to trigger some loggable entries.
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));
            ClientPtr->ReadValue(L"http://opcua.demo-this.com:51211/UA/SampleServer",
L"nsu=http://test.org/UA/Data/;i=10853");

            _tprintf(_T("Processing log entry events for 1 minute...\n"));
            Sleep(60*1000);

            // Unhook events
            pClientConfigurationEvents->DispEventUnadvise(ClientConfigurationPtr,
&pClientConfigurationEvents->m_iid);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example demonstrates the loggable entries originating in the OPC-UA
// client engine and the EasyUAClient component.

type
  TClientConfigurationEventHandlers = class
    procedure OnLogEntry(
      Sender: TObject;
      sender0: OleVariant;
      eventArgs: _LogEntryEventArgs);
  end;

// Event handler for the LogEntry event. It simply prints out the event.
procedure TClientConfigurationEventHandlers.OnLogEntry(
  Sender: TObject;
  sender0: OleVariant;
  eventArgs: _LogEntryEventArgs);
begin
    WriteLn(eventArgs.ToString);
end;

class procedure LogEntry.Main;
var
  Client: EasyUAClient;
```

```
  EvsClientConfiguration: TEvsEasyUAClientConfiguration;
  ClientConfiguration: EasyUAClientConfiguration;
  ClientConfigurationEventHandlers: TClientConfigurationEventHandlers;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior - here, the
  // shared LogEntry event.
  EvsClientConfiguration := TEvsEasyUAClientConfiguration.Create(nil);
  ClientConfiguration := EvsClientConfiguration.ComServer;
  ClientConfigurationEventHandlers := TClientConfigurationEventHandlers.Create;
  EvsClientConfiguration.OnLogEntry := @ClientConfigurationEventHandlers.OnLogEntry;

  // Do something - invoke an OPC read, to trigger some loggable entries.
  Client := CoEasyUAClient.Create;
  Value := Client.ReadValue(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853');

  WriteLn('Processing log entry events for 1 minute...');
  PumpSleep(60*1000);
end;
```

Object Pascal

```
// This example demonstrates the loggable entries originating in the OPC-UA
// client engine and the EasyUAClient component.

type
  TClientConfigurationEventHandlers130 = class
    procedure OnLogEntry(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _LogEntryEventArgs);
  end;

// Event handler for the LogEntry event. It simply prints out the event.
procedure TClientConfigurationEventHandlers130.OnLogEntry(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _LogEntryEventArgs);
begin
    WriteLn(eventArgs.ToString);
end;

class procedure LogEntry.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  ClientConfigurationEventHandlers: TClientConfigurationEventHandlers130;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior - here, the
  // shared LogEntry event.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfigurationEventHandlers := TClientConfigurationEventHandlers130.Create;
  ClientConfiguration.OnLogEntry := ClientConfigurationEventHandlers.OnLogEntry;
  ClientConfiguration.Connect;
```

```
  // Do something - invoke an OPC read, to trigger some loggable entries.
  Client := CoEasyUAClient.Create;
  try
    Value := Client.ReadValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      //Exit;
    end;
  end;

  WriteLn('Processing log entry events for 1 minute...');
  PumpSleep(60*1000);

  WriteLn('Finished.');
  FreeAndNil(ClientConfiguration);
  FreeAndNil(ClientConfigurationEventHandlers);
end;
```

PHP

```
// This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

class ClientConfigurationEvents {
    // Event handler for the LogEntry event. It simply prints out the event.
    function LogEntry($Sender, $E)
    {
    printf("%s\n", $E);
    }
}



// The configuration object allows access to static behavior - here, the shared
LogEntry event.
$ClientConfiguration = new COM("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration");
$ClientConfigurationEvents = new ClientConfigurationEvents();
com_event_sink($ClientConfiguration, $ClientConfigurationEvents,
"DEasyUAClientConfigurationEvents");

// Do something - invoke an OPC read, to trigger some loggable entries.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}
```

```
printf("Processing log entry events for 1 minute...");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

Visual Basic (VB 6.)

```
REM This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

' The configuration object allows access to static behavior - here, the shared LogEntry
event.
'Public WithEvents ClientConfiguration1 As EasyUAClientConfiguration

Private Sub LogEntry_Main_Command_Click()
    OutputText = ""

    Set ClientConfiguration1 = New EasyUAClientConfiguration

    ' Do something - invoke an OPC read, to trigger some loggable entries.
    Dim Client As New EasyUAClient
    On Error Resume Next
    Dim value As Variant
    value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    OutputText = OutputText & "Processing log entry events for 1 minute..." & vbCrLf
    Pause 60000

    Set ClientConfiguration1 = Nothing
    OutputText = OutputText & "Finished..." & vbCrLf
End Sub

' Event handler for the LogEntry event. It simply prints out the event.
Private Sub ClientConfiguration1_LogEntry(ByVal sender As Variant, ByVal eventArgs As
OpcLabs_BaseLib.LogEntryEventArgs)
    OutputText = OutputText & eventArgs & vbCrLf
End Sub
```

VBScript

```
Rem This example demonstrates the loggable entries originating in the OPC-UA client
engine and the EasyUAClient component.

Option Explicit

' The configuration object allows access to static behavior - here, the shared LogEntry
event.
Dim ClientConfiguration: Set ClientConfiguration =
CreateObject("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration")
WScript.ConnectObject ClientConfiguration, "ClientConfiguration_"
```

```
' Do something - invoke an OPC read, to trigger some loggable entries.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

WScript.Echo "Processing log entry events for 1 minute..."
WScript.Sleep 60*1000



' Event handler for the LogEntry event. It simply prints out the event.
Sub ClientConfiguration_LogEntry(Sender, e)
    WScript.Echo e
End Sub
```

13.2.13.20 Examples - OPC Unified Architecture - Event pull of data change notifications

C#

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class PullDataChangeNotification
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyUAClient { PullDataChangeNotificationQueueCapacity =
1000 };
```

```
            Console.WriteLine("Subscribing...");
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000);

            Console.WriteLine("Processing data change events for 1 minute...");
            int endTick = Environment.TickCount + 60 * 1000;
            do
            {
                EasyUADataChangeNotificationEventArgs eventArgs =
client.PullDataChangeNotification(2 * 1000);
                if (eventArgs != null)
                    // Handle the notification event
                    Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class PullDataChangeNotification
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            client.PullDataChangeNotificationQueueCapacity = 1000

            Console.WriteLine("Subscribing...")
            client.SubscribeDataChange(
                endpointDescriptor,
                "nsu=http://test.org/UA/Data/;i=10853",
                1000)    ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

            Console.WriteLine("Processing data change events for 1 minute...")
            Dim endTick As Integer = Environment.TickCount + 60 * 1000
            Do
                Dim eventArgs As EasyUADataChangeNotificationEventArgs =
client.PullDataChangeNotification(2 * 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
```

```
                End If
            Loop While Environment.TickCount < endTick
        End Sub
    End Class
End Namespace
```

Free Pascal

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

class procedure PullDataChangeNotification.Main;
var
  Client: EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUADataChangeNotificationEventArgs;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullDataChangeNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing data change events for 1 minute...');
  EndTick := GetTickCount + 60*1000;
  while GetTickCount < EndTick do
  begin
    EventArgs := Client.PullDataChangeNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;
```

JScript

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

var Client = new ActiveXObject("OpcLabs.EasyOpc.UA.EasyUAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullDataChangeNotificationQueueCapacity = 1000;

WScript.Echo("Subscribing...");
Client.SubscribeDataChange("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000);
```

```
WScript.Echo("Processing data change events for 1 minute...");
var endTime = new Date().getTime() + 60*1000
do {
    var EventArgs = Client.PullDataChangeNotification(2*1000);
    if (EventArgs !== null) {
        //  Handle the notification event
        WScript.Echo(EventArgs);
    }
} while(new Date().getTime() < endTime);
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

class procedure PullDataChangeNotification.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  EndTick: Cardinal;
  EventArgs: _EasyUADataChangeNotificationEventArgs;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;
  // In order to use event pull, you must set a non-zero queue capacity upfront.
  Client.PullDataChangeNotificationQueueCapacity := 1000;

  WriteLn('Subscribing...');
  Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing data change events for 1 minute...');
  EndTick := Ticks + 60*1000;
  while Ticks < EndTick do
  begin
    EventArgs := Client.PullDataChangeNotification(2*1000);
    if EventArgs <> nil then
      // Handle the notification event
      WriteLn(EventArgs.ToString);
  end;

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Finished.');
end;
```

PHP

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
// In order to use event pull, you must set a non-zero queue capacity upfront.
$Client->PullDataChangeNotificationQueueCapacity = 1000;
```

```
print "Subscribing...\n";
$Client->SubscribeDataChange("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000);

print "Processing data change events for 1 minute...\n";
$endTime = time() + 60;
do {
    $EventArgs = $Client->PullDataChangeNotification(2*1000);
    if (!is_null($EventArgs)) {
        // Handle the notification event
        print $EventArgs->ToString();
        print "\n";
    }
} while (time() < $endTime);
```

PowerScript

```
// This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")
// In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullDataChangeNotificationQueueCapacity = 1000

mle_outputtext.Text = mle_outputtext.Text + "Subscribing data changes..." + "~r~n"
client.SubscribeDataChange("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000)

mle_outputtext.Text = mle_outputtext.Text + "Processing data change events for 1
minute..." + "~r~n"

Time endTime = RelativeTime(Now(), 60)
DO
    OLEObject eventArgs
    eventArgs = client.PullDataChangeNotification(2*1000)
    IF NOT IsNull(eventArgs) THEN
        // Handle the notification event
        mle_outputtext.Text = mle_outputtext.Text + eventArgs.DisplayString + "~r~n"
    END IF
LOOP WHILE Now() < endTime

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

import time
import win32com.client
```

```
# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')
# In order to use event pull, you must set a non-zero queue capacity upfront.
client.PullDataChangeNotificationQueueCapacity = 1000

print('Subscribing...')
client.SubscribeDataChange('http://opcua.demo-this.com:51211/UA/SampleServer',
'nsu=http://test.org/UA/Data/;i=10853', 1000)

print('Processing data change events for 1 minute...')
endTime = time.time() + 60
while time.time() < endTime:
    eventArgs = client.PullDataChangeNotification(2*1000)
    if eventArgs is not None:
        # Handle the notification event
        print(eventArgs)
```

VBScript

```
Rem This example shows how to subscribe to changes of a single monitored item, pull
events, and display each change.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
' In order to use event pull, you must set a non-zero queue capacity upfront.
Client.PullDataChangeNotificationQueueCapacity = 1000

WScript.Echo "Subscribing..."
Client.SubscribeDataChange "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000

WScript.Echo "Processing data change events for 1 minute..."
Dim endTime: endTime = Now() + 60*(1/24/60/60)
Do
    Dim EventArgs: Set EventArgs = Client.PullDataChangeNotification(2*1000)
    If Not (EventArgs Is Nothing) Then
        ' Handle the notification event
        WScript.Echo EventArgs
    End If
Loop While Now() < endTime
```

13.2.13.21 Examples - OPC Unified Architecture - Find applications and their endpoints

C#

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

using System;
```

```
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Discovery;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class FindLocalApplications
    {
        public static void Main1()
        {

            string[] discoveryUriStrings =
            {
                "opc.tcp://opcua.demo-this.com:4840/UADiscovery",
                "http://opcua.demo-this.com/UADiscovery/Default.svc",
                "http://opcua.demo-this.com:52601/UADiscovery"
            };

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain collection of application elements
            UADiscoveryElementCollection discoveryElementCollection;
            try
            {
                discoveryElementCollection =
client.FindLocalApplications(discoveryUriStrings, UAApplicationTypes.Server);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UADiscoveryElement discoveryElement in discoveryElementCollection)
                Console.WriteLine("discoveryElementCollection[\"
{0}\"].ApplicationUriString: {1}",
                    discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString);

            // Example output:
            //discoveryElementCollection["http://opcua.demo-
this.com:62543/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            //discoveryElementCollection["opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            //discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            //discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            //discoveryElementCollection["https://opcua.demo-
this.com:51212/UA/SampleServer/"].ApplicationUriString: urn:opcua.demo-this.com:UA
```

```
Sample Server
            //discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

class procedure FindLocalApplications.Main;
const
  UAApplicationTypes_Server = 1;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Count: Cardinal;
  Element: OleVariant;
  DiscoveryElement: _UADiscoveryElement;
  DiscoveryElementEnumerator: IEnumVariant;
  DiscoveryElements: _UADiscoveryElementCollection;
  DiscoveryUriStrings: OleVariant;
begin
  DiscoveryUriStrings := VarArrayCreate([0, 2], varVariant);
  DiscoveryUriStrings[0] := 'opc.tcp://opcua.demo-this.com:4840/UADiscovery';
  DiscoveryUriStrings[1] := 'http://opcua.demo-this.com/UADiscovery/Default.svc';
  DiscoveryUriStrings[2] := 'http://opcua.demo-this.com:52601/UADiscovery';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain collection of application elements
  try
    DiscoveryElements := Client.FindLocalApplications(DiscoveryUriStrings,
UAApplicationTypes_Server);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  DiscoveryElementEnumerator := DiscoveryElements.GetEnumerator;
  while (DiscoveryElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    DiscoveryElement := IUnknown(Element) as _UADiscoveryElement;
    WriteLn(
      'DiscoveryElements["',
      DiscoveryElement.DiscoveryUriString,
      '".ApplicationUriString: ',
      DiscoveryElement.ApplicationUriString);
  end;
end;
```

VB.NET

```
' This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Discovery
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class FindLocalApplications
        Public Shared Sub Main1()
            Dim discoveryUriStrings() As String =
            { _
                "opc.tcp://opcua.demo-this.com:4840/UADiscovery", _
                "http://opcua.demo-this.com/UADiscovery/Default.svc", _
                "http://opcua.demo-this.com:52601/UADiscovery" _
            }

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain collection of application elements
            Dim discoveryElementCollection As UADiscoveryElementCollection
            Try
                discoveryElementCollection =
client.FindLocalApplications(discoveryUriStrings, UAApplicationTypes.Server)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each discoveryElement As UADiscoveryElement In
discoveryElementCollection
                Console.WriteLine("discoveryElementCollection[""
{0}""].ApplicationUriString: {1}", _
                    discoveryElement.DiscoveryUriString,
discoveryElement.ApplicationUriString)
            Next discoveryElement

            ' Example output:
            'discoveryElementCollection["http://opcua.demo-
this.com:62543/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:62544/Quickstarts/AlarmConditionServer"].ApplicationUriString: urn:opcua.demo-
this.com:Quickstart Alarm Condition Server
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            'discoveryElementCollection["http://opcua.demo-
this.com:51211/UA/SampleServer"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            'discoveryElementCollection["https://opcua.demo-
```

```
this.com:51212/UA/SampleServer/"].ApplicationUriString: urn:opcua.demo-this.com:UA
Sample Server
            'discoveryElementCollection["opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"].ApplicationUriString: urn:Test-PC:UA Sample Server
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain application URLs of all OPC Unified Architecture
servers, using specified discovery URI strings.

Option Explicit

Const UAApplicationTypes_Server = 1

Dim discoveryUriStrings(2)
discoveryUriStrings(0) = "opc.tcp://opcua.demo-this.com:4840/UADiscovery"
discoveryUriStrings(1) = "http://opcua.demo-this.com/UADiscovery/Default.svc"
discoveryUriStrings(2) = "http://opcua.demo-this.com:52601/UADiscovery"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain collection of application elements
On Error Resume Next
Dim DiscoveryElementCollection: Set DiscoveryElementCollection =
Client.FindLocalApplications(discoveryUriStrings, UAApplicationTypes_Server)
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim DiscoveryElement: For Each DiscoveryElement In DiscoveryElementCollection
    WScript.Echo "DiscoveryElementCollection[""" & DiscoveryElement.DiscoveryUriString
& """].ApplicationUriString: " & _
        DiscoveryElement.ApplicationUriString
Next
```

13.2.13.22 Examples - OPC Unified Architecture - Get arguments of a subscription

C#

```
// This example shows how to obtain parameters of certain monitored item subscription.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples._EasyUAClient
{
    class GetMonitoredItemArguments
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Getting monitored item arguments...");
            EasyUAMonitoredItemArguments monitoredItemArguments =
                client.GetMonitoredItemArguments(handleArray[2]);
            Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor);
            Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval);
            Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval);

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Your code would do the processing here
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain parameters of certain monitored item subscription.

type
  TClientEventHandlers115 = class
    procedure Client_DataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers115.Client_DataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Your code would do the processing here
end;

class procedure GetMonitoredItemArguments.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers115;
  HandleArray: OleVariant;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers115.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.Client_DataChangeNotification;

  WriteLn('Subscribing...');
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;
```

```
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Getting monitored item arguments...');
  MonitoredItemArguments := Client.GetMonitoredItemArguments(HandleArray[2]);

  WriteLn('NodeDescriptor: ', MonitoredItemArguments.NodeDescriptor.ToString);
  WriteLn('SamplingInterval: ', MonitoredItemArguments.MonitoringParameters.ToString);
  WriteLn('PublishingInterval: ',
MonitoredItemArguments.SubscriptionParameters.ToString);

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how to obtain parameters of certain monitored item subscription.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class GetMonitoredItemArguments
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
```

```
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
            )

            Console.WriteLine("Getting monitored item arguments...")
            Dim monitoredItemArguments As EasyUAMonitoredItemArguments =
client.GetMonitoredItemArguments(handleArray(2))
            Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor)
            Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval)
            Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval)

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Your code would do the processing here
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain parameters of certain monitored item subscription.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."

Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
```

```
"nsu=http://test.org/UA/Data/;i=10853"

Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"

Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3

Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

WScript.Echo "Getting monitored item arguments..."
Dim MonitoredItemArguments: Set MonitoredItemArguments =
Client.GetMonitoredItemArguments(handleArray(2))

WScript.Echo "NodeDescriptor: " & MonitoredItemArguments.NodeDescriptor
WScript.Echo "SamplingInterval: " &
MonitoredItemArguments.MonitoringParameters.SamplingInterval
WScript.Echo "PublishingInterval: " &
MonitoredItemArguments.SubscriptionParameters.PublishingInterval

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Your code would do the processing here
End Sub
```

13.2.13.23 Examples - OPC Unified Architecture - Get arguments of all subscriptions

C#

```
// This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;
```

```
namespace UADocExamples._EasyUAClient
{
    class GetMonitoredItemArgumentsDictionary
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Getting monitored item arguments dictionary...");
            EasyUAMonitoredItemArgumentsDictionary monitoredItemArgumentsDictionary =
                client.GetMonitoredItemArgumentsDictionary();

            foreach (EasyUAMonitoredItemArguments monitoredItemArguments in
monitoredItemArgumentsDictionary.Values)
            {
                Console.WriteLine();
                Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor);
                Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval);
                Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval);
            }

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
```

```
            // Your code would do the processing here
        }
    }
}
```

Object Pascal

```
// This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

type
  TClientEventHandlers116 = class
    procedure Client_DataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers116.Client_DataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Your code would do the processing here
end;

class procedure GetMonitoredItemArgumentsDictionary.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers116;
  Count: Cardinal;
  Element: OleVariant;
  HandleArray: OleVariant;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoredItemArguments: _EasyUAMonitoredItemArguments;
  MonitoredItemArgumentsDictionary: _EasyUAMonitoredItemArgumentsDictionary;
  MonitoredItemArgumentsEnumerator: IEnumVARIANT;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers116.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.Client_DataChangeNotification;

  WriteLn('Subscribing...');
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
```

```
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Getting monitored item arguments dictionary...');
  MonitoredItemArgumentsDictionary := Client.GetMonitoredItemArgumentsDictionary;

  MonitoredItemArgumentsEnumerator := MonitoredItemArgumentsDictionary.GetEnumerator;
  while (MonitoredItemArgumentsEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    MonitoredItemArguments := IUnknown(Element.Value) as _EasyUAMonitoredItemArguments;
    WriteLn;
    WriteLn('NodeDescriptor: ', MonitoredItemArguments.NodeDescriptor.ToString);
    WriteLn('SamplingInterval: ',
MonitoredItemArguments.MonitoringParameters.ToString);
    WriteLn('PublishingInterval: ',
MonitoredItemArguments.SubscriptionParameters.ToString);
  end;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VB.NET

```
' This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class GetMonitoredItemArgumentsDictionary
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
```

```
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
             )

            Console.WriteLine("Getting monitored item arguments dictionary...")
            Dim monitoredItemArgumentsDictionary As
EasyUAMonitoredItemArgumentsDictionary = _
                    client.GetMonitoredItemArgumentsDictionary()

            For Each monitoredItemArguments As EasyUAMonitoredItemArguments In
monitoredItemArgumentsDictionary.Values
                Console.WriteLine()
                Console.WriteLine("NodeDescriptor: {0}",
monitoredItemArguments.NodeDescriptor)
                Console.WriteLine("SamplingInterval: {0}",
monitoredItemArguments.MonitoringParameters.SamplingInterval)
                Console.WriteLine("PublishingInterval: {0}",
monitoredItemArguments.SubscriptionParameters.PublishingInterval)
            Next monitoredItemArguments

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Your code would do the processing here
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to obtain dictionary of parameters of all monitored item
subscriptions.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."

Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"

Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3

Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

WScript.Echo "Getting monitored item arguments dictionary..."
Dim MonitoredItemArgumentsDictionary: Set MonitoredItemArgumentsDictionary =
Client.GetMonitoredItemArgumentsDictionary

Dim DictionaryEntry: For Each DictionaryEntry In MonitoredItemArgumentsDictionary
    Dim MonitoredItemArguments: Set MonitoredItemArguments = DictionaryEntry.Value
    WScript.Echo
    WScript.Echo "NodeDescriptor: " & MonitoredItemArguments.NodeDescriptor
    WScript.Echo "SamplingInterval: " &
MonitoredItemArguments.MonitoringParameters.SamplingInterval
    WScript.Echo "PublishingInterval: " &
MonitoredItemArguments.SubscriptionParameters.PublishingInterval
Next

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000

WScript.Echo "Unsubscribing..."
```

```
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Your code would do the processing here
End Sub
```

13.2.13.24 Examples - OPC Unified Architecture - Identify subscriptions by an integer state

C#

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// integer.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void StateAsInteger()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification +=
ClientOnDataChangeNotification_StateAsInteger;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000)
                        {State = 1},    // An integer we have chosen to identify the
subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000)
                        {State = 2},    // An integer we have chosen to identify the
subscription
```

```
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                        {State = 3}     // An integer we have chosen to identify the
subscription
                });

            for (int i = 0; i < handleArray.Length; i++)
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray[i]);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Done.");
        }

        static void ClientOnDataChangeNotification_StateAsInteger(object sender,
EasyUADataChangeNotificationEventArgs eventArgs)
        {
            // Obtain the integer state we have passed in.
            var stateAsInteger = (int) eventArgs.Arguments.State;

            // Display the data
            if (eventArgs.Succeeded)
                Console.WriteLine("{0}: {1}", stateAsInteger, eventArgs.AttributeData);
            else
                Console.WriteLine("{0} *** Failure: {1}", stateAsInteger,
eventArgs.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item
' and display each change, identifying the different subscription by an
' integer.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub StateAsInteger()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
ClientOnDataChangeNotification_StateAsInteger

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10845", 1000) _
                         With {.State = 1}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10853", 1000) _
                         With {.State = 2}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10855", 1000) _
                        With {.State = 3} _
                } _
             ) ' An integer we have chosen to identify the subscription

            For i As Integer = 0 To handleArray.Length - 1
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray(i))
            Next i

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Done.")
        End Sub

        Private Shared Sub ClientOnDataChangeNotification_StateAsInteger(ByVal sender
As Object, ByVal eventArgs As EasyUADataChangeNotificationEventArgs)
            ' Obtain the integer state we have passed in.
            Dim stateAsInteger As Integer = CInt(eventArgs.Arguments.State)

            ' Display the data
            If eventArgs.Succeeded Then
                Console.WriteLine("{0}: {1}", stateAsInteger, eventArgs.AttributeData)
            Else
                Console.WriteLine("{0} *** Failure: {1}", stateAsInteger,
eventArgs.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// integer.

type
  TIntegerClientEventHandlers124 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TIntegerClientEventHandlers124.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
var
  stateAsInteger: Integer;
begin
  // Obtain the integer state we have passed in.
  stateAsInteger := eventArgs.Arguments.State;
  if eventArgs.Succeeded then
      WriteLn(stateAsInteger, ': ', eventArgs.AttributeData.ToString)
  else
      WriteLn(stateAsInteger, ' *** Failure: ', eventArgs.ErrorMessageBrief);
end;


class procedure SubscribeMultipleMonitoredItems.StateAsInteger;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TIntegerClientEventHandlers124;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TIntegerClientEventHandlers124.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;

  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments1.SetState(1);  // An integer we have chosen to identify the
subscription
```

```
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2.SetState(2);  // An integer we have chosen to identify the
subscription

  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3.SetState(3);  // An integer we have chosen to identify the
subscription

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// integer.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
```

```
        // Obtain the integer state we have passed in.
        $stateAsInteger = $E->Arguments->State;
        if ($E->Succeeded)
            printf("%d: %s\n", $stateAsInteger, $E->AttributeData);
        else
            printf("%d *** Failure: %s\n", $stateAsInteger, $E->ErrorMessageBrief);
    }
}


// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;

$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments1->SetState(1);  // An integer we have chosen to identify the
subscription

$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2->SetState(2);  // An integer we have chosen to identify the
subscription

$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3->SetState(3);  // An integer we have chosen to identify the
subscription

$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;

$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
```

```
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

13.2.13.25 Examples - OPC Unified Architecture - Identify subscriptions by an object state

PHP: We have found no way of retrieving the original PHP object from State (COM VARIANT). Use other data types (such as integers) for State in PHP: **Examples - OPC Unified Architecture - Identify subscriptions by an integer state (Section 13.2.13.24)** .

C#

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// object.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        class CustomObject
        {
            public CustomObject(string name)
            {
                Name = name;
            }

            public string Name { get; private set; }
        }


        public static void StateAsObject()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"
```

```
            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification +=
ClientOnDataChangeNotification_StateAsObject;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000)
                        {State = new CustomObject("First")},    // A custom object that
corresponds to the subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000)
                        {State = new CustomObject("Second")},    // A custom object
that corresponds to the subscription
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                        {State = new CustomObject("Third")},    // A custom object that
corresponds to the subscription
                });

            for (int i = 0; i < handleArray.Length; i++)
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray[i]);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);

            Console.WriteLine("Done.");
        }

        static void ClientOnDataChangeNotification_StateAsObject(object sender,
EasyUADataChangeNotificationEventArgs eventArgs)
        {
            // Obtain the custom object we have passed in.
            var stateAsObject = (CustomObject) eventArgs.Arguments.State;

            // Display the data
            if (eventArgs.Succeeded)
                Console.WriteLine("{0}: {1}", stateAsObject.Name,
eventArgs.AttributeData);
            else
                Console.WriteLine("{0} *** Failure: {1}", stateAsObject.Name,
eventArgs.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item
' and display each change, identifying the different subscription by an
' object.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Class CustomObject
            Public Sub New(ByVal name As String)
                _Name = name
            End Sub
            Public ReadOnly Property Name As String
                Get
                    Return _Name
                End Get
            End Property
            Private ReadOnly _Name As String
        End Class

        Public Shared Sub StateAsObject()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
ClientOnDataChangeNotification_StateAsObject

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10845", 1000) _
                         With {.State = New CustomObject("First")}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10853", 1000) _
                         With {.State = New CustomObject("Second")}, _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10855", 1000) _
                        With {.State = New CustomObject("Third")} _
                } _
            ) ' A custom object that corresponds to the subscription

            For i As Integer = 0 To handleArray.Length - 1
                Console.WriteLine("handleArray[{0}]: {1}", i, handleArray(i))
            Next i
```

```
            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)

            Console.WriteLine("Done.")
        End Sub

        Private Shared Sub ClientOnDataChangeNotification_StateAsObject(ByVal sender As
Object, ByVal eventArgs As EasyUADataChangeNotificationEventArgs)
            ' Obtain the custom object we have passed in.
            Dim stateAsObject As CustomObject = CType(eventArgs.Arguments.State,
CustomObject)

            ' Display the data
            If eventArgs.Succeeded Then
                Console.WriteLine("{0}: {1}", stateAsObject.Name,
eventArgs.AttributeData)
            Else
                Console.WriteLine("{0} *** Failure: {1}", stateAsObject.Name,
eventArgs.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change, identifying the different subscription by an
// object.

type
  TCustomObject = class
    Name: string;
    constructor Create(name: string);
  end;

constructor TCustomObject.Create(name: string);
begin
  Self.Name := name;
end;

type
  TObjectClientEventHandlers125 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;
```

```
procedure TObjectClientEventHandlers125.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
var
  stateAsObject: TCustomObject;
begin
  // Obtain the custom object we have passed in.
  stateAsObject := TCustomObject(INT_PTR(eventArgs.Arguments.State));
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(stateAsObject.Name, ': ', eventArgs.AttributeData.ToString)
  else
      WriteLn(stateAsObject.Name, ' *** Failure: ', eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleMonitoredItems.StateAsObject;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TObjectClientEventHandlers125;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  CustomObject1, CustomObject2, CustomObject3: TCustomObject;
begin
  CustomObject1 := TCustomObject.Create('First');
  CustomObject2 := TCustomObject.Create('Second');
  CustomObject3 := TCustomObject.Create('Third');

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TObjectClientEventHandlers125.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;

  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments1.SetState(INT_PTR(CustomObject1));  // A custom object that
corresponds to the subscription

  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
```

```
  MonitoredItemArguments2.SetState(INT_PTR(CustomObject2));  // A custom object that
corresponds to the subscription

  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3.SetState(INT_PTR(CustomObject3));  // A custom object that
corresponds to the subscription

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
  FreeAndNil(CustomObject1);
  FreeAndNil(CustomObject2);
  FreeAndNil(CustomObject3);
end;
```

13.2.13.26 Examples - OPC Unified Architecture - Parse a relative browse path

C#

```
// Parses a relative OPC-UA browse path and displays its elements.

using System;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;
```

```
namespace UADocExamples._UABrowsePathParser
{
    class ParseRelative
    {
        public static void Main1()
        {
            var browsePathParser = new UABrowsePathParser();
            UABrowsePathElementCollection browsePathElements;
            try
            {
                browsePathElements =
browsePathParser.ParseRelative("/Data.Dynamic.Scalar.CycleComplete");
            }
            catch (UABrowsePathFormatException browsePathFormatException)
            {
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException().Message);
                return;
            }

            // Display results
            foreach (UABrowsePathElement browsePathElement in browsePathElements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // /Data
            // .Dynamic
            // .Scalar
            // .CycleComplete
        }
    }
}
```

Object Pascal

```
// Parses a relative OPC-UA browse path and displays its elements.

class procedure ParseRelative.Main;
var
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathElements: _UABrowsePathElementCollection;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;

  try
    BrowsePathElements :=
BrowsePathParser.ParseRelative('/Data.Dynamic.Scalar.CycleComplete');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
```

```
    end;
  end;

  // Display results
  ElementEnumerator := BrowsePathElements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // /Data
  // .Dynamic
  // .Scalar
  // .CycleComplete

end;
```

PHP

```
// Parses a relative OPC-UA browse path and displays its elements.


$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

try
{
    $BrowsePathElements = $BrowsePathParser-
>ParseRelative("/Data.Dynamic.Scalar.CycleComplete");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
for ($i = 0; $i < $BrowsePathElements->Count; $i++)
{
    printf("%s\n", $BrowsePathElements[$i]);
}

// Example output:
// /Data
// .Dynamic
// .Scalar
// .CycleComplete
```

VB.NET

```
' Parses a relative OPC-UA browse path and displays its elements.

Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing
```

```
Namespace UADocExamples._UABrowsePathParser
    Friend Class ParseRelative
        Public Shared Sub Main1()
            Dim browsePathParser = New UABrowsePathParser()
            Dim browsePathElements As UABrowsePathElementCollection
            Try
                browsePathElements =
browsePathParser.ParseRelative("/Data.Dynamic.Scalar.CycleComplete")
            Catch browsePathFormatException As UABrowsePathFormatException
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            For Each browsePathElement As UABrowsePathElement In browsePathElements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' /Data
            ' .Dynamic
            ' .Scalar
            ' .CycleComplete
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Parses a relative OPC-UA browse path and displays its elements.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
On Error Resume Next
Dim BrowsePathElements: Set BrowsePathElements =
BrowsePathParser.ParseRelative("/Data.Dynamic.Scalar.CycleComplete")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
Dim BrowsePathElement: For Each BrowsePathElement In BrowsePathElements
    WScript.Echo BrowsePathElement
Next
```

13.2.13.27 Examples - OPC Unified Architecture - Parse an absolute browse path

C#

```
// Parses an absolute  OPC-UA browse path and displays its starting node and elements.

using System;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class Parse
    {
        public static void Main1()
        {
            var browsePathParser = new UABrowsePathParser();
            UABrowsePath browsePath;
            try
            {
                browsePath = browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar");
            }
            catch (UABrowsePathFormatException browsePathFormatException)
            {
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId);

            foreach (UABrowsePathElement browsePathElement in browsePath.Elements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // StartingNodeId: ObjectsFolder
            // /Data
            // /Static
            // /UserScalar
        }
    }
}
```

Object Pascal

```
// Parses an absolute  OPC-UA browse path and displays its starting node and elements.

class procedure Parse.Main;
var
  BrowsePath: _UABrowsePath;
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
```

```
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;

  try
    BrowsePath := BrowsePathParser.Parse('[ObjectsFolder]/Data/Static/UserScalar');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('StartingNodeId: ', BrowsePath.StartingNodeId.ToString);

  WriteLn('Elements:');
  ElementEnumerator := BrowsePath.Elements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // StartingNodeId: ObjectsFolder
  // Elements:
  // /Data
  // /Static
  // /UserScalar

end;
```

PHP

```
// Parses an absolute  OPC-UA browse path and displays its starting node and elements.


$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

try
{
    $BrowsePath = $BrowsePathParser->Parse("[ObjectsFolder]/Data/Static/UserScalar");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    exit();
}

// Display results
printf("StartingNodeId: %s\n", $BrowsePath->StartingNodeId);
```

```
printf("Elements:\n");

for ($i = 0; $i < $BrowsePath->Elements->Count; $i++)
{
    printf("%s\n", $BrowsePath->Elements[$i]);
}

// Example output:
// StartingNodeId: ObjectsFolder
// Elements:
// /Data
// /Static
// /UserScalar
```

VB.NET

```
' Parses an absolute  OPC-UA browse path and displays its starting node and elements.

Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class Parse
        Public Shared Sub Main1()
            Dim browsePathParser = New UABrowsePathParser()
            Dim browsePath As UABrowsePath
            Try
                browsePath = browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar")
            Catch browsePathFormatException As UABrowsePathFormatException
                Console.WriteLine("*** Failure: {0}",
browsePathFormatException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId)

            For Each browsePathElement As UABrowsePathElement In browsePath.Elements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' StartingNodeId: ObjectsFolder
            ' /Data
            ' /Static
            ' /UserScalar
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Parses an absolute  OPC-UA browse path and displays its starting node and elements.

Option Explicit
```

```
Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
On Error Resume Next
Dim BrowsePath: Set BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "StartingNodeId: " & BrowsePath.StartingNodeId

WScript.Echo "Elements:"
Dim BrowsePathElement: For Each BrowsePathElement In BrowsePath.Elements
    WScript.Echo BrowsePathElement
Next
```

13.2.13.28 Examples - OPC Unified Architecture - Read a range of elements from an array

C#

```
// This example shows how to read a range of values from an array.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UAIndexRangeList
{
    class Usage
    {
        public static void ReadValue()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain the value, indicating that just the elements 2 to 4 should be
returned
            object value;
            try
            {
                value = client.ReadValue(
```

```
                    new UAReadArguments(
                        endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;ns=2;i=10305",
                        UAIndexRangeList.OneDimension(2, 4)));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Cast to typed array
            var arrayValue = (Int32[]) value;

            // Display results
            for (int i = 0; i < 3; i++)
                Console.WriteLine("arrayValue[{0}]: {1}", i, arrayValue[i]);


            // Example output:
            //arrayValue[0]: 180410224
            //arrayValue[1]: 1919239969
            //arrayValue[2]: 1700185172
        }
    }
}
```

Object Pascal

```
// This example shows how to read a range of values from an array.

class procedure Usage.ReadValue;
var
  Client: _EasyUAClient;
  EndpointDescriptor: string;
  ReadArguments1: _UAReadArguments;
  Arguments, Results: OleVariant;
  I: Cardinal;
  IndexRange: _UAIndexRange;
  IndexRangeList: OpcLabs_EasyOpcUA_TLB._UAIndexRangeList;
  ArrayValue: OleVariant;
  Value: Integer;
  ValueResult: _ValueResult;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  //or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer';
  //or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Prepare the arguments, indicating that just the elements 2 to 4 should be
returned.
  IndexRangeList := CoUAIndexRangeList.Create;
  IndexRange := CoUAIndexRange.Create;
  IndexRange.Minimum := 2;
```

```
  IndexRange.Maximum := 4;
  IndexRangeList.Add(IndexRange);

  ReadArguments1 := CoUAReadArguments.Create;
  ReadArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  ReadArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;ns=2;i=10305';
  ReadArguments1.IndexRangeList := IndexRangeList;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := ReadArguments1;

  // Obtain value.
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultipleValues(Arguments));

  ValueResult := IInterface(Results[0]) as _ValueResult;
  if not ValueResult.Succeeded then
    begin
      WriteLn(' *** Failure: ', ValueResult.Exception.GetBaseException.Message);
      Exit;
    end;

  // Display results
  ArrayValue := ValueResult.Value;
  for I := VarArrayLowBound(ArrayValue, 1) to VarArrayHighBound(ArrayValue, 1) do
  begin
      Value := ArrayValue[I];
      WriteLn('arrayValue[', I, ']: ', Value);
  end;

  // Example output:
  //arrayValue[0]: 180410224
  //arrayValue[1]: 1919239969
  //arrayValue[2]: 1700185172
end;
```

PHP

```
// This example shows how to read a range of values from an array.

$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
//or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer";
//or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Prepare the arguments, indicating that just the elements 2 to 4 should be returned.
$IndexRangeList = new COM("OpcLabs.EasyOpc.UA.UAIndexRangeList");
$IndexRange = new COM("OpcLabs.EasyOpc.UA.UAIndexRange");
$IndexRange->Minimum = 2;
$IndexRange->Maximum = 4;
$IndexRangeList->Add($IndexRange);

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
```

```
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;ns=2;i=10305";
$ReadArguments1->IndexRangeList = $IndexRangeList;

$arguments[0] = $ReadArguments1;

// Obtain value.
$results = $Client->ReadMultipleValues($arguments);

$ValueResult = $results[0];

if (!$ValueResult->Succeeded) {
    printf("*** Failure: %s\n", $ValueResult->ErrorMessageBrief);
    Exit();
}

// Display results
$ArrayValue = $ValueResult->Value;
for ($i = 0; $i <= 2; $i++)
{
    printf("arrayValue[%d]: %s\n", $i, $ArrayValue[$i]);
}

// Example output:
//arrayValue[0]: 180410224
//arrayValue[1]: 1919239969
//arrayValue[2]: 1700185172
```

VB.NET

```
' This example shows how to read a range of values from an array.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._UAIndexRangeList
    Friend Class Usage
        Public Shared Sub ReadValue()

            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain the value, indicating that just the elements 2 to 4 should be
returned
            Dim value As Object
            Try
                value = client.ReadValue( _
                    New UAReadArguments( _
                        endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10305", _
```

```
                        UAIndexRangeList.OneDimension(2, 4)))
                ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Cast to typed array
            Dim arrayValue = DirectCast(value, Int32())

            ' Display results
            For i = 0 To 2
                Console.WriteLine("arrayValue[{0}]: {1}", i, arrayValue(i))
            Next
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to read a range of values from an array.

Option Explicit

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Prepare the arguments, indicating that just the elements 2 to 4 should be returned.
Dim IndexRangeList: Set IndexRangeList =
CreateObject("OpcLabs.EasyOpc.UA.UAIndexRangeList")
Dim IndexRange: Set IndexRange = CreateObject("OpcLabs.EasyOpc.UA.UAIndexRange")
IndexRange.Minimum = 2
IndexRange.Maximum = 4
IndexRangeList.Add IndexRange
'
Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = endpointDescriptor
ReadArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;ns=2;i=10305"
ReadArguments1.IndexRangeList = IndexRangeList

Dim arguments(0)
Set arguments(0) = ReadArguments1

' Obtain the value.
Dim results: results = Client.ReadMultipleValues(arguments)
Dim ValueResult: Set ValueResult = results(0)
If Not ValueResult.Succeeded Then
    WScript.Echo "*** Failure: " & ValueResult.Exception.GetBaseException().Message
    WScript.Quit
```

```
End If
' VBScript can only handle well arrays of VARIANTs; most other COM tool will be able to
use simply the .Value property.
Dim arrayValue: arrayValue = ValueResult.RegularizedValue

' Display results
Dim i: For i = 0 To 2
    WScript.Echo "arrayValue[" & i & "]: " & arrayValue(i)
Next


' Example output:
'arrayValue[0]: 180410224
'arrayValue[1]: 1919239969
'arrayValue[2]: 1700185172
```

13.2.13.29 Examples - OPC Unified Architecture - Read a single attribute of a single node

C#

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class Read
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain attribute data. By default, the Value attribute of a node will be
read.
            UAAttributeData attributeData;
            try
            {
                attributeData = client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
            }
```

```
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("Value: {0}", attributeData.Value);
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp);
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp);
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode);

            // Example output:
            //
            //Value: -2.230064E-31
            //ServerTimestamp: 11/6/2011 1:34:30 PM
            //SourceTimestamp: 11/6/2011 1:34:30 PM
            //StatusCode: Good
        }
    }
}
```

VB.NET

```
' This example shows how to read and display data of an attribute (value, timestamps,
and status code).

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class Read
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain attribute data. By default, the Value attribute of a node will be
read.
            Dim attributeData As UAAttributeData
            Try
                attributeData = client.Read(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
```

```
            ' Display results
            Console.WriteLine("Value: {0}", attributeData.Value)
            Console.WriteLine("ServerTimestamp: {0}", attributeData.ServerTimestamp)
            Console.WriteLine("SourceTimestamp: {0}", attributeData.SourceTimestamp)
            Console.WriteLine("StatusCode: {0}", attributeData.StatusCode)

            ' Example output:
            '
            'Value: -2.230064E-31
            'ServerTimestamp: 11/6/2011 1:34:30 PM
            'SourceTimestamp: 11/6/2011 1:34:30 PM
            'StatusCode: Good
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "Read.h"

namespace _EasyUAClient
{
    void Read::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Obtain attribute data. By default, the Value attribute of a node will be
read.
            _UAAttributeDataPtr AttributeDataPtr = ClientPtr->Read(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10853"); // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"

            // Display results
            _variant_t vString;
            vString.ChangeType(VT_BSTR, &AttributeDataPtr->Value);
            _tprintf(_T("Value: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            vString.ChangeType(VT_BSTR, &_variant_t(AttributeDataPtr->ServerTimestamp,
VT_DATE));
            _tprintf(_T("ServerTimestamp: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            vString.ChangeType(VT_BSTR, &_variant_t(AttributeDataPtr->SourceTimestamp,
VT_DATE));
            _tprintf(_T("SourceTimestamp: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            _tprintf(_T("StatusCode: %s\n"), (LPCTSTR)CW2CT(AttributeDataPtr-
>StatusCode->ToString));

            // Example output:
            //
            //Value: -2.230064E-31
```

```
            //ServerTimestamp: 11/6/2011 1:34:30 PM
            //SourceTimestamp: 11/6/2011 1:34:30 PM
            //StatusCode: Good

        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

PHP

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Obtain attribute data. By default, the Value attribute of a node will be read.
try
{
    $AttributeData = $Client->Read(
        "http://opcua.demo-this.com:51211/UA/SampleServer",
        "nsu=http://test.org/UA/Data/;i=10853"); // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}

// Display results
printf("Value: %s\n", $AttributeData->Value);
printf("ServerTimestamp: %s\n", $AttributeData->ServerTimestamp);
printf("SourceTimestamp: %s\n", $AttributeData->SourceTimestamp);
printf("StatusCode: %s\n", $AttributeData->StatusCode);

// Example output:
//
//Value: -2.230064E-31
//ServerTimestamp: 11/6/2011 1:34:30 PM
//SourceTimestamp: 11/6/2011 1:34:30 PM
//StatusCode: Good
```

PowerScript

```
// This example shows how to read and display data of an attribute (value, timestamps,
and status code).

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")
```

```
// Obtain attribute data. By default, the Value attribute of a node will be read.
OLEObject attributeData
TRY
    attributeData = client.Read("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")        // or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + "Value: " + String(attributeData.value) +
"~r~n"
mle_outputtext.Text = mle_outputtext.Text + "ServerTimestamp: " +
String(attributeData.ServerTimestamp) + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "SourceTimestamp: " +
String(attributeData.SourceTimestamp) + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "StatusCode: " +
String(attributeData.StatusCode) + "~r~n"

// Example output:
//
//Value: -2.230064E-31
//ServerTimestamp: 11/6/2011 1:34:30 PM
//SourceTimestamp: 11/6/2011 1:34:30 PM
//StatusCode: Good
```

Python

```
# This example shows how to read and display data of an attribute (value, timestamps,
and status code).

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Obtain attribute data. By default, the Value attribute of a node will be read.
attributeData = client.Read('http://opcua.demo-this.com:51211/UA/SampleServer', # or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
                            'nsu=http://test.org/UA/Data/;i=10853')

# Display results
print('Value: ', attributeData.Value)
print('ServerTimestamp: ', attributeData.ServerTimestamp)
print('SourceTimestamp: ', attributeData.SourceTimestamp)
print('StatusCode: ', attributeData.StatusCode)

# Example output:
#
#Value: -2.230064E-31
#ServerTimestamp: 11/6/2011 1:34:30 PM
#SourceTimestamp: 11/6/2011 1:34:30 PM
```

```
#StatusCode: Good
```

Visual Basic (VB 6.)

```
Rem This example shows how to read and display data of an attribute (value, timestamps,
and status code).

Private Sub Read_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Obtain attribute data. By default, the Value attribute of a node will be read.
    On Error Resume Next
    Dim AttributeData As UAAttributeData
    Set AttributeData = Client.Read("http://opcua.demo-this.com:51211/UA/SampleServer",
_
                                    "nsu=http://test.org/UA/Data/;i=10853") ' or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    OutputText = OutputText & "Value: " & AttributeData.value & vbCrLf
    OutputText = OutputText & "ServerTimestamp: " & AttributeData.ServerTimestamp &
vbCrLf
    OutputText = OutputText & "SourceTimestamp: " & AttributeData.SourceTimestamp &
vbCrLf
    OutputText = OutputText & "StatusCode: " & AttributeData.StatusCode & vbCrLf

    ' Example output:
    '
    'Value: -2.230064E-31
    'ServerTimestamp: 11/6/2011 1:34:30 PM
    'SourceTimestamp: 11/6/2011 1:34:30 PM
    'StatusCode: Good
End Sub
```

VBScript

```
Rem This example shows how to read and display data of an attribute (value, timestamps,
and status code).

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Obtain attribute data. By default, the Value attribute of a node will be read.
On Error Resume Next
Dim AttributeData: Set AttributeData = Client.Read("http://opcua.demo-
```

```
this.com:51211/UA/SampleServer", _

"nsu=http://test.org/UA/Data/;i=10853") ' or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "Value: " & AttributeData.Value
WScript.Echo "ServerTimestamp: " & AttributeData.ServerTimestamp
WScript.Echo "SourceTimestamp: " & AttributeData.SourceTimestamp
WScript.Echo "StatusCode: " & AttributeData.StatusCode

' Example output:
'
'Value: -2.230064E-31
'ServerTimestamp: 11/6/2011 1:34:30 PM
'SourceTimestamp: 11/6/2011 1:34:30 PM
'StatusCode: Good
```

13.2.13.30 Examples - OPC Unified Architecture - Read a single value

C#

```
// This example shows how to read value of a single node, and display it.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadValue
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain value of a node
            object value;
```

```
            try
            {
                value = client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("value: {0}", value);
        }
    }
}
```

VB.NET

```
' This example shows how to read value of a single node, and display it.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadValue
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain value of a node
            Dim value As Object
            Try
                value = client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("value: {0}", value)
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read value of a single node, and display it.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ReadValue.h"

namespace _EasyUAClient
{
    void ReadValue::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            _variant_t value = ClientPtr->ReadValue(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10853");

            // Display results
            _variant_t vString;
            vString.ChangeType(VT_BSTR, &value);
            _tprintf(_T("value: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

JScript

```
// This example shows how to read value of a single node, and display it.
var Client = new ActiveXObject("OpcLabs.EasyOpc.UA.EasyUAClient");
var value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
WScript.Echo("value: " + value);
```

Free Pascal

```
// This example shows how to read value of a single node, and display it.

class procedure ReadValue.Main;
var
  Client: EasyUAClient;
  Value: OleVariant;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  Value := Client.ReadValue(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853');
  WriteLn('value: ', Value);
```

```
end;
```

Object Pascal

```
// This example shows how to read value of a single node, and display it.

class procedure ReadValue.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Value: OleVariant;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Obtain value of a node
  try
    Value := Client.ReadValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer',
      'nsu=http://test.org/UA/Data/;i=10853');
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

  // Display results
  WriteLn('value: ', Value);
end;
```

PHP

```
// This example shows how to read value of a single node, and display it.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
    Exit();
}


// Display results
printf("Value: %s\n", $value);
```

PowerScript

```
// This example shows how to read value of a single node, and display it.
```

```
mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Obtain value of a node
mle_outputtext.Text = mle_outputtext.Text + "Reading node value..." + "~r~n"
Any value
TRY
    value = client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

// Display results
mle_outputtext.Text = mle_outputtext.Text + String(value) + "~r~n"
```

Python

```
# This example shows how to read value of a single node, and display it.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Perform the operation
value = client.ReadValue('http://opcua.demo-this.com:51211/UA/SampleServer',
'nsu=http://test.org/UA/Data/;i=10853')

# Display results
print('value: ', value)
```

Visual Basic (VB 6.)

```
Rem This example shows how to read value of a single node, and display it.

Private Sub ReadValue_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Dim value As Variant
    value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
```

```
        Exit Sub
    End If
    On Error GoTo 0

    ' Display results
    OutputText = OutputText & "value: " & value & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to read value of a single node, and display it.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' Display results
WScript.Echo "value: " & value
```

13.2.13.31 Examples - OPC Unified Architecture - Read DataType attributes

C#

```
// This example shows how to read the Value attributes of 3 different nodes at once.
Using the same method, it is also possible
// to read multiple attributes of the same node.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultipleValues
    {
        public static void DataType()
        {
            UAEndpointDescriptor endpointDescriptor =
```

```
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain values.
            ValueResult[] valueResultArray = client.ReadMultipleValues(new[]
                {
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", UAAttributeId.DataType),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", UAAttributeId.DataType),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", UAAttributeId.DataType)
                });

            // Display results
            foreach (ValueResult valueResult in valueResultArray)
            {
                Console.WriteLine();

                if (valueResult.Succeeded)
                {
                    Console.WriteLine($"Value: {valueResult.Value}");
                    var dataTypeId = valueResult.Value as UANodeId;
                    if (!(dataTypeId is null))
                    {
                        Console.WriteLine($"Value.ExpandedText:
{dataTypeId.ExpandedText}");
                        Console.WriteLine($"Value.NamespaceUriString:
{dataTypeId.NamespaceUriString}");
                        Console.WriteLine($"Value.NamespaceIndex:
{dataTypeId.NamespaceIndex}");
                        Console.WriteLine($"Value.NumericIdentifier:
{dataTypeId.NumericIdentifier}");
                    }
                }
                else
                    Console.WriteLine($"*** Failure: {valueResult.ErrorMessageBrief}");
            }

            // Example output:
            //
            //Value: SByte
            //Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2
            //Value.NamespaceUriString: http://opcfoundation.org/UA/
            //Value.NamespaceIndex: 0
            //Value.NumericIdentifier: 2
            //
            //Value: Float
            //Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=10
            //Value.NamespaceUriString: http://opcfoundation.org/UA/
            //Value.NamespaceIndex: 0
            //Value.NumericIdentifier: 10
```

```
            //
            //Value: String
            //Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=12
            //Value.NamespaceUriString: http://opcfoundation.org/UA/
            //Value.NamespaceIndex: 0
            //Value.NumericIdentifier: 12
        }
    }
}
```

VB.NET

```
' This example shows how to read the Value attributes of 3 different nodes at once.
Using the same method, it is also possible
' to read multiple attributes of the same node.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultipleValues
        Public Shared Sub DataType()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain values.
            Dim valueResultArray() As ValueResult = client.ReadMultipleValues(New
UAReadArguments() _
                {
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", UAAttributeId.DataType),
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", UAAttributeId.DataType),
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", UAAttributeId.DataType)
                }
            )

            ' Display results
            For Each valueResult As ValueResult In valueResultArray
                Console.WriteLine()

                If valueResult.Succeeded Then
                    Console.WriteLine("Value: {0}", valueResult.Value)
                    Dim dataTypeId = CType(valueResult.Value, UANodeId)
                    If Not dataTypeId Is Nothing Then
```

```
                        Console.WriteLine("Value.ExpandedText: {0}",
dataTypeId.ExpandedText)
                        Console.WriteLine("Value.NamespaceUriString: {0}",
dataTypeId.NamespaceUriString)
                        Console.WriteLine("Value.NamespaceIndex: {0}",
dataTypeId.NamespaceIndex)
                        Console.WriteLine("Value.NumericIdentifier: {0}",
dataTypeId.NumericIdentifier)
                    End If
                Else
                    Console.WriteLine("*** Failure: {0}",
valueResult.ErrorMessageBrief)
                End If
            Next valueResult

            ' Example output:
            '
            'Value: SByte
            'Value.ExpandedText: nsu = http : //opcfoundation.org/UA/ ;i=2
            'Value.NamespaceUriString: http : //opcfoundation.org/UA/
            'Value.NamespaceIndex: 0
            'Value.NumericIdentifier: 2
            '
            'Value: Float
            'Value.ExpandedText: nsu = http : //opcfoundation.org/UA/ ;i=10
            'Value.NamespaceUriString: http : //opcfoundation.org/UA/
            'Value.NamespaceIndex: 0
            'Value.NumericIdentifier: 10
            '
            'Value: String
            'Value.ExpandedText: nsu = http : //opcfoundation.org/UA/ ;i=12
            'Value.NamespaceUriString: http : //opcfoundation.org/UA/
            'Value.NamespaceIndex: 0
            'Value.NumericIdentifier: 12
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to read the DataType attributes of 3 different nodes at
// once. Using the same method, it is also possible to read multiple attributes
// of the same node.

const UAAttributeId_DataType = 14;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$ReadArguments1->AttributeId = UAAttributeId_DataType;

$ReadArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
```

```
$ReadArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$ReadArguments2->AttributeId = UAAttributeId_DataType;

$ReadArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$ReadArguments3->AttributeId = UAAttributeId_DataType;

$arguments[0] = $ReadArguments1;
$arguments[1] = $ReadArguments2;
$arguments[2] = $ReadArguments3;

// Obtain values. By default, the Value attributes of the nodes will be read.
$results = $Client->ReadMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $ValueResult = $results[$i];
    printf("\n");
    if ($ValueResult->Succeeded)
    {
        printf("Value: %s\n", $ValueResult->Value);
        printf("Value.ExpandedText: %s\n", $ValueResult->Value->ExpandedText);
        printf("Value.NamespaceUriString: %s\n", $ValueResult->Value-
>NamespaceUriString);
        printf("Value.NamespaceIndex: %s\n", $ValueResult->Value->NamespaceIndex);
        printf("Value.NumericIdentifier: %s\n", $ValueResult->Value-
>NumericIdentifier);
    }
    else
        printf("*** Failure: %s\n", $ValueResult->ErrorMessageBrief);
}

// Example output:
//
//
//Value: SByte
//Value.ExpandedText: nsu=http://opcfoundation.org/UA/;i=2
//Value.NamespaceUriString: http://opcfoundation.org/UA/
//Value.NamespaceIndex: 0
//Value.NumericIdentifier: 2
//
//Value: Float
//Value.ExpandedText: nsu=http://opcfoundation.org/UA/;i=10
//Value.NamespaceUriString: http://opcfoundation.org/UA/
//Value.NamespaceIndex: 0
//Value.NumericIdentifier: 10
//
//Value: String
//Value.ExpandedText: nsu=http://opcfoundation.org/UA/;i=12
//Value.NamespaceUriString: http://opcfoundation.org/UA/
```

```
//Value.NamespaceIndex: 0
//Value.NumericIdentifier: 12
```

Visual Basic (VB 6.)

```
Rem This example shows how to read the Value attributes of 3 different nodes at once.
Using the same method, it is also possible
Rem to read multiple attributes of the same node.

Private Sub ReadMultipleValues_DataType_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    Dim ReadArguments1 As New UAReadArguments
    ReadArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"
    ReadArguments1.AttributeId = UAAttributeId_DataType

    Dim ReadArguments2 As New UAReadArguments
    ReadArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"
    ReadArguments2.AttributeId = UAAttributeId_DataType

    Dim ReadArguments3 As New UAReadArguments
    ReadArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10855"
    ReadArguments3.AttributeId = UAAttributeId_DataType

    Dim arguments(2) As Variant
    Set arguments(0) = ReadArguments1
    Set arguments(1) = ReadArguments2
    Set arguments(2) = ReadArguments3

    ' Obtain values.
    Dim results() As Variant
    results = Client.ReadMultipleValues(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        OutputText = OutputText & vbCrLf

        Dim Result As ValueResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "Value: " & Result.value & vbCrLf
            On Error Resume Next
            OutputText = OutputText & "Value.ExpandedText: " &
Result.value.expandedText & vbCrLf
            OutputText = OutputText & "Value.NamespaceUriString: " &
```

```
Result.value.NamespaceUriString & vbCrLf
            OutputText = OutputText & "Value.NamespaceIndex: " &
Result.value.NamespaceIndex & vbCrLf
            OutputText = OutputText & "Value.NumericIdentifier: " &
Result.value.NumericIdentifier & vbCrLf
            On Error GoTo 0
        Else
            OutputText = OutputText & "*** Failure: " & Result.ErrorMessageBrief &
vbCrLf
        End If
    Next

    ' Example output:
    '
    'Value: SByte
    'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2
    'Value.NamespaceUriString: http://opcfoundation.org/UA/
    'Value.NamespaceIndex: 0
    'Value.NumericIdentifier: 2
    '
    'Value: Float
    'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=10
    'Value.NamespaceUriString: http://opcfoundation.org/UA/
    'Value.NamespaceIndex: 0
    'Value.NumericIdentifier: 10
    '
    'Value: String
    'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=12
    'Value.NamespaceUriString: http://opcfoundation.org/UA/
    'Value.NamespaceIndex: 0
    'Value.NumericIdentifier: 12
End Sub
```

VBScript

```
Rem This example shows how to read the DataType attributes of 3 different nodes at
once. Using the same method, it is also possible
Rem to read multiple attributes of the same node.

Option Explicit

Const UAAttributeId_DataType = 14

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
ReadArguments1.AttributeId = UAAttributeId_DataType

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
```

```
this.com:51211/UA/SampleServer"
ReadArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
ReadArguments2.AttributeId = UAAttributeId_DataType

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
ReadArguments3.AttributeId = UAAttributeId_DataType

Dim arguments(2)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3

' Obtain values.
Dim results: results = Client.ReadMultipleValues(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    WScript.Echo

    Dim ValueResult: Set ValueResult = results(i)
    If ValueResult.Succeeded Then
        WScript.Echo "Value: " & ValueResult.Value
        On Error Resume Next
        WScript.Echo "Value.ExpandedText: " & ValueResult.Value.ExpandedText
        WScript.Echo "Value.NamespaceUriString: " &
ValueResult.Value.NamespaceUriString
        WScript.Echo "Value.NamespaceIndex: " & ValueResult.Value.NamespaceIndex
        WScript.Echo "Value.NumericIdentifier: " & ValueResult.Value.NumericIdentifier
        On Error Goto 0
    Else
        WScript.Echo "*** Failure: " & ValueResult.ErrorMessageBrief
    End If
Next

' Example output:
'
'Value: SByte
'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=2
'Value.NamespaceUriString: http://opcfoundation.org/UA/
'Value.NamespaceIndex: 0
'Value.NumericIdentifier: 2
'
'Value: Float
'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=10
'Value.NamespaceUriString: http://opcfoundation.org/UA/
'Value.NamespaceIndex: 0
'Value.NumericIdentifier: 10
'
'Value: String
'Value.ExpandedText: nsu=http://opcfoundation.org/UA/ ;i=12
'Value.NamespaceUriString: http://opcfoundation.org/UA/
```

```
'Value.NamespaceIndex: 0
'Value.NumericIdentifier: 12
```

13.2.13.32 Examples - OPC Unified Architecture - Read multiple nodes or attributes

C#

```
// This example shows how to read data (value, timestamps, and status code) of 3
attributes at once. In this example,
// we are reading a Value attribute of 3 different nodes, but the method can also be
used to read multiple attributes
// of the same node.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultiple
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain attribute data. By default, the Value attributes of the nodes
will be read.
            UAAttributeDataResult[] attributeDataResultArray =
client.ReadMultiple(new[]
                {
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845"),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853"),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855")
                });

            // Display results
            foreach (UAAttributeDataResult attributeDataResult in
attributeDataResultArray)
            {
                if (attributeDataResult.Succeeded)
                    Console.WriteLine("AttributeData: {0}",
attributeDataResult.AttributeData);
```

```
                else
                    Console.WriteLine("*** Failure: {0}",
attributeDataResult.ErrorMessageBrief);
            }
        }

        // Example output:
        //
        //AttributeData: 51 {System.Int16} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM;
Good
        //AttributeData: -1993984 {System.Single} @11/6/2011 1:49:19 PM @11/6/2011
1:49:19 PM; Good
        //AttributeData: Yellow% Dragon Cat) White Blue Dog# Green Banana-
{System.String} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
    }
}
```

VB.NET

```
' This example shows how to read data (value, timestamps, and status code) of 3
attributes at once. In this example,
' we are reading a Value attribute of 3 different nodes, but the method can also be
used to read multiple attributes
' of the same node.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultiple
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain attribute data. By default, the Value attributes of the nodes will
be read.
            Dim attributeDataResultArray() As UAAttributeDataResult =
client.ReadMultiple(New UAReadArguments() _
                {
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845"),
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853"),
                    New UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855")
                })

            ' Display results
            For Each attributeDataResult As UAAttributeDataResult In
```

```
attributeDataResultArray
                If attributeDataResult.Succeeded Then
                    Console.WriteLine("AttributeData: {0}",
attributeDataResult.AttributeData)
                Else
                    Console.WriteLine("*** Failure: {0}",
attributeDataResult.ErrorMessageBrief)
                End If
            Next attributeDataResult

            ' Example output:
            '
            'AttributeData: 51 {System.Int16} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19
PM; Good
            'AttributeData: -1993984 {System.Single} @11/6/2011 1:49:19 PM @11/6/2011
1:49:19 PM; Good
            'AttributeData: Yellow% Dragon Cat) White Blue Dog# Green Banana-
{System.String} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read the attributes of 4 OPC-UA nodes at once, and display
the results.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "ReadMultiple.h"

namespace _EasyUAClient
{
    void ReadMultiple::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            _UAReadArgumentsPtr ReadArguments1Ptr(__uuidof(UAReadArguments));
            ReadArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10853";

            _UAReadArgumentsPtr ReadArguments2Ptr(__uuidof(UAReadArguments));
            ReadArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10845";

            _UAReadArgumentsPtr ReadArguments3Ptr(__uuidof(UAReadArguments));
            ReadArguments3Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10304";

            _UAReadArgumentsPtr ReadArguments4Ptr(__uuidof(UAReadArguments));
```

```
            ReadArguments4Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments4Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10389";

            CComSafeArray<VARIANT> arguments(4);
            arguments.SetAt(0, _variant_t((IDispatch*)ReadArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)ReadArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)ReadArguments3Ptr));
            arguments.SetAt(3, _variant_t((IDispatch*)ReadArguments4Ptr));
            CComVariant vArguments(arguments);

            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Obtain values. By default, the Value attributes of the nodes will be
read.
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->ReadMultiple(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _UAAttributeDataResultPtr ResultPtr = results[i];

                _variant_t vString;
                vString.ChangeType(VT_BSTR, &_variant_t((IDispatch*)ResultPtr-
>AttributeData));
                _tprintf(_T("results(%d).AttributeData: %s\n"), i,
(LPCTSTR)CW2CT((_bstr_t)vString));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

PHP

```
// This example shows how to read the attributes of 4 OPC-UA nodes at once, and display
the results.

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";

$ReadArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";

$ReadArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
```

```
$ReadArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10304";

$ReadArguments4 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments4->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments4->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10389";

$arguments[0] = $ReadArguments1;
$arguments[1] = $ReadArguments2;
$arguments[2] = $ReadArguments3;
$arguments[3] = $ReadArguments4;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
$results = $Client->ReadMultiple($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $attributeDataResult = $results[$i];
    if ($attributeDataResult->Succeeded)
        printf("results[%d].AttributeData: %s\n", $i, $attributeDataResult-
>AttributeData);
    else
        printf("results[%d]: *** Failure: %s\n", $i, $attributeDataResult-
>ErrorMessageBrief);
}
```

PowerScript

```
// This example shows how to read data (value, timestamps, and status code) of 3
attributes at once. In this example,
// we are reading a Value attribute of 3 different nodes, but the method can also be
used to read multiple attributes
// of the same node.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments. By default, the Value attributes of the nodes will be read.

OLEObject readArguments1
readArguments1 = CREATE OLEObject
readArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
readArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
```

```
OLEObject readArguments2
readArguments2 = CREATE OLEObject
readArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
readArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

OLEObject readArguments3
readArguments3 = CREATE OLEObject
readArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
readArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"

OLEObject readArgumentsList
readArgumentsList = CREATE OLEObject
readArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readArgumentsList.Add(readArguments1)
readArgumentsList.Add(readArguments2)
readArgumentsList.Add(readArguments3)

// Obtain attribute data.
OLEObject attributeDataResultList
attributeDataResultList = client.ReadList(readArgumentsList)

// Display results
Int i
FOR i = 0 TO attributeDataResultList.Count - 1
    OLEObject attributeDataResult
    attributeDataResult = attributeDataResultList.Item[i]
    IF attributeDataResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "AttributeData: " +
String(attributeDataResult.AttributeData) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
attributeDataResult.ErrorMessageBrief + "~r~n"
    END IF
NEXT

// Example output:
//
//AttributeData: 51 {System.Int16} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
//AttributeData: -1993984 {System.Single} @11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM;
Good
//AttributeData: Yellow% Dragon Cat) White Blue Dog# Green Banana- {System.String}
@11/6/2011 1:49:19 PM @11/6/2011 1:49:19 PM; Good
```

Visual Basic (VB 6.)

```
Rem This example shows how to read the attributes of 4 OPC-UA nodes at once, and
display the results.

Private Sub ReadMultiple_Main_Command_Click()
    OutputText = ""
```

```
    Dim ReadArguments1 As New UAReadArguments
    ReadArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"

    Dim ReadArguments2 As New UAReadArguments
    ReadArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"

    Dim ReadArguments3 As New UAReadArguments
    ReadArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10304"

    Dim ReadArguments4 As New UAReadArguments
    ReadArguments4.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments4.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10389"

    Dim arguments(3) As Variant
    Set arguments(0) = ReadArguments1
    Set arguments(1) = ReadArguments2
    Set arguments(2) = ReadArguments3
    Set arguments(3) = ReadArguments4

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Obtain values. By default, the Value attributes of the nodes will be read.
    Dim results() As Variant
    results = Client.ReadMultiple(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        Dim Result As UAAttributeDataResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "results(" & i & ").AttributeData: " &
Result.AttributeData & vbCrLf
        Else
            OutputText = OutputText & "results(" & i & ") *** Failure: " &
Result.ErrorMessageBrief & vbCrLf
        End If
    Next
End Sub
```

VBScript

```
Rem This example shows how to read the attributes of 4 OPC-UA nodes at once, and
display the results.

Option Explicit
```

```
Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10304"

Dim ReadArguments4: Set ReadArguments4 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments4.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
ReadArguments4.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10389"

Dim arguments(3)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3
Set arguments(3) = ReadArguments4

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
Dim results: results = Client.ReadMultiple(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim AttributeDataResult: Set AttributeDataResult = results(i)
    If AttributeDataResult.Succeeded Then
        WScript.Echo "results(" & i & ").AttributeData: " &
AttributeDataResult.AttributeData
    Else
        WScript.Echo "results(" & i & ") *** Failure: " &
AttributeDataResult.ErrorMessageBrief
    End If
Next
```

13.2.13.33 Examples - OPC Unified Architecture - Read multiple values

C#

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultipleValues
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain values. By default, the Value attributes of the nodes will be read.
            ValueResult[] valueResultArray = client.ReadMultipleValues(new[]
                {
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845"),
                    new UAReadArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853"),
                    new UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10855")
                });

            // Display results
            foreach (ValueResult valueResult in valueResultArray)
            {
                if (valueResult.Succeeded)
                    Console.WriteLine("Value: {0}", valueResult.Value);
                else
                    Console.WriteLine("*** Failure: {0}", valueResult.ErrorMessageBrief);
            }

            // Example output:
            //
            //Value: 8
            //Value: -8.06803E+21
            //Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
        }
    }
}
```

VB.NET

```
' This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
' to read multiple attributes of the same node.
```

```
Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultipleValues
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Obtain values. By default, the Value attributes of the nodes will be read.
            Dim valueResultArray() As ValueResult = client.ReadMultipleValues(New UAReadArguments()
_
               {
                   New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10845"),
                   New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10853"),
                   New UAReadArguments(endpointDescriptor, "nsu=http://test.org/UA/Data/;i=10855")
               }
            )

            ' Display results
            For Each valueResult As ValueResult In valueResultArray
                If valueResult.Succeeded Then
                    Console.WriteLine("Value: {0}", valueResult.Value)
                Else
                    Console.WriteLine("*** Failure: {0}", valueResult.ErrorMessageBrief)
                End If
            Next valueResult

            ' Example output:
            '
            'Value: 8
            'Value: -8.06803E+21
            'Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "ReadMultipleValues.h"

namespace _EasyUAClient
{
    void ReadMultipleValues::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
```

```
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            _UAReadArgumentsPtr ReadArguments1Ptr(__uuidof(UAReadArguments));
            ReadArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10845";

            _UAReadArgumentsPtr ReadArguments2Ptr(__uuidof(UAReadArguments));
            ReadArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10853";

            _UAReadArgumentsPtr ReadArguments3Ptr(__uuidof(UAReadArguments));
            ReadArguments3Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            ReadArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10855";

            CComSafeArray<VARIANT> arguments(3);
            arguments.SetAt(0, _variant_t((IDispatch*)ReadArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)ReadArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)ReadArguments3Ptr));
            CComVariant vArguments(arguments);

            // Obtain values. By default, the Value attributes of the nodes will be read.
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->ReadMultipleValues(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _ValueResultPtr ValueResultPtr = results[i];

                _variant_t vString;
                vString.ChangeType(VT_BSTR, &ValueResultPtr->Value);
                _tprintf(_T("Value: %s\n"), (LPCTSTR)CW2CT((_bstr_t)vString));
            }

            // Example output:
            //
            //Value: 8
            //Value: -8.06803E+21
            //Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

PHP

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$ReadArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments1->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10845";
```

```
$ReadArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments2->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10853";

$ReadArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments");
$ReadArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$ReadArguments3->NodeDescriptor->NodeId->ExpandedText = "nsu=http://test.org/UA/Data/;i=10855";

$arguments[0] = $ReadArguments1;
$arguments[1] = $ReadArguments2;
$arguments[2] = $ReadArguments3;

// Obtain values. By default, the Value attributes of the nodes will be read.
$results = $Client->ReadMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $ValueResult = $results[$i];
    if ($ValueResult->Succeeded)
        printf("Value: %s\n", $ValueResult->Value);
    else
        printf("*** Failure: %s\n", $ValueResult->ErrorMessageBrief);
}

// Example output:
//
//Value: 8
//Value: -8.06803E+21
//Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

PowerScript

```
// This example shows how to read the Value attributes of 3 different nodes at once. Using the same
method, it is also possible
// to read multiple attributes of the same node.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments. By default, the Value attributes of the nodes will be read.

OLEObject readArguments1
readArguments1 = CREATE OLEObject
readArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10845"

OLEObject readArguments2
readArguments2 = CREATE OLEObject
readArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"

OLEObject readArguments3
readArguments3 = CREATE OLEObject
```

```
readArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
readArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
readArguments3.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10855"

OLEObject readArgumentsList
readArgumentsList = CREATE OLEObject
readArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
readArgumentsList.Add(readArguments1)
readArgumentsList.Add(readArguments2)
readArgumentsList.Add(readArguments3)

// Obtain values.
OLEObject valueResultList
valueResultList = client.ReadValueList(readArgumentsList)

// Display results
Int i
FOR i = 0 TO valueResultList.Count - 1
    OLEObject valueResult
    valueResult = valueResultList.Item[i]
    IF valueResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Value: " + String(valueResult.Value) + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " + valueResult.ErrorMessageBrief
+ "~r~n"
    END IF
NEXT

// Example output:
//
//Value: 8
//Value: -8.06803E+21
//Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

Visual Basic (VB 6.)

```
Rem This example shows how to read the Value attributes of 3 different nodes at once. Using the
same method, it is also possible
Rem to read multiple attributes of the same node.

Private Sub ReadMultipleValues_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    Dim ReadArguments1 As New UAReadArguments
    ReadArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments1.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10845"

    Dim ReadArguments2 As New UAReadArguments
    ReadArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments2.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10853"

    Dim ReadArguments3 As New UAReadArguments
    ReadArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    ReadArguments3.NodeDescriptor.NodeId.expandedText = "nsu=http://test.org/UA/Data/;i=10855"

    Dim arguments(2) As Variant
    Set arguments(0) = ReadArguments1
```

```
    Set arguments(1) = ReadArguments2
    Set arguments(2) = ReadArguments3

    ' Obtain values. By default, the Value attributes of the nodes will be read.
    Dim results() As Variant
    results = Client.ReadMultipleValues(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        Dim Result As ValueResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "Value: " & Result.value & vbCrLf
        Else
            OutputText = OutputText & "*** Failure: " & Result.ErrorMessageBrief & vbCrLf
        End If
    Next

    ' Example output:
    '
    'Value: 8
    'Value: -8.06803E+21
    'Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
End Sub
```

VBScript

```
Rem This example shows how to read the Value attributes of 3 different nodes at once. Using the
same method, it is also possible
Rem to read multiple attributes of the same node.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10845"

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10853"

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer"
ReadArguments3.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10855"

Dim arguments(2)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3

' Obtain values. By default, the Value attributes of the nodes will be read.
Dim results: results = Client.ReadMultipleValues(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim ValueResult: Set ValueResult = results(i)
    If ValueResult.Succeeded Then
        WScript.Echo "Value: " & ValueResult.Value
    Else
```

```
        WScript.Echo "*** Failure: " & ValueResult.ErrorMessageBrief
    End If
Next

' Example output:
'
'Value: 8
'Value: -8.06803E+21
'Value: Strawberry Pig Banana Snake Mango Purple Grape Monkey Purple? Blueberry Lemon^
```

13.2.13.34 Examples - OPC Unified Architecture - Read nodes specified by browse paths

C#

```
// This example shows how to read the attributes of 4 OPC-UA nodes specified by browse
paths at once, and display the
// results.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadMultiple
    {
        public static void BrowsePath()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            var browsePathParser = new UABrowsePathParser {DefaultNamespaceUriString =
"http://test.org/UA/Data/"};

            // Prepare arguments
            // Note: Add error handling around the following statement if the browse
paths are not guaranteed to be
            // syntactically valid.
            var readArgumentsArray = new[]
            {
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue")),
                new UAReadArguments(endpointDescriptor,
```

```
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue")),
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/Array/UInt16Value")),
                new UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar/Int32Value"))
            };

            // Obtain attribute data.
            UAAttributeDataResult[] resultArray =
client.ReadMultiple(readArgumentsArray);

            // Display results
            for (int i = 0; i < resultArray.Length; i++)
            {
                UAAttributeDataResult attributeDataResult = resultArray[i];
                if (attributeDataResult.Succeeded)
                    Console.WriteLine($"results[{i}].AttributeData:
{attributeDataResult.AttributeData}");
                else
                    Console.WriteLine($"results[{i}]: *** Failure:
{attributeDataResult.ErrorMessageBrief}");
            }
        }

        // Example output:
        //results[0].AttributeData: 4.187603E+21 {System.Single} @2019-11-
09T14:05:46.268 @@2019-11-09T14:05:46.268; Good
        //results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268 @@2019-
11-09T14:05:46.268; Good
        //results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
        //results[3].AttributeData: 1280120396 {System.Int32} @2019-11-09T14:00:07.590
@@2019-11-09T14:05:46.268; Good
    }
}
```

VB.NET

```
' This example shows how to read the attributes of 4 OPC-UA nodes specified by browse
paths at once, and display the
' results.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadMultiple
        Public Shared Sub BrowsePath()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
```

```
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Dim browsePathParser = New UABrowsePathParser()
            browsePathParser.DefaultNamespaceUriString = "http://test.org/UA/Data/"

            ' Prepare arguments
            ' Note: Add error handling around the following statement if the browse
paths are Not guaranteed to be
            ' syntactically valid.
            Dim readArgumentsArray = New UAReadArguments() _
            {
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue")),
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue")),
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/Array/UInt16Value")),
                New UAReadArguments(endpointDescriptor,
                    browsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar/Int32Value"))
            }


            ' Obtain attribute data. By default, the Value attributes of the nodes will
be read.
            Dim resultArray() As UAAttributeDataResult =
client.ReadMultiple(readArgumentsArray)

            ' Display results
            For i As Integer = 0 To resultArray.Length - 1
                Dim attributeDataResult As UAAttributeDataResult = resultArray(i)
                If attributeDataResult.Succeeded Then
                    Console.WriteLine("results[{0}].AttributeData: {1}", i,
attributeDataResult.AttributeData)
                Else
                    Console.WriteLine("results[{0}]: *** Failure: {1}", i,
attributeDataResult.ErrorMessageBrief)
                End If
            Next i

            ' Example output:
            'results[0].AttributeData 4.187603E+21 {System.Single} @2019-11-
09T14:05:46.268 @@2019-11-09T14:05:46.268; Good
            'results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268
@@2019-11-09T14:05:46.268; Good
            'results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
            'results[3].AttributeData: 1280120396 {System.Int32} @2019-11-
09T14:00:07.590 @@2019-11-09T14:05:46.268; Good
        End Sub
```

```
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to read the attributes of 4 OPC-UA nodes specified
// by browse paths at once, and display the results.

class procedure ReadMultiple.BrowsePath;
var
  Arguments: OleVariant;
  BrowsePathParser: _UABrowsePathParser;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  ReadArguments1, ReadArguments2, ReadArguments3, ReadArguments4: _UAReadArguments;
  Result: _UAAttributeDataResult;
  Results: OleVariant;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;
  BrowsePathParser.DefaultNamespaceUriString := 'http://test.org/UA/Data/';

  ReadArguments1 := CoUAReadArguments.Create;
  ReadArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments1.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue');

  ReadArguments2 := CoUAReadArguments.Create;
  ReadArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments2.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue');

  ReadArguments3 := CoUAReadArguments.Create;
  ReadArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments3.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Static/Array/UInt16Value');

  ReadArguments4 := CoUAReadArguments.Create;
  ReadArguments4.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  //  Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
  ReadArguments4.NodeDescriptor.BrowsePath :=
BrowsePathParser.Parse('[ObjectsFolder]/Data/Static/UserScalar/Int32Value');

  Arguments := VarArrayCreate([0, 3], varVariant);
  Arguments[0] := ReadArguments1;
  Arguments[1] := ReadArguments2;
  Arguments[2] := ReadArguments3;
```

```
  Arguments[3] := ReadArguments4;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.ReadMultiple(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    Result := IInterface(Results[I]) as _UAAttributeDataResult;
    if Result.Succeeded then
      WriteLn('results[', I, '].AttributeData: ', Result.AttributeData.ToString)
    else
      WriteLn('results[', I, '] *** Failure: ', Result.ErrorMessageBrief);
  end;

  // Example output:
  //results[0].AttributeData: 4.187603E+21 {System.Single} @2019-11-09T14:05:46.268
@@2019-11-09T14:05:46.268; Good
  //results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268 @@2019-11-
09T14:05:46.268; Good
  //results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
  //results[3].AttributeData: 1280120396 {System.Int32} @2019-11-09T14:00:07.590
@@2019-11-09T14:05:46.268; Good

end;
```

VBScript

```
Rem This example shows how to read the attributes of 4 OPC-UA nodes specified by browse
paths at once, and display the
Rem results.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
BrowsePathParser.DefaultNamespaceUriString = "http://test.org/UA/Data/"

Dim ReadArguments1: Set ReadArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments1.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/FloatValue")

Dim ReadArguments2: Set ReadArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
```

```
guaranteed to be syntactically valid.
ReadArguments2.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Dynamic/Scalar/SByteValue")

Dim ReadArguments3: Set ReadArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments3.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/Array/UInt16Value")

Dim ReadArguments4: Set ReadArguments4 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAReadArguments")
ReadArguments4.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
' Note: Add error handling around the following statement if the browse path is not
guaranteed to be syntactically valid.
ReadArguments4.NodeDescriptor.BrowsePath = BrowsePathParser.Parse("
[ObjectsFolder]/Data/Static/UserScalar/Int32Value")

Dim arguments(3)
Set arguments(0) = ReadArguments1
Set arguments(1) = ReadArguments2
Set arguments(2) = ReadArguments3
Set arguments(3) = ReadArguments4

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
Dim results: results = Client.ReadMultiple(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim AttributeDataResult: Set AttributeDataResult = results(i)
    If AttributeDataResult.Succeeded Then
        WScript.Echo "results[" & i & "].AttributeData: " &
AttributeDataResult.AttributeData
    Else
        WScript.Echo "results[" & i & "] *** Failure: " &
AttributeDataResult.ErrorMessageBrief
    End If
Next

' Example output:
'results[0].AttributeData: 4.187603E+21 {System.Single} @2019-11-09T14:05:46.268
@@2019-11-09T14:05:46.268; Good
'results[1].AttributeData: -98 {System.Int16} @2019-11-09T14:05:46.268 @@2019-11-
09T14:05:46.268; Good
'results[2].AttributeData: [58] {38240, 11129, 64397, 22845, 30525, ...}
{System.Int32[]} @2019-11-09T14:00:07.543 @@2019-11-09T14:05:46.268; Good
'results[3].AttributeData: 1280120396 {System.Int32} @2019-11-09T14:00:07.590 @@2019-
11-09T14:05:46.268; Good
```

13.2.13.35 Examples - OPC Unified Architecture - Read value of specific attribute of a single node

C#

```
// This example shows how to read a value of a specific attribute of a single node, and
display it.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class ReadValue
    {
        public static void Overload2()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            UANodeDescriptor nodeDescriptor = "nsu=http://test.org/UA/Data/;i=10853";

            // Instantiate the client object
            var client = new EasyUAClient();

            // Obtain value of a DataType attribute
            object value;
            try
            {
                value = client.ReadValue(endpointDescriptor, nodeDescriptor,
UAAttributeId.DataType);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Display results
            Console.WriteLine("value type: {0}", value?.GetType());
            Console.WriteLine("value: {0}", value);
        }
    }
}
```

VB.NET

```
' This example shows how to read a value of a specific attribute of a single node, and
display it.
```

```
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class ReadValue
        Public Shared Sub Overload2()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            Dim nodeDescriptor As UANodeDescriptor =
"nsu=http://test.org/UA/Data/;i=10853"

            ' Instantiate the client object.
            Dim client = New EasyUAClient()

            ' Obtain value of a DataType attribute.
            Dim value As Object
            Try
                value = client.ReadValue(endpointDescriptor, nodeDescriptor,
UAAttributeId.DataType)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Display results
            Console.WriteLine("value type: {0}", value.GetType())
            Console.WriteLine("value: {0}", value)
        End Sub
    End Class
End Namespace
```

13.2.13.36 Examples - OPC Unified Architecture - Set application name for the client certificate

C#

```
// This example demonstrates how to set the application name for the client certificate.

using System;
using OpcLabs.BaseLib.Instrumentation;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._UAApplicationParameters
```

```
{
    class ApplicationName
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Hook static events
            EasyUAClient.LogEntry += EasyUAClientOnLogEntry;

            try
            {
                // Set the application name, which determines the subject of the client
certificate.
                // Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName =
                    "QuickOPC - CSharp example application";

                // Do something - invoke an OPC read, to trigger some loggable entries.
                var client = new EasyUAClient();
                try
                {
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853");
                }
                catch (UAException uaException)
                {
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                }

                // The certificate will be located or created in a directory similar to:
                // C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
                // and its subject will be as given by the application name.

                Console.WriteLine("Processing log entry events for 10 seconds...");
                System.Threading.Thread.Sleep(10 * 1000);

                Console.WriteLine("Done.");
            }
            finally
            {
                // Unhook static events
                EasyUAClient.LogEntry -= EasyUAClientOnLogEntry;
            }
        }

        // Event handler for the LogEntry event.
        // Print the loggable entry containing client certificate parameters.
        private static void EasyUAClientOnLogEntry(object sender, LogEntryEventArgs
logEntryEventArgs)
        {
            if (logEntryEventArgs.EventId == 161)
                Console.WriteLine(logEntryEventArgs);
```

```
        }
    }
}
```

Object Pascal

```
// This example demonstrates how to set the application name for the client certificate.

type
  TClientConfigurationEventHandlers103 = class
    procedure OnLogEntry(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _LogEntryEventArgs);
  end;

// Event handler for the LogEntry event.
// Print the loggable entry containing client certificate parameters.
procedure TClientConfigurationEventHandlers103.OnLogEntry(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _LogEntryEventArgs);
begin
  if eventArgs.EventId = 161 then
      WriteLn(eventArgs.ToString);
end;

class procedure ApplicationName.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  ClientConfiguration: TEasyUAClientConfiguration;
  ClientConfigurationEventHandlers: TClientConfigurationEventHandlers103;
  Value: OleVariant;
begin
  // The configuration object allows access to static behavior - here, the
  // shared LogEntry event.
  ClientConfiguration := TEasyUAClientConfiguration.Create(nil);
  ClientConfigurationEventHandlers := TClientConfigurationEventHandlers103.Create;
  ClientConfiguration.OnLogEntry := ClientConfigurationEventHandlers.OnLogEntry;
  ClientConfiguration.Connect;

  try
    // Set the application name, which determines the subject of the client certificate.
    // Note that this only works once in each host process.

ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName
:=
      'QuickOPC - Delphi example application';

    // Do something - invoke an OPC read, to trigger some loggable entries.
    Client := CoEasyUAClient.Create;
    try
      Value := Client.ReadValue(
        'http://opcua.demo-this.com:51211/UA/SampleServer',
        'nsu=http://test.org/UA/Data/;i=10853');
    except
      on E: EOleException do
      begin
```

```
        WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      end;
    end;

    // The certificate will be located or created in a directory similar to:
    // C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
    // and its subject will be as given by the application name.

    WriteLn('Processing log entry events for 10 seconds...');
    PumpSleep(10*1000);

    WriteLn('Done.');
  finally
    FreeAndNil(ClientConfiguration);
    FreeAndNil(ClientConfigurationEventHandlers);
  end;
end;
```

PHP

```
// This example demonstrates how to set the application name for the client certificate.

class ClientConfigurationEvents {
    // Event handler for the LogEntry event.
    // Print the loggable entry containing client certificate parameters.
    function LogEntry($Sender, $E)
    {
        if ($E->EventId = 161)
            printf("%s\n", $E);
    }
}

// The configuration object allows access to static behavior - here, the
// shared LogEntry event.
$ClientConfiguration = new COM("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration");
$ClientConfigurationEvents = new ClientConfigurationEvents();
com_event_sink($ClientConfiguration, $ClientConfigurationEvents,
"DEasyUAClientConfigurationEvents");

// Set the application name, which determines the subject of the client certificate.
// Note that this only works once in each host process.
$ClientConfiguration->SharedParameters->EngineParameters->ApplicationParameters-
>ApplicationName =
    "QuickOPC - PHP example application";

// Do something - invoke an OPC read, to trigger some loggable entries.
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
try
{
    $value = $Client->ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853");
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}

// The certificate will be located or created in a directory similar to:
```

```
// C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
// and its subject will be as given by the application name.

printf("Processing log entry events for 10 seconds...");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Done.\n");
```

VB.NET

```
' This example demonstrates how to set the application name for the client certificate.

Imports OpcLabs.BaseLib.Instrumentation
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._UAApplicationParameters
    Friend Class ApplicationName
        Public Shared Sub Main1()

            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Hook static events
            AddHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry

            Try
                ' Set the application name, which determins the subject of the client
certificate.
                ' Note that this only works once in each host process.

EasyUAClient.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName = _
                    "QuickOPC - VBNet example application"

                ' Do something - invoke an OPC read, to trigger some loggable entries.
                Dim client = New EasyUAClient()
                Try
                    client.ReadValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853")
                Catch uaException As UAException
                    Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                    Exit Sub
                End Try

                ' The certificate will be located or created in a directory similar to:
                ' C:\ProgramData\OPC Foundation\CertificateStores\MachineDefault\certs
                ' and its subject will be as given by the application name.

                Console.WriteLine("Processing log entry events for 10 seconds...")
                Threading.Thread.Sleep(10 * 1000)

                Console.WriteLine("Done.")
            Finally
                ' Unhook static events
```

```
                RemoveHandler EasyUAClient.LogEntry, AddressOf EasyUAClientOnLogEntry
            End Try

        End Sub


        ' Event handler for the LogEntry event.
        ' Print the loggable entry containing client certificate parameters.
        Private Shared Sub EasyUAClientOnLogEntry(ByVal sender As Object, ByVal
logEntryEventArgs As LogEntryEventArgs)
            If (logEntryEventArgs.EventId = 161) Then
                Console.WriteLine(logEntryEventArgs)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example demonstrates how to set the application name for the client certificate.

Option Explicit

' The configuration object allows access to static behavior.
Dim ClientConfiguration: Set ClientConfiguration =
CreateObject("OpcLabs.EasyOpc.UA.EasyUAClientConfiguration")
WScript.ConnectObject ClientConfiguration, "ClientConfiguration_"

' Set the application name, which determins the subject of the client certificate.
' Note that this only works once in each host process.
ClientConfiguration.SharedParameters.EngineParameters.ApplicationParameters.ApplicationName
= "QuickOPC - VBScript example application"

' Do something - invoke an OPC read, to trigger some loggable entries.
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
On Error Resume Next
Dim value: value = Client.ReadValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853")
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0

' The certificate will be located or created in a directory similar to:
' C:\Users\All Users\OPC Foundation\CertificateStores\UA Applications\certs\
' and its subject will be as given by the application name.

WScript.Echo "Processing log entry events for 10 seconds..."
WScript.Sleep 10*1000



' Event handler for the LogEntry event.
' Print the loggable entry containing client certificate parameters.
Sub ClientConfiguration_LogEntry(Sender, e)
    If e.EventId = 161 Then WScript.Echo e
End Sub
```

13.2.13.37 Examples - OPC Unified Architecture - Subscribe to a single node for data changes

C#

```
// This example shows how to subscribe to changes of a single monitored item and
display the value of the item with each change.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeDataChange
    {
        public static void Overload1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000);

            Console.WriteLine("Processing data change events for 20 seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("Value: {0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
```

```
}
```

VB.NET

```
' This example shows how to subscribe to changes of a single monitored item and display
the value of the item with each change.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class SubscribeDataChange
        Public Shared Sub Overload1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to subscribe to changes of a single monitored item and
display each change.
```

```
#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include "SubscribeDataChange.h"

namespace _EasyUAClient
{
    // CEasyUAClientEvents

    class CEasyUAClientEvents : public IDispEventImpl<1, CEasyUAClientEvents>
    {
    public:
    BEGIN_SINK_MAP(CEasyUAClientEvents)
        // Event handlers must have the __stdcall calling convention
        SINK_ENTRY(1, DISPID_EASYUACLIENTEVENTS_DATACHANGENOTIFICATION,
&CEasyUAClientEvents::DataChangeNotification)
    END_SINK_MAP()

    public:
        // The handler for EasyUAClient.DataChangeNotification event
        STDMETHOD(DataChangeNotification)(VARIANT varSender,
_EasyUADataChangeNotificationEventArgs* pEventArgs)
        {
            // Display the data
            // Remark: Production code would check EventArgsPtr->Exception before
accessing EventArgsPtr->AttributeData.
            _UAAttributeDataPtr AttributeDataPtr(pEventArgs->AttributeData);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(AttributeDataPtr->ToString));

            return S_OK;
        }
    };



    void SubscribeDataChange::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Hook events
            CEasyUAClientEvents* pClientEvents = new CEasyUAClientEvents();
            AtlGetObjectSourceInterface(ClientPtr, &pClientEvents->m_libid,
&pClientEvents->m_iid,
                &pClientEvents->m_wMajorVerNum, &pClientEvents->m_wMinorVerNum);
            pClientEvents->m_iid = _uuidof(DEasyUAClientEvents);
            pClientEvents->DispEventAdvise(ClientPtr, &pClientEvents->m_iid);

            //
            _tprintf(_T("Subscribing...\n"));
            ClientPtr->SubscribeDataChange(
                L"http://opcua.demo-this.com:51211/UA/SampleServer",
                L"nsu=http://test.org/UA/Data/;i=10853",
                1000);
```

```
            _tprintf(_T("Processing monitored item changed events for 1 minute...\n"));
            Sleep(60*1000);

            // Unhook events
            pClientEvents->DispEventUnadvise(ClientPtr, &pClientEvents->m_iid);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to subscribe to changes of a single monitored item
// and display each change.

type
  TClientEventHandlers121 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers121.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeDataChange.Main;
var
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers121;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers121.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  Client.SubscribeDataChange(
    'http://opcua.demo-this.com:51211/UA/SampleServer',
    'nsu=http://test.org/UA/Data/;i=10853',
    1000);

  WriteLn('Processing data change events for 1 minute...');
  PumpSleep(60*1000);
```

```
  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a single monitored item and
display each change.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
        printf("%s\n", $E->AttributeData);
        else
        printf("*** Failure : %s\n", $E->ErrorMessageBrief);
    }
}



// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$Client->SubscribeDataChange(
   "http://opcua.demo-this.com:51211/UA/SampleServer",
   "nsu=http://test.org/UA/Data/;i=10853",
    1000);

printf("Processing monitored item changed events for 1 minute...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 60);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of a single monitored item and
display each change.

' The client object, with events
'Public WithEvents Client1 As EasyUAClient

Private Sub SubscribeDataChange_Main_Command_Click()
    OutputText = ""

    Set Client1 = New EasyUAClient
```

```
    OutputText = OutputText & "Subscribing..." & vbCrLf
    Call Client1.SubscribeDataChange("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10853", 1000)

    OutputText = OutputText & "Processing data changed notification events for 1
minute..." & vbCrLf
    Pause 60000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Client1.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client1 = Nothing
End Sub

Private Sub Client1_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to changes of a single monitored item and
display each change.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Client.SubscribeDataChange "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10853", 1000

WScript.Echo "Processing monitored item changed events for 1 minute..."
WScript.Sleep 60*1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

13.2.13.38 Examples - OPC Unified Architecture - Subscribe to a single node with filter

C#

```
// This example shows how to subscribe to changes of a monitored item with data change
filter.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeDataChange
    {
        public static void Filter()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification_Filter;

            Console.WriteLine("Subscribing...");
            // Report a notification if either the StatusCode or the value change.
            // The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
and can thus be used in its place.
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000,
                UADataChangeTrigger.StatusValue);

            Console.WriteLine("Processing data change events for 20 seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification_Filter(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("Value: {0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
```

```
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of a monitored item with data change
filter.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeDataChange
        Public Shared Sub Filter()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification_Filter

            Console.WriteLine("Subscribing...")
            ' Report a notification if either the StatusCode Or the value change.
            ' The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
And can thus be used in its place.
            client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000, UADataChangeTrigger.StatusValue)

            Console.WriteLine("Processing data change events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification_Filter(ByVal sender As Object,
ByVal e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of a monitored item
// with data change filter.

type
  TClientEventHandlers120 = class
    procedure Client_DataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers120.Client_DataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
  begin
      WriteLn(eventArgs.AttributeData.ToString);
  end
  else
      WriteLn(' *** Failure: ', eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeDataChange.Filter;
const
  UADataChangeFilter_StatusValue = 1;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers120;
  DataChangeFilter: _UADataChangeFilter;
  EndpointDescriptor: string;
  MonitoringItemArguments1: _EasyUAMonitoredItemArguments;
  MonitoringParameters:  _UAMonitoringParameters;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers120.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.Client_DataChangeNotification;

  // Prepare the arguments.
  // Report a notification if either the StatusCode or the value change.
  DataChangeFilter := CoUADataChangeFilter.Create;
  DataChangeFilter.Trigger := UADataChangeFilter_StatusValue;
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.DataChangeFilter := DataChangeFilter;
  MonitoringParameters.SamplingInterval := 100;
  MonitoringItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
```

```
  MonitoringItemArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoringItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoringItemArguments1.MonitoringParameters := MonitoringParameters;

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := MonitoringItemArguments1;

  WriteLn('Subscribing...');
  Client.SubscribeMultipleMonitoredItems(arguments);

  WriteLn('Processing monitored item changed events for 20 seconds...');
  PumpSleep(20*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  PumpSleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of a monitored item
// with data change filter.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s\n", $E->AttributeData);
        else
            printf(" *** Failure: %s\n", $E->ErrorMessageBrief);
    }
}

const UADataChangeFilter_StatusValue = 1;

$EndpointDescriptor = new COM("OpcLabs.EasyOpc.UA.UAEndpointDescriptor");
$EndpointDescriptor->UrlString = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor->UrlString = "https://opcua.demo-
this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor->UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer";


// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

// Prepare the arguments.
```

```
// Report a notification if either the StatusCode or the value change.
$DataChangeFilter = new COM("OpcLabs.EasyOpc.UA.UADataChangeFilter");
$DataChangeFilter->Trigger = UADataChangeFilter_StatusValue;
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->DataChangeFilter = $DataChangeFilter;
$MonitoringParameters->SamplingInterval = 100;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;

printf("Subscribing...\n");
$Client->SubscribeMultipleMonitoredItems($arguments);

printf("Processing monitored item changed events for 20 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 20);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of a monitored item with data change
filter.

' The client object, with events
'Public WithEvents Client3 As EasyUAClient

Private Sub SubscribeDataChange_Filter_Command_Click()
    OutputText = ""

    Dim endpointDescriptor As String
    endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
    'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
    'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

    ' Instantiate the client object and hook events
    Set Client3 = New EasyUAClient

    ' Prepare the arguments.
    ' Report a notification if either the StatusCode or the value change.
    Dim DataChangeFilter As New UADataChangeFilter
    DataChangeFilter.Trigger = UADataChangeTrigger_StatusValue

    Dim MonitoringParameters As New UAMonitoringParameters
    Set MonitoringParameters.DataChangeFilter = DataChangeFilter
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments1 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments1.endpointDescriptor.UrlString = endpointDescriptor
    MonitoredItemArguments1.NodeDescriptor.NodeId.expandedText =
```

```
"nsu=http://test.org/UA/Data/;i=10853"
    Set MonitoredItemArguments1.MonitoringParameters = MonitoringParameters

    Dim arguments(0) As Variant
    Set arguments(0) = MonitoredItemArguments1

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Call Client3.SubscribeMultipleMonitoredItems(arguments)

    OutputText = OutputText & "Processing monitored item changed events for 20
seconds..." & vbCrLf
    Pause 20000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Client3.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client3 = Nothing
End Sub

Private Sub Client3_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to changes of a monitored item with data change
filter.

Option Explicit

Const UADataChangeTrigger_StatusValue = 1

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

' Prepare the arguments.
' Report a notification if either the StatusCode or the value change.
Dim DataChangeFilter: Set DataChangeFilter =
CreateObject("OpcLabs.EasyOpc.UA.UADataChangeFilter")
DataChangeFilter.Trigger = UADataChangeTrigger_StatusValue
'
Dim MonitoringParameters: Set MonitoringParameters =
```

```
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
Set MonitoringParameters.DataChangeFilter = DataChangeFilter
MonitoringParameters.SamplingInterval = 1000
'
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = endpointDescriptor
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
'
Dim arguments(0)
Set arguments(0) = MonitoredItemArguments1

WScript.Echo "Subscribing..."
Client.SubscribeMultipleMonitoredItems arguments

WScript.Echo "Processing monitored item changed events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display value
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= "*** Failure: " & e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

13.2.13.39 Examples - OPC Unified Architecture - Subscribe to all variables in an object

C#

```
// This example shows how to subscribe to changes of all data variables under a
specified object in OPC UA address space.

using System;
using System.Linq;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.AddressSpace;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
```

```
    {
        public static void AllInObject()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification_AllInObject;

            // Obtain variables under "Scalar" node
            Console.WriteLine("Browsing...");
            UANodeElementCollection nodeElementCollection;
            try
            {
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
"nsu=http://test.org/UA/Data/;ns=2;i=10787");
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
                return;
            }

            // Create array with monitored item arguments
            EasyUAMonitoredItemArguments[] monitoredItemArgumentsArray =
nodeElementCollection
                .Select(element => new EasyUAMonitoredItemArguments(null,
endpointDescriptor, element))
                .ToArray();

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(monitoredItemArgumentsArray);

            Console.WriteLine("Processing monitored item changed events for 20
seconds...");
            System.Threading.Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification_AllInObject(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
```

```
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of all data variables under a
specified object in OPC UA address space.

Imports System.Linq
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.AddressSpace
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub AllInObject()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification_AllInObject

            ' Obtain variables under "Scalar" node
            Console.WriteLine("Subscribing...")
            Dim nodeElementCollection As UANodeElementCollection
            Try
                nodeElementCollection = client.BrowseDataVariables(endpointDescriptor,
"nsu=http://test.org/UA/Data/;ns=2;i=10787")
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try

            ' Create array with monitored item arguments

            Dim monitoredItemArgumentsArray() As EasyUAMonitoredItemArguments =
nodeElementCollection.Select(Function(element) New
EasyUAMonitoredItemArguments(Nothing, endpointDescriptor, element)).ToArray()

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(monitoredItemArgumentsArray)

            Console.WriteLine("Processing monitored item changed events for 20
seconds...")
            Threading.Thread.Sleep(20 * 1000)
```

```
            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification_AllInObject(ByVal sender As
Object, ByVal e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

13.2.13.40 Examples - OPC Unified Architecture - Subscribe to multiple nodes for data changes

C#

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
```

```
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
' each change.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
```

```
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
             )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include <atlsafe.h>
#include "SubscribeMultipleMonitoredItems.h"

namespace _EasyUAClient
{
    // CEasyUAClientEvents

    class CEasyUAClientEvents : public IDispEventImpl<1, CEasyUAClientEvents>
    {
    public:
```

```
    BEGIN_SINK_MAP(CEasyUAClientEvents)
        // Event handlers must have the __stdcall calling convention
        SINK_ENTRY(1, DISPID_EASYUACLIENTEVENTS_DATACHANGENOTIFICATION,
&CEasyUAClientEvents::DataChangeNotification)
    END_SINK_MAP()

    public:
        // The handler for EasyUAClient.DataChangeNotification event
        STDMETHOD(DataChangeNotification)(VARIANT varSender,
_EasyUADataChangeNotificationEventArgs* pEventArgs)
        {
            // Display the data
            _tprintf(_T("%s: "), (LPCTSTR)CW2CT(pEventArgs->Arguments->NodeDescriptor-
>ToString));
            // Remark: Production code would check EventArgsPtr->Exception before
accessing EventArgsPtr->AttributeData.
            _UAAttributeDataPtr AttributeDataPtr(pEventArgs->AttributeData);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(AttributeDataPtr->ToString));

            return S_OK;
        }
    };



    void SubscribeMultipleMonitoredItems::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Hook events
            CEasyUAClientEvents* pClientEvents = new CEasyUAClientEvents();
            AtlGetObjectSourceInterface(ClientPtr, &pClientEvents->m_libid,
&pClientEvents->m_iid,
                &pClientEvents->m_wMajorVerNum, &pClientEvents->m_wMinorVerNum);
            pClientEvents->m_iid = _uuidof(DEasyUAClientEvents);
            pClientEvents->DispEventAdvise(ClientPtr, &pClientEvents->m_iid);

            //
            _tprintf(_T("Subscribing...\n"));

            _UAMonitoringParametersPtr
MonitoringParametersPtr(_uuidof(UAMonitoringParameters));
            MonitoringParametersPtr->SamplingInterval = 1000;

            _UAMonitoredItemArgumentsPtr
MonitoringArguments1Ptr(_uuidof(EasyUAMonitoredItemArguments));
            MonitoringArguments1Ptr->EndpointDescriptor->UrlString =
L"http://opcua.demo-this.com:51211/UA/SampleServer";
            MonitoringArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10845";
            MonitoringArguments1Ptr->MonitoringParameters = MonitoringParametersPtr;

            _UAMonitoredItemArgumentsPtr
```

```
MonitoringArguments2Ptr(_uuidof(EasyUAMonitoredItemArguments));
            MonitoringArguments2Ptr->EndpointDescriptor->UrlString =
L"http://opcua.demo-this.com:51211/UA/SampleServer";
            MonitoringArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10853";
            MonitoringArguments2Ptr->MonitoringParameters = MonitoringParametersPtr;

            _UAMonitoredItemArgumentsPtr
MonitoringArguments3Ptr(_uuidof(EasyUAMonitoredItemArguments));
            MonitoringArguments3Ptr->EndpointDescriptor->UrlString =
L"http://opcua.demo-this.com:51211/UA/SampleServer";
            MonitoringArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10855";
            MonitoringArguments3Ptr->MonitoringParameters = MonitoringParametersPtr;

            CComSafeArray<VARIANT> arguments(3);
            arguments.SetAt(0, _variant_t((IDispatch*)MonitoringArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)MonitoringArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)MonitoringArguments3Ptr));
            CComVariant vArguments(arguments);

            CComSafeArray<VARIANT> handles;
            handles.Attach(ClientPtr->SubscribeMultipleMonitoredItems(&vArguments));

            for (int i = handles.GetLowerBound(); i <= handles.GetUpperBound(); i++)
            {
                _variant_t vString;
                vString.ChangeType(VT_BSTR, &_variant_t(handles[i]));
                _tprintf(_T("handleArray(%d): %s\n"), i,
(LPCTSTR)CW2CT((_bstr_t)vString));
            }

            _tprintf(_T("Processing monitored item changed events for 10
seconds...\n"));
            Sleep(10*1000);

            _tprintf(_T("Unsubscribing...\n"));
            ClientPtr->UnsubscribeAllMonitoredItems();

            _tprintf(_T("Waiting for 5 seconds...\n"));
            Sleep(5*1000);

            // Unhook events
            pClientEvents->DispEventUnadvise(ClientPtr, &pClientEvents->m_iid);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

type
```

```
  TClientEventHandlers123 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers123.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleMonitoredItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers123;
  Handle: Cardinal;
  HandleArray: OleVariant;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers123.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
```

```
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
// each change.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}



// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
```

```
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
Rem each change.

' The client object, with events
'Public WithEvents Client2 As EasyUAClient

Private Sub SubscribeMultipleMonitoredItems_Main_Command_Click()
    OutputText = ""

    Set Client2 = New EasyUAClient

    OutputText = OutputText & "Subscribing..." & vbCrLf
```

```
    Dim MonitoringParameters As New UAMonitoringParameters
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments1 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"
    Set MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments1.SetState ("Item1")

    Dim MonitoredItemArguments2 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"
    Set MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments2.SetState ("Item2")

    Dim MonitoredItemArguments3 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10855"
    Set MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments3.SetState ("Item3")

    Dim arguments(2) As Variant
    Set arguments(0) = MonitoredItemArguments1
    Set arguments(1) = MonitoredItemArguments2
    Set arguments(2) = MonitoredItemArguments3
    Dim handleArray As Variant
    handleArray = Client2.SubscribeMultipleMonitoredItems(arguments)

    Dim i As Long: For i = LBound(handleArray) To UBound(handleArray)
        OutputText = OutputText & "handleArray(" & i & "): " & handleArray(i) & vbCrLf
    Next

    OutputText = OutputText & "Processing monitored item changed events for 10
seconds..." & vbCrLf
    Pause 10000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Call Client2.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client2 = Nothing
    OutputText = OutputText & "Done." & vbCrLf
End Sub

Private Sub Client2_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
```

```
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

VBScript

```
Rem This example shows how to subscribe to changes of multiple monitored items and
display the value of the monitored item with
Rem each change.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
MonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000
```

```
WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

13.2.13.41 Examples - OPC Unified Architecture - Subscribe to multiple nodes with filter

C#

```
// This example shows how to subscribe to changes of multiple monitored items and use a
data change filter.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class SubscribeMultipleMonitoredItems
    {
        public static void Filter()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification_Filter;

            Console.WriteLine("Subscribing...");
            // Report a notification if either the StatusCode or the value change.
            // The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
and can thus be used in its place.
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845",
                        new UAMonitoringParameters(1000,
```

```
UADataChangeTrigger.StatusValue)),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853",
                        new UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855",
                        new UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue))
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification_Filter(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to changes of multiple monitored items and use a
data change filter.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class SubscribeMultipleMonitoredItems
        Public Shared Sub Filter()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
```

```
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification_Filter

            Console.WriteLine("Subscribing...")
            ' Report a notification if either the StatusCode Or the value change.
            ' The UADataChangeTrigger has an implicit conversion to UADataChangeFilter
And can thus be used in its place.
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                {
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", New UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", New UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue)),
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", New UAMonitoringParameters(1000,
UADataChangeTrigger.StatusValue))
                }
             )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification_Filter(ByVal sender As Object,
ByVal e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to subscribe to changes of multiple monitored items
// and use a data change filter.

type
  TClientEventHandlers122 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
```

```
  end;

procedure TClientEventHandlers122.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure SubscribeMultipleMonitoredItems.Filter;
const
  UADataChangeFilter_StatusValue = 1;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers122;
  DataChangeFilter: _UADataChangeFilter;
  HandleArray: OleVariant;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers122.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  // Prepare the arguments.
  // Report a notification if either the StatusCode or the value change.
  DataChangeFilter := CoUADataChangeFilter.Create;
  DataChangeFilter.Trigger := UADataChangeFilter_StatusValue;

  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.DataChangeFilter := DataChangeFilter;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
```

```
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  WriteLn('Subscribing...');
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(
    Client.SubscribeMultipleMonitoredItems(Arguments));

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to subscribe to changes of multiple monitored items
// and use a data change filter.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}

const UADataChangeFilter_StatusValue = 1;

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

// Prepare the arguments.
// Report a notification if either the StatusCode or the value change.
$DataChangeFilter = new COM("OpcLabs.EasyOpc.UA.UADataChangeFilter");
$DataChangeFilter->Trigger = UADataChangeFilter_StatusValue;
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->DataChangeFilter = $DataChangeFilter;
```

```
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;

printf("Subscribing...\n");
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

Visual Basic (VB 6.)

```
Rem This example shows how to subscribe to changes of multiple monitored items
Rem and use a data change filter.

' The client object, with events
'Public WithEvents Client4 As EasyUAClient

Private Sub SubscribeMultipleMonitoredItems_Filter_Command_Click()
    OutputText = ""

    ' Instantiate the client object and hook events
    Set Client4 = New EasyUAClient
```

```
    ' Prepare the arguments.
    ' Report a notification if either the StatusCode or the value change.
    Dim DataChangeFilter As New UADataChangeFilter
    DataChangeFilter.Trigger = UADataChangeTrigger_StatusValue
    '
    Dim MonitoringParameters As New UAMonitoringParameters
    Set MonitoringParameters.DataChangeFilter = DataChangeFilter
    MonitoringParameters.SamplingInterval = 1000

    Dim MonitoredItemArguments1 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10845"
    Set MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments1.SetState ("Item1")

    Dim MonitoredItemArguments2 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10853"
    Set MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments2.SetState ("Item2")

    Dim MonitoredItemArguments3 As New EasyUAMonitoredItemArguments
    MonitoredItemArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    MonitoredItemArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10855"
    Set MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
    MonitoredItemArguments3.SetState ("Item3")

    Dim arguments(2) As Variant
    Set arguments(0) = MonitoredItemArguments1
    Set arguments(1) = MonitoredItemArguments2
    Set arguments(2) = MonitoredItemArguments3

    OutputText = OutputText & "Subscribing..." & vbCrLf
    Dim handleArray As Variant
    handleArray = Client4.SubscribeMultipleMonitoredItems(arguments)

    Dim i As Long: For i = LBound(handleArray) To UBound(handleArray)
        OutputText = OutputText & "handleArray(" & i & "): " & handleArray(i) & vbCrLf
    Next

    OutputText = OutputText & "Processing monitored item changed events for 10
seconds..." & vbCrLf
    Pause 10000

    OutputText = OutputText & "Unsubscribing..." & vbCrLf
    Call Client4.UnsubscribeAllMonitoredItems

    OutputText = OutputText & "Waiting for 5 seconds..." & vbCrLf
    Pause 5000

    Set Client4 = Nothing
```

```
    OutputText = OutputText & "Done." & vbCrLf
End Sub

Private Sub Client4_DataChangeNotification(ByVal sender As Variant, ByVal EventArgs As
EasyUADataChangeNotificationEventArgs)
    ' Display the data
    If EventArgs.Exception Is Nothing Then
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.AttributeData & vbCrLf
    Else
        OutputText = OutputText & "[" & EventArgs.arguments.State & "] " &
EventArgs.arguments.NodeDescriptor & ": " & EventArgs.ErrorMessageBrief & vbCrLf
    End If
End Sub
```

13.2.13.42 Examples - OPC Unified Architecture - Try to parse a relative browse path

C#

```
// Attempts to parse a relative OPC-UA browse path and displays its elements.

using System;
using OpcLabs.BaseLib;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class TryParseRelative
    {
        public static void Main1()
        {
            var browsePathElements = new UABrowsePathElementCollection();

            var browsePathParser = new UABrowsePathParser();
            IStringParsingError stringParsingError =
browsePathParser.TryParseRelative("/Data.Dynamic.Scalar.CycleComplete",
browsePathElements);

            // Display results
            if (stringParsingError != null)
            {
                Console.WriteLine("*** Error: {0}", stringParsingError);
                return;
            }

            foreach (UABrowsePathElement browsePathElement in browsePathElements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // /Data
```

```
            // .Dynamic
            // .Scalar
            // .CycleComplete
        }
    }
}
```

Object Pascal

```
// Attempts to parse a relative OPC-UA browse path and displays its elements.

class procedure TryParseRelative.Main;
var
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathElements: _UABrowsePathElementCollection;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
  StringParsingError: _StringParsingError;
begin
  BrowsePathElements := CoUABrowsePathElementCollection.Create;

  BrowsePathParser := CoUABrowsePathParser.Create;
  StringParsingError :=
BrowsePathParser.TryParseRelative('/Data.Dynamic.Scalar.CycleComplete',
BrowsePathElements);

  // Display results
  if StringParsingError <> nil then
  begin
    WriteLn('*** Error: ', StringParsingError.ToString);
    Exit;
  end;

  ElementEnumerator := BrowsePathElements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // /Data
  // .Dynamic
  // .Scalar
  // .CycleComplete

end;
```

PHP

```
// Attempts to parse a relative OPC-UA browse path and displays its elements.

$BrowsePathElements = new
COM("OpcLabs.EasyOpc.UA.Navigation.UABrowsePathElementCollection");
```

```
$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

$StringParsingError = $BrowsePathParser-
>TryParseRelative("/Data.Dynamic.Scalar.CycleComplete", $BrowsePathElements);

// Display results
if (!is_null($StringParsingError)) {
    printf("*** Error: %s\n", $StringParsingError);
    exit();
}

printf("StartingNodeId: %s\n", $BrowsePath->StartingNodeId);

printf("Elements:\n");

for ($i = 0; $i < $BrowsePathElements->Count; $i++)
{
    printf("%s\n", $BrowsePathElements[$i]);
}

// Example output:
// /Data
// .Dynamic
// .Scalar
// .CycleComplete
```

VB.NET

```
' Attempts to parse a relative OPC-UA browse path and displays its elements.

Imports System
Imports OpcLabs.BaseLib
Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class TryParseRelative
        Public Shared Sub Main1()
            Dim browsePathElements = New UABrowsePathElementCollection()

            Dim browsePathParser = New UABrowsePathParser()
            Dim stringParsingError As IStringParsingError =
browsePathParser.TryParseRelative("/Data.Dynamic.Scalar.CycleComplete",
browsePathElements)

            ' Display results
            If Not stringParsingError Is Nothing Then
                Console.WriteLine("*** Error: {0}", stringParsingError)
                Exit Sub
            End If

            For Each browsePathElement As UABrowsePathElement In browsePathElements
                Console.WriteLine(browsePathElement)
            Next browsePathElement
```

```
            ' Example output:
            ' /Data
            ' .Dynamic
            ' .Scalar
            ' .CycleComplete
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Attempts to parse a relative OPC-UA browse path and displays its elements.

Option Explicit

Dim BrowsePathElements: Set BrowsePathElements =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.UABrowsePathElementCollection")

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
Dim StringParsingError: Set StringParsingError =
BrowsePathParser.TryParseRelative("/Data.Dynamic.Scalar.CycleComplete",
BrowsePathElements)

' Display results
If Not (StringParsingError Is Nothing) Then
    WScript.Echo "*** Error: " & StringParsingError
    WScript.Quit
End If

Dim BrowsePathElement: For Each BrowsePathElement In BrowsePathElements
    WScript.Echo BrowsePathElement
Next
```

13.2.13.43 Examples - OPC Unified Architecture - Try to parse an absolute browse path

C#

```
// Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

using System;
using OpcLabs.BaseLib;
using OpcLabs.EasyOpc.UA.Navigation;
using OpcLabs.EasyOpc.UA.Navigation.Parsing;

namespace UADocExamples._UABrowsePathParser
{
    class TryParse
    {
        public static void Main1()
```

```
        {
            var browsePathParser = new UABrowsePathParser();

            IStringParsingError stringParsingError = browsePathParser.TryParse(
                "[ObjectsFolder]/Data/Static/UserScalar",
                out UABrowsePath browsePath);

            // Display results
            if (stringParsingError != null)
            {
                Console.WriteLine("*** Error: {0}", stringParsingError);
                return;
            }

            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId);

            foreach (UABrowsePathElement browsePathElement in browsePath.Elements)
                Console.WriteLine(browsePathElement);

            // Example output:
            // StartingNodeId: ObjectsFolder
            // /Data
            // /Static
            // /UserScalar
        }
    }
}
```

Object Pascal

```
// Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

class procedure TryParse.Main;
var
  BrowsePath: _UABrowsePath;
  BrowsePathElement: _UABrowsePathElement;
  BrowsePathParser: OpcLabs_EasyOpcUA_TLB._UABrowsePathParser;
  BrowsePathResult: OleVariant;
  Count: Cardinal;
  Element: OleVariant;
  ElementEnumerator: IEnumVariant;
  StringParsingError: _StringParsingError;
begin
  BrowsePathParser := CoUABrowsePathParser.Create;

  StringParsingError :=
BrowsePathParser.TryParse('[ObjectsFolder]/Data/Static/UserScalar', BrowsePathResult);

  // Display results
  if StringParsingError <> nil then
  begin
    WriteLn('*** Error: ', StringParsingError.ToString);
    Exit;
  end;

  BrowsePath := IUnknown(BrowsePathResult) as _UABrowsePath;
```

```
  WriteLn('StartingNodeId: ', BrowsePath.StartingNodeId.ToString);

  WriteLn('Elements:');
  ElementEnumerator := BrowsePath.Elements.GetEnumerator;
  while (ElementEnumerator.Next(1, Element, Count) = S_OK) do
  begin
    BrowsePathElement := IUnknown(Element) as _UABrowsePathElement;
    WriteLn(BrowsePathElement.ToString);
  end;

  // Example output:
  // StartingNodeId: ObjectsFolder
  // Elements:
  // /Data
  // /Static
  // /UserScalar

end;
```

PHP

```
// Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

$BrowsePath = new COM("OpcLabs.EasyOpc.UA.Navigation.UABrowsePath");

$BrowsePathParser = new
COM("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser");

$StringParsingError = $BrowsePathParser->TryParse("
[ObjectsFolder]/Data/Static/UserScalar", $BrowsePath);

// Display results
if (!is_null($StringParsingError)) {
    printf("*** Error: %s\n", $StringParsingError);
    exit();
}

printf("StartingNodeId: %s\n", $BrowsePath->StartingNodeId);

printf("Elements:\n");

for ($i = 0; $i < $BrowsePath->Elements->Count; $i++)
{
    printf("%s\n", $BrowsePath->Elements[$i]);
}

// Example output:
// StartingNodeId: ObjectsFolder
// Elements:
// /Data
// /Static
// /UserScalar
```

VB.NET

```
' Attempts to parses an absolute  OPC-UA browse path and displays its starting node and
elements.

Imports System
Imports OpcLabs.BaseLib
Imports OpcLabs.EasyOpc.UA.Navigation
Imports OpcLabs.EasyOpc.UA.Navigation.Parsing

Namespace UADocExamples._UABrowsePathParser
    Friend Class TryParse
        Public Shared Sub Main1()
            Dim browsePathParser = New UABrowsePathParser()

            Dim browsePath As UABrowsePath = Nothing
            Dim stringParsingError As IStringParsingError = browsePathParser.TryParse("
[ObjectsFolder]/Data/Static/UserScalar", browsePath)

            ' Display results
            If Not stringParsingError Is Nothing Then
                Console.WriteLine("*** Error: {0}", stringParsingError)
                Exit Sub
            End If

            Console.WriteLine("StartingNodeId: {0}", browsePath.StartingNodeId)

            For Each browsePathElement As UABrowsePathElement In browsePath.Elements
                Console.WriteLine(browsePathElement)
            Next browsePathElement

            ' Example output:
            ' StartingNodeId: ObjectsFolder
            ' /Data
            ' /Static
            ' /UserScalar
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem Attempts to parses an absolute  OPC-UA browse path and displays its starting node
and elements.

Option Explicit

Dim BrowsePathParser: Set BrowsePathParser =
CreateObject("OpcLabs.EasyOpc.UA.Navigation.Parsing.UABrowsePathParser")
Dim BrowsePath
Dim StringParsingError: Set StringParsingError = BrowsePathParser.TryParse("
[ObjectsFolder]/Data/Static/UserScalar", BrowsePath)

' Display results
If Not (StringParsingError Is Nothing) Then
    WScript.Echo "*** Error: " & StringParsingError
    WScript.Quit
End If
```

```
WScript.Echo "StartingNodeId: " & BrowsePath.StartingNodeId

WScript.Echo "Elements:"
Dim BrowsePathElement: For Each BrowsePathElement In BrowsePath.Elements
    WScript.Echo BrowsePathElement
Next
```

13.2.13.44 Examples - OPC Unified Architecture - Unsubscribe from a single monitored item

C#

```
// This example shows how unsubscribe from changes of a single monitored item.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class UnsubscribeMonitoredItem
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int handle = client.SubscribeDataChange(
                endpointDescriptor,
                "nsu=http://test.org/UA/Data/;i=10853",
                1000);

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeMonitoredItem(handle);

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
```

```
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}", e.AttributeData.Value);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how unsubscribe from changes of a single monitored item.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class UnsubscribeMonitoredItem
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handle As Integer =
                client.SubscribeDataChange(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000)

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeMonitoredItem(handle)

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("Value: {0}", e.AttributeData.Value)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
```

```
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how unsubscribe from changes of a single monitored item.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client= CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim handle: handle = Client.SubscribeDataChange("http://opcua.demo-
this.com:51211/UA/SampleServer", "nsu=http://test.org/UA/Data/;i=10853", 1000)

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10*1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeMonitoredItem handle

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo display
End Sub
```

13.2.13.45 Examples - OPC Unified Architecture - Unsubscribe from all monitored items

C#

```
// This example shows how to unsubscribe from changes of all items.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class UnsubscribeAllMonitoredItems
    {
```

```
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllMonitoredItems();

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

Object Pascal

```
// This example shows how to unsubscribe from changes of all items.

type
  TClientEventHandlers126 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
```

```
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers126.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure UnsubscribeAllMonitoredItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers126;
  EndpointDescriptor: string;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';

  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers126.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := EndpointDescriptor;
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
```

```
  Arguments[2] := MonitoredItemArguments3;

  Client.SubscribeMultipleMonitoredItems(Arguments);

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to unsubscribe from changes of all items.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
    }
}

$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";

// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
```

```
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = $EndpointDescriptor;
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("Unsubscribing...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

VB.NET

```
' This example shows how to unsubscribe from changes of all items.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class UnsubscribeAllMonitoredItems
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            client.SubscribeMultipleMonitoredItems(New EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
```

```
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
             )

            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllMonitoredItems()

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to unsubscribe from changes of all items.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
MonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
```

```
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

13.2.13.46 Examples - OPC Unified Architecture - Unsubscribe from just some monitored items

Object Pascal

```
// This example shows how to unsubscribe from changes of just some monitored
// items.

type
  TClientEventHandlers128 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers128.OnDataChangeNotification(
```

```
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure UnsubscribeMultipleMonitoredItems.Some;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers128;
  Handle: Cardinal;
  HandleArray: OleVariant;
  HandleSafeArray: PSafeArray;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
  SelectedHandles: OleVariant;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers128.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn;
  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
```

```
  Arguments[2] := MonitoredItemArguments3;

  HandleSafeArray := Client.SubscribeMultipleMonitoredItems(Arguments);
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(HandleSafeArray);

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn;
  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn;
  WriteLn('Unsubscribing from 2 monitored items...');
  SelectedHandles := VarArrayCreate([0, 1], varVariant);
  // We will unsubscribe from the first and third monitored item we have
  // previously subscribed to.
  SelectedHandles[0] := HandleArray[0];
  SelectedHandles[1] := HandleArray[2];
  Client.UnsubscribeMultipleMonitoredItems(SelectedHandles);

  WriteLn;
  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn;
  WriteLn('Unsubscribing from all remaining monitored items...');
  Client.UnsubscribeAllMonitoredItems;

  WriteLn;
  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);

  WriteLn;
  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

PHP

```
// This example shows how to unsubscribe from changes of just some monitored
// items.

class ClientEvents {
    function DataChangeNotification($Sender, $E)
    {
        // Display the data
        if ($E->Succeeded)
            printf("%s: %s\n", $E->Arguments->NodeDescriptor, $E->AttributeData);
        else
            printf("%s *** Failure: %s\n", $E->Arguments->NodeDescriptor, $E-
>ErrorMessageBrief);
```

```
    }
}



// Instantiate the client object and hook events
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");
$ClientEvents = new ClientEvents();
com_event_sink($Client, $ClientEvents, "DEasyUAClientEvents");

printf("\n");
printf("Subscribing...\n");
$MonitoringParameters = new COM("OpcLabs.EasyOpc.UA.UAMonitoringParameters");
$MonitoringParameters->SamplingInterval = 1000;
$MonitoredItemArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845";
$MonitoredItemArguments1->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853";
$MonitoredItemArguments2->MonitoringParameters = $MonitoringParameters;
$MonitoredItemArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments");
$MonitoredItemArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$MonitoredItemArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855";
$MonitoredItemArguments3->MonitoringParameters = $MonitoringParameters;
$arguments[0] = $MonitoredItemArguments1;
$arguments[1] = $MonitoredItemArguments2;
$arguments[2] = $MonitoredItemArguments3;
$handleArray = $Client->SubscribeMultipleMonitoredItems($arguments);

for ($i = 0; $i < count($handleArray); $i++)
{
    printf("handleArray[%d]: %d\n", $i, $handleArray[$i]);
}

printf("\n");
printf("Processing monitored item changed events for 10 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("\n");
printf("Unsubscribing from 2 monitored items...\n");
$SelectedHandles[0] = $handleArray[0];
$SelectedHandles[1] = $handleArray[2];
$Client->UnsubscribeMultipleMonitoredItems($SelectedHandles);

printf("\n");
printf("Processing monitored item changed events for 10 seconds...\n");
```

```
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 10);

printf("\n");
printf("Unsubscribing from all remaining monitored items...\n");
$Client->UnsubscribeAllMonitoredItems;

printf("\n");
printf("Waiting for 5 seconds...\n");
$startTime = time(); do { com_message_pump(1000); } while (time() < $startTime + 5);
```

13.2.13.47 Examples - OPC Unified Architecture - Unsubscribe from multiple monitored items

C#

```
// This example shows how to unsubscribe from changes of multiple items.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class UnsubscribeMultipleMonitoredItems
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object and hook events
            var client = new EasyUAClient();
            client.DataChangeNotification += client_DataChangeNotification;

            Console.WriteLine("Subscribing...");
            int[] handleArray = client.SubscribeMultipleMonitoredItems(new[]
                {
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10845", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10853", 1000),
                    new EasyUAMonitoredItemArguments(null, endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10855", 1000)
                });

            Console.WriteLine("Processing monitored item changed events for 10
seconds...");
            System.Threading.Thread.Sleep(10 * 1000);
```

```
            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeMultipleMonitoredItems(handleArray);

            Console.WriteLine("Waiting for 5 seconds...");
            System.Threading.Thread.Sleep(5 * 1000);
        }

        static void client_DataChangeNotification(object sender,
EasyUADataChangeNotificationEventArgs e)
        {
            // Display value
            if (e.Succeeded)
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value);
            else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how to unsubscribe from changes of multiple items.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class UnsubscribeMultipleMonitoredItems
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object and hook events
            Dim client = New EasyUAClient()
            AddHandler client.DataChangeNotification, AddressOf
client_DataChangeNotification

            Console.WriteLine("Subscribing...")
            Dim handleArray() As Integer = client.SubscribeMultipleMonitoredItems(New
EasyUAMonitoredItemArguments() _
                { _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10845", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10853", 1000), _
                    New EasyUAMonitoredItemArguments(Nothing, endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10855", 1000) _
                } _
            )
```

```
            Console.WriteLine("Processing monitored item changed events for 10
seconds...")
            Threading.Thread.Sleep(10 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeMultipleMonitoredItems(handleArray)

            Console.WriteLine("Waiting for 5 seconds...")
            Threading.Thread.Sleep(5 * 1000)
        End Sub

        Private Shared Sub client_DataChangeNotification(ByVal sender As Object, ByVal
e As EasyUADataChangeNotificationEventArgs)
            ' Display value
            If e.Succeeded Then
                Console.WriteLine("{0}: {1}", e.Arguments.NodeDescriptor,
e.AttributeData.Value)
            Else
                Console.WriteLine("{0} *** Failure: {1}", e.Arguments.NodeDescriptor,
e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to unsubscribe from changes of multiple items.

type
  TClientEventHandlers127 = class
    procedure OnDataChangeNotification(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataChangeNotificationEventArgs);
  end;

procedure TClientEventHandlers127.OnDataChangeNotification(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataChangeNotificationEventArgs);
begin
  // Display the data
  if eventArgs.Succeeded then
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ': ',
      eventArgs.AttributeData.ToString)
  else
      WriteLn(eventArgs.Arguments.NodeDescriptor.ToString, ' *** Failure: ',
      eventArgs.ErrorMessageBrief);
end;

class procedure UnsubscribeMultipleMonitoredItems.Main;
var
  Arguments: OleVariant;
  Client: TEasyUAClient;
  ClientEventHandlers: TClientEventHandlers127;
  Handle: Cardinal;
```

```
  HandleArray: OleVariant;
  HandleSafeArray: PSafeArray;
  I: Cardinal;
  MonitoredItemArguments1, MonitoredItemArguments2, MonitoredItemArguments3:
    _EasyUAMonitoredItemArguments;
  MonitoringParameters: _UAMonitoringParameters;
begin
  // Instantiate the client object and hook events
  Client := TEasyUAClient.Create(nil);
  ClientEventHandlers := TClientEventHandlers127.Create;
  Client.OnDataChangeNotification := ClientEventHandlers.OnDataChangeNotification;

  WriteLn('Subscribing...');
  MonitoringParameters := CoUAMonitoringParameters.Create;
  MonitoringParameters.SamplingInterval := 1000;
  MonitoredItemArguments1 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10845';
  MonitoredItemArguments1.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments2 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10853';
  MonitoredItemArguments2.MonitoringParameters := MonitoringParameters;
  MonitoredItemArguments3 := CoEasyUAMonitoredItemArguments.Create;
  MonitoredItemArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10855';
  MonitoredItemArguments3.MonitoringParameters := MonitoringParameters;
  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := MonitoredItemArguments1;
  Arguments[1] := MonitoredItemArguments2;
  Arguments[2] := MonitoredItemArguments3;

  HandleSafeArray := Client.SubscribeMultipleMonitoredItems(Arguments);
  TVarData(HandleArray).VType := varArray or varVariant;
  TVarData(HandleArray).VArray := PVarArray(HandleSafeArray);

  for I := VarArrayLowBound(HandleArray, 1) to VarArrayHighBound(HandleArray, 1) do
  begin
      Handle := Cardinal(HandleArray[I]);
      WriteLn('HandleArray[', I, ']: ', Handle);
  end;

  WriteLn('Processing monitored item changed events for 10 seconds...');
  PumpSleep(10*1000);

  WriteLn('Unsubscribing...');
  Client.UnsubscribeMultipleMonitoredItems(HandleArray);

  WriteLn('Waiting for 5 seconds...');
  Sleep(5*1000);
```

```
  WriteLn('Finished.');
  FreeAndNil(Client);
  FreeAndNil(ClientEventHandlers);
end;
```

VBScript

```
Rem This example shows how to unsubscribe from changes of multiple items.

Option Explicit

' Instantiate the client object and hook events
Dim Client: Set Client= CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")
WScript.ConnectObject Client, "Client_"

WScript.Echo "Subscribing..."
Dim MonitoringParameters: Set MonitoringParameters =
CreateObject("OpcLabs.EasyOpc.UA.UAMonitoringParameters")
MonitoringParameters.SamplingInterval = 1000
Dim MonitoredItemArguments1: Set MonitoredItemArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10845"
MonitoredItemArguments1.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments2: Set MonitoredItemArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10853"
MonitoredItemArguments2.MonitoringParameters = MonitoringParameters
Dim MonitoredItemArguments3: Set MonitoredItemArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.EasyUAMonitoredItemArguments")
MonitoredItemArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
MonitoredItemArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10855"
MonitoredItemArguments3.MonitoringParameters = MonitoringParameters
Dim arguments(2)
Set arguments(0) = MonitoredItemArguments1
Set arguments(1) = MonitoredItemArguments2
Set arguments(2) = MonitoredItemArguments3
Dim handleArray: handleArray = Client.SubscribeMultipleMonitoredItems(arguments)

Dim i: For i = LBound(handleArray) To UBound(handleArray)
    WScript.Echo "handleArray(" & i & "): " & handleArray(i)
Next

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from two monitored items..."
Dim handles1(1)
handles1(0) = handleArray(1)
handles1(1) = handleArray(2)
```

```
Client.UnsubscribeMultipleMonitoredItems handles1

WScript.Echo "Processing monitored item changed events for 10 seconds..."
WScript.Sleep 10 * 1000

WScript.Echo "Unsubscribing from all remaining items..."
Client.UnsubscribeAllMonitoredItems

WScript.Echo "Waiting for 5 seconds..."
WScript.Sleep 5 * 1000



Sub Client_DataChangeNotification(Sender, e)
    ' Display the data
    Dim display: If e.Exception Is Nothing Then display = e.AttributeData Else display
= e.ErrorMessageBrief
    WScript.Echo e.Arguments.NodeDescriptor & ":" & display
End Sub
```

13.2.13.48 Examples - OPC Unified Architecture - Write a ByteString

C#

```
// This example shows how to write a value into a single node that is of type
ByteString.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void ByteString()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10230",
```

```
                    new byte[] {11, 22, 33, 44, 55});
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single node that is of type
ByteString.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
        Public Shared Sub ByteString()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10230", New Byte() {11, 22, 33, 44, 55})
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to write a value into a single node that is of type
ByteString.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlcom.h>
#include <atlsafe.h>
#include "WriteValue.h"

namespace _EasyUAClient
```

```
{
    void WriteValue::ByteString()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Prepare the value to be written
            CComSafeArray<BYTE> array(5);
            array.SetAt(0, 11);
            array.SetAt(1, 22);
            array.SetAt(2, 33);
            array.SetAt(3, 44);
            array.SetAt(4, 55);

            const CComVariant value(array);

            // Perform the operation
            ClientPtr->WriteValue(
                L"http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
                L"nsu=http://test.org/UA/Data/;i=10230",
                value);
        }
        // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to write a value into a single node that is of type
ByteString.

class procedure WriteValue.ByteString;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  Values: OleVariant;
begin
  Values := VarArrayCreate([0, 4], varByte);
  Values[0] := 11;
  Values[1] := 22;
  Values[2] := 33;
  Values[3] := 44;
  Values[4] := 55;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify value of a node
  try
    Client.WriteValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer', // or 'opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer'
      'nsu=http://test.org/UA/Data/;i=10230', Values);
```

```
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PHP

```
// This example shows how to write a value into a single node that is of type
ByteString.

$Values[0] = 11;
$Values[1] = 22;
$Values[2] = 33;
$Values[3] = 44;
$Values[4] = 55;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify value of a node
try
{
    $Client->WriteValue(
        "http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        "nsu=http://test.org/UA/Data/;i=10230",
        $Values);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}
```

Visual Basic (VB 6.)

```
Rem This example shows how to write a value into a single node that is of type
ByteString.

Private Sub WriteValue_ByteString_Command_Click()
    OutputText = ""

    Dim Values(4) As Byte
    Values(0) = 11
    Values(1) = 22
    Values(2) = 33
    Values(3) = 44
    Values(4) = 55

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
```

```
    On Error Resume Next
    Call Client.WriteValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10230", Values)
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

End Sub
```

13.2.13.49 Examples - OPC Unified Architecture - Write a single value

C#

```
// This example shows how to write a value into a single node.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 12345);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single node.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 12345)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to write a value into a single node.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "WriteValue.h"

namespace _EasyUAClient
{
    void WriteValue::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            // Perform the operation
            ClientPtr->WriteValue(
                L"http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
                L"nsu=http://test.org/UA/Data/;i=10221",
```

```
                12345);
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to write a value into a single node.

class procedure WriteValue.Main;
var
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
begin
  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Perform the operation
  try
    Client.WriteValue(
      'http://opcua.demo-this.com:51211/UA/SampleServer', // or 'opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer'
      'nsu=http://test.org/UA/Data/;i=10221',
      12345);
  except
    on E: EOleException do
    begin
      WriteLn(Format('*** Failure: %s', [E.GetBaseException.Message]));
      Exit;
    end;
  end;

end;
```

PHP

```
// This example shows how to write a value into a single node.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Perform the operation
try
{
    $Client->WriteValue(
        "http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        "nsu=http://test.org/UA/Data/;i=10221",
        12345);
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}
```

PowerScript

```
// This example shows how to write a value into a single node.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Modify value of a node
TRY
    client.WriteValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", 12345)
CATCH (OLERuntimeError oleRuntimeError)
    mle_outputtext.Text = mle_outputtext.Text + "*** Failure: " +
oleRuntimeError.Description + "~r~n"
    RETURN
END TRY

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Python

```
# This example shows how to write a value into a single node.

import win32com.client

# Instantiate the client object
client = win32com.client.Dispatch('OpcLabs.EasyOpc.UA.EasyUAClient')

# Perform the operation
client.WriteValue(
    'http://opcua.demo-this.com:51211/UA/SampleServer', # or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
    'nsu=http://test.org/UA/Data/;i=10221',
    12345)
```

Visual Basic (VB 6.)

```
Rem This example shows how to write a value into a single node.

Private Sub WriteValue_Main_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Perform the operation
    On Error Resume Next
    Call Client.WriteValue("http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", 12345)
    If Err.Number <> 0 Then
        OutputText = OutputText & "*** Failure: " & Err.Source & ": " & Err.Description
```

```
& vbCrLf
        Exit Sub
    End If
    On Error GoTo 0

End Sub
```

VBScript

```
Rem This example shows how to write a value into a single node.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Perform the operation
On Error Resume Next
Client.WriteValue "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", 12345 ' or "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

13.2.13.50 Examples - OPC Unified Architecture - Write and specify data type

The example below writes a single value, specifying the type to be written using a .NET Type.

Note that this example uses an overload of the WriteValue method that is not available in COM. If you are calling from a COM language or tool, use the WriteMultipleValues method instead. An example for that is given further down.

C#

```
// This example shows how to write a value into a single node, specifying a type
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the server reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// For a selected subset of most common types, you can also use a different overload of
the WriteValue method, and specify
// a value from the TypeCode enumeration instead.

using System;
```

```
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
        public static void Type()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10221",
                    12345,
                    typeof(Int32));
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single node, specifying a type
explicitly.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the server reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' For a selected subset of most common types, you can also use a different overload of
the WriteValue method, and specify
' a value from the TypeCode enumeration instead.

Imports System
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
```

```
        Public Shared Sub Type()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue( _
                    endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;i=10221", _
                    12345, _
                    GetType(Int32))
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

The example below writes a single value, specifying the type to be written using a TypeCode.

Note that this example uses an overload of the WriteValue method that is not available in COM. If you are calling from a COM language or tool, use the WriteMultipleValues method instead. An example for that is given further down.

C#

```
// This example shows how to write a value into a single node, specifying a type code
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
// a different overload of the WriteValue method.

using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteValue
    {
```

```
        public static void TypeCode()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            try
            {
                client.WriteValue(
                    endpointDescriptor,
                    "nsu=http://test.org/UA/Data/;i=10221",
                    12345,
                    System.TypeCode.Int32);
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

Object Pascal

```
// This example shows how to write a value into a single node, specifying a type code
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// TypeCode is easy to use, but it does not cover all possible types. It is also
possible
// to specify the .NET Type, using a different overload of the WriteValue method.


class procedure WriteValue.TypeCode;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  EndpointDescriptor: string;
  WriteResult: _UAWriteResult;
  WriteValueArguments1: _UAWriteValueArguments;
  Results: OleVariant;
begin
  EndpointDescriptor := 'http://opcua.demo-this.com:51211/UA/SampleServer';
  // or EndpointDescriptor := 'https://opcua.demo-this.com:51212/UA/SampleServer/';
  // or EndpointDescriptor := 'opc.tcp://opcua.demo-this.com:51210/UA/SampleServer';
```

```
  // Prepare the arguments
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := EndpointDescriptor;
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 12345;
  WriteValueArguments1.ValueTypeCode := TypeCode_Int32;    // here is the type
explicitly specified

  Arguments := VarArrayCreate([0, 0], varVariant);
  Arguments[0] := WriteValueArguments1;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify value of node
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  WriteResult := IUnknown(Results[0]) as _UAWriteResult;
  if not WriteResult.Succeeded then
    WriteLn('*** Failure: ', WriteResult.Exception.GetBaseException.Message);

end;
```

PHP

```
// This example shows how to write a value into a single node, specifying a type code
explicitly.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// TypeCode is easy to use, but it does not cover all possible types. It is also
possible
// to specify the .NET Type, using a different overload of the WriteValue method.

const TypeCode_Int32 = 9;

$EndpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer";
// or $EndpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/";
// or $EndpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";

// Prepare the arguments
$WriteValueArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments1->EndpointDescriptor->UrlString = $EndpointDescriptor;
$WriteValueArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$WriteValueArguments1->Value = 12345;
$WriteValueArguments1->ValueTypeCode = TypeCode_Int32;    // here is the type
explicitly specified
```

```
$arguments[0] = $WriteValueArguments1;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify value of node
$results = $Client->WriteMultipleValues($arguments);

$WriteResult = $results[0];
if (!$WriteResult->Succeeded)
    printf("*** Failure: %s\n", $WriteResult->Exception->GetBaseException()->Message);
```

VB.NET

```
' This example shows how to write a value into a single node, specifying a type code
explicitly.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
' a different overload of the WriteValue method.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteValue
        Public Shared Sub TypeCode()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Try
                client.WriteValue( _
                    endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;i=10221", _
                    12345, _
                    System.TypeCode.Int32)
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException.Message)
                Exit Sub
            End Try
        End Sub
```

```
    End Class
End Namespace
```

VBScript

```
Rem This example shows how to write a value into a single node, specifying a type code
explicitly.
Rem
Rem Reasons for specifying the type explicitly might be:
Rem - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
Rem - The data type that the reports is incorrect.
Rem - Writing with an explicitly specified type is more efficient.
Rem
Rem TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
Rem a different overload of the WriteValue method.

Option Explicit

Const TypeCode_Int32 = 9

Dim endpointDescriptor
endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Prepare the arguments
Dim WriteValueArguments1: Set WriteValueArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments1.EndpointDescriptor.UrlString = endpointDescriptor
WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221"
WriteValueArguments1.Value = 12345
WriteValueArguments1.ValueTypeCode = TypeCode_Int32

Dim arguments(0)
Set arguments(0) = WriteValueArguments1

' Modify value of a node
Dim results: results = Client.WriteMultipleValues(arguments)
Dim WriteResult: Set WriteResult = results(0)
If Not WriteResult.Succeeded Then
    WScript.Echo "*** Failure: " & WriteResult.Exception.GetBaseException().Message
End If
```

Visual Basic (VB 6.)

```
Rem This example shows how to write a value into a single node, specifying a type code
explicitly.
Rem
Rem Reasons for specifying the type explicitly might be:
Rem - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
```

```
Rem - The data type that the reports is incorrect.
Rem - Writing with an explicitly specified type is more efficient.
Rem
Rem TypeCode is easy to use, but it does not cover all possible types. It is also
possible to specify the .NET Type, using
Rem a different overload of the WriteValue method.

Private Sub WriteValue_TypeCode_Command_Click()
    OutputText = ""

    Const TypeCode_Int32 = 9

    Dim endpointDescriptor As String
    endpointDescriptor = "http://opcua.demo-this.com:51211/UA/SampleServer"
    'or endpointDescriptor = "https://opcua.demo-this.com:51212/UA/SampleServer/"
    'or endpointDescriptor = "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    ' Prepare the arguments
    Dim WriteValueArguments1 As New UAWriteValueArguments
    WriteValueArguments1.endpointDescriptor.UrlString = endpointDescriptor
    WriteValueArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10221"
    WriteValueArguments1.SetValue 12345
    WriteValueArguments1.ValueTypeCode = TypeCode_Int32

    Dim arguments(0) As Variant
    Set arguments(0) = WriteValueArguments1

    ' Modify value of node
    Dim results As Variant
    results = Client.WriteMultipleValues(arguments)

    ' Display results
    Dim Result As UAWriteResult: Set Result = results(0)
    If Not Result.Succeeded Then
        OutputText = OutputText & "*** Failure: " &
Result.Exception.GetBaseException().Message & vbCrLf
    End If
End Sub
```

The example below writes multiple values, specifying the types to be written in the ValueType property, using a .NET Type.

Note that this approach is, at least theoretically, also available in COM languages and tools, but due to difficulties with obtaining the .NET Type, you are much more likely to specify the type using the ValueTypeCode or ValueTypeFullName property, for which the examples are given further down.

C#

```
// This example shows how to write values into 3 nodes at once, specifying a type
explicitly. It tests for success of each
// write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
```

```
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueTypeCode or
ValueTypeFullName properties.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void ValueType()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10221", 23456)
                        {ValueType = typeof(Int32)},    // here is the type explicitly
specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double")
                        {ValueType = typeof(Double)},    // here is the type explicitly
specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                        {ValueType = typeof(string)}    // here is the type explicitly
specified
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, specifying a type
explicitly. It tests for success of each
' write and displays the exception message in case of failure.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' Alternative ways of specifying the type are using the ValueTypeCode or
ValueTypeFullName properties.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub ValueType()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult =
client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10221", 23456) _
                        With {.ValueType = GetType(Int32)}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double") _
                        With {.ValueType = GetType(Double)}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                        With {.ValueType = GetType(String)} _
                } _
             )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
```

```
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

The example below writes multiple values, specifying the types to be written in the ValueTypeCode property, using a TypeCode.

This approach is suitable for a basic set of types covered by the TypeCode enumeration, and is also accessible to COM languages and tools. If you need to specify a type that is not covered by the TypeCode enumeration, use the ValueType or ValueTypeFullName property instead.

C#

```
// This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
// each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void ValueTypeCode()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10221", 23456)
                        {ValueTypeCode = TypeCode.Int32},    // here is the type
explicitly specified
```

```
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double")
                        {ValueTypeCode = TypeCode.Double},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                        {ValueTypeCode = TypeCode.String}    // here is the type
explicitly specified
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
' each write and displays the exception message in case of failure.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub ValueTypeCode()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult =
```

```
client.WriteMultipleValues(New UAWriteValueArguments() _
                {
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10221", 23456) _
                        With {.ValueTypeCode = TypeCode.Int32}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double") _
                        With {.ValueTypeCode = TypeCode.Double}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                        With {.ValueTypeCode = TypeCode.String} _
                } _
            )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

Object Pascal

```
// This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
// each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

class procedure WriteMultipleValues.ValueTypeCode;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  WriteResult: _UAWriteResult;
  WriteValueArguments1, WriteValueArguments2, WriteValueArguments3:
_UAWriteValueArguments;
  Results: OleVariant;
begin
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
```

```
  WriteValueArguments1.Value := 23456;
  WriteValueArguments1.ValueTypeCode := TypeCode_Int32;    // here is the type
explicitly specified

  WriteValueArguments2 := CoUAWriteValueArguments.Create;
  WriteValueArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10226';
  WriteValueArguments2.Value := 'This string cannot be converted to Double';
  WriteValueArguments2.ValueTypeCode := TypeCode_Double;    // here is the type
explicitly specified

  WriteValueArguments3 := CoUAWriteValueArguments.Create;
  WriteValueArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;s=UnknownNode';
  WriteValueArguments3.Value := 'ABC';
  WriteValueArguments3.ValueTypeCode := TypeCode_String;    // here is the type
explicitly specified

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := WriteValueArguments1;
  Arguments[1] := WriteValueArguments2;
  Arguments[2] := WriteValueArguments3;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteResult := IInterface(Results[I]) as _UAWriteResult;
    if WriteResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ': ', WriteResult.Exception.GetBaseException.Message);
  end;
end;
```

PHP

```
// This example shows how to write values into 3 nodes at once, specifying a type code
explicitly. It tests for success of
// each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
```

```
// Alternative ways of specifying the type are using the ValueType or ValueTypeFullName
properties.

const TypeCode_Int32 = 9;
const TypeCode_Double = 14;
const TypeCode_String = 18;

$WriteValueArguments1 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$WriteValueArguments1->Value = 23456;
$WriteValueArguments1->ValueTypeCode = TypeCode_Int32;    // here is the type
explicitly specified

$WriteValueArguments2 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226";
$WriteValueArguments2->Value = "This string cannot be converted to Double";
$WriteValueArguments2->ValueTypeCode = TypeCode_Double;    // here is the type
explicitly specified

$WriteValueArguments3 = new
COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode";
$WriteValueArguments3->Value = "ABC";
$WriteValueArguments3->ValueTypeCode = TypeCode_String;    // here is the type
explicitly specified

$arguments[0] = $WriteValueArguments1;
$arguments[1] = $WriteValueArguments2;
$arguments[2] = $WriteValueArguments3;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify values of nodes
$results = $Client->WriteMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $WriteResult = $results[$i];
    if ($WriteResult->Succeeded)
        printf("Result %d success\n", $i);
    else
        printf("Result %d: %s \n", $i, $WriteResult->Exception->GetBaseException()-
>Message);
}
```

The example below writes multiple values, specifying the types to be written in the ValueTypeFullName property, using a string form of the .NET Type.

This approach is especially suitable in COM languages and tools, if you need to specify a type that is not covered by the TypeCode enumeration.

C#

```
// This example shows how to write values into 3 nodes at once, specifying a type's
full name explicitly. It tests for
// success of each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeCode
properties.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void ValueTypeFullName()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10221", 23456)
                        {ValueTypeFullName = "System.Int32"},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double")
                        {ValueTypeFullName = "System.Double"},    // here is the type
explicitly specified
                    new UAWriteValueArguments(endpointDescriptor,
                        "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                        {ValueTypeFullName = "System.String"}    // here is the type
```

```
explicitly specified
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, specifying a type's full
name explicitly. It tests for
' success of each write and displays the exception message in case of failure.
'
' Reasons for specifying the type explicitly might be:
' - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
' - The data type that the reports is incorrect.
' - Writing with an explicitly specified type is more efficient.
'
' Alternative ways of specifying the type are using the ValueType or ValueTypeCode
properties.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub ValueTypeFullName()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult =
client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10221", 23456) _
                        With {.ValueTypeFullName = "System.Int32"}, _
                    New UAWriteValueArguments(endpointDescriptor, _
                        "nsu=http://test.org/UA/Data/;i=10226", "This string cannot be
converted to Double") _
```

```
                    With {.ValueTypeFullName = "System.Double"}, _
                New UAWriteValueArguments(endpointDescriptor, _
                    "nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                    With {.ValueTypeFullName = "System.String"} _
            } _
        )

        For i As Integer = 0 To operationResultArray.Length - 1
            If operationResultArray(i).Succeeded Then
                Console.WriteLine("Result {0}: success", i)
            Else
                Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
            End If
        Next i
    End Sub
End Class
End Namespace
```

Object Pascal

```
// This example shows how to write values into 3 nodes at once, specifying a type's
full name explicitly. It tests for
// success of each write and displays the exception message in case of failure.
//
// Reasons for specifying the type explicitly might be:
// - The data type in the server has subtypes, and the client therefore needs to pick
the subtype to be written.
// - The data type that the reports is incorrect.
// - Writing with an explicitly specified type is more efficient.
//
// Alternative ways of specifying the type are using the ValueType or ValueTypeCode
properties.

class procedure WriteMultipleValues.ValueTypeFullName;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  WriteResult: _UAWriteResult;
  WriteValueArguments1, WriteValueArguments2, WriteValueArguments3:
_UAWriteValueArguments;
  Results: OleVariant;
begin
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 23456;
  WriteValueArguments1.ValueTypeFullName := 'System.Int32';    // here is the type
explicitly specified

  WriteValueArguments2 := CoUAWriteValueArguments.Create;
  WriteValueArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText :=
```

```
'nsu=http://test.org/UA/Data/;i=10226';
  WriteValueArguments2.Value := 'This string cannot be converted to Double';
  WriteValueArguments2.ValueTypeFullName := 'System.Double';    // here is the type
explicitly specified

  WriteValueArguments3 := CoUAWriteValueArguments.Create;
  WriteValueArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;s=UnknownNode';
  WriteValueArguments3.Value := 'ABC';
  WriteValueArguments3.ValueTypeFullName := 'System.String';    // here is the type
explicitly specified

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := WriteValueArguments1;
  Arguments[1] := WriteValueArguments2;
  Arguments[2] := WriteValueArguments3;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteResult := IInterface(Results[I]) as _UAWriteResult;
    if WriteResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ': ', WriteResult.Exception.GetBaseException.Message);
  end;
end;
```

13.2.13.51 Examples - OPC Unified Architecture - Write multiple values

C#

```
// This example shows how to write values into 3 nodes at once.

using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
```

```
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226", 2.34567890),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227", "ABC")
                });
            // Production code would check the success of the operation. See separate example for
that.
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            client.WriteMultipleValues(New UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226", 2.3456789), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227", "ABC") _
                } _
             )
            ' Production code would check the success of the operation. See separate example for
that.
        End Sub
    End Class
End Namespace
```

The example above can be extended by properly testing for success of each operation, like this:

C#

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    partial class WriteMultipleValues
    {
        public static void TestSuccess()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify value of a node
            OperationResult[] operationResultArray = client.WriteMultipleValues(new[]
                {
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226",
                        "This string cannot be converted to Double"),
                    new UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC")
                });

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write values into 3 nodes at once, test for success of each write and
display the exception
' message in case of failure.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Partial Friend Class WriteMultipleValues
        Public Shared Sub TestSuccess()
```

```
            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify value of a node
            Dim operationResultArray() As OperationResult = client.WriteMultipleValues(New
UAWriteValueArguments() _
                { _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221", 23456), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226", "This string cannot be converted to Double"), _
                    New UAWriteValueArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;s=UnknownNode", "ABC") _
                } _
             )

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

C++

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include <atlsafe.h>
#include "WriteMultipleValues.h"

namespace _EasyUAClient
{
    void WriteMultipleValues::TestSuccess()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            // Instantiate the client object
            _EasyUAClientPtr ClientPtr(__uuidof(EasyUAClient));

            _UAWriteValueArgumentsPtr WriteValueArguments1Ptr(_uuidof(UAWriteValueArguments));
            WriteValueArguments1Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            WriteValueArguments1Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10221";
            WriteValueArguments1Ptr->Value = 23456;

            _UAWriteValueArgumentsPtr WriteValueArguments2Ptr(_uuidof(UAWriteValueArguments));
            WriteValueArguments2Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
```

```
this.com:51211/UA/SampleServer";
            WriteValueArguments2Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;i=10226";
            WriteValueArguments2Ptr->Value = L"This string cannot be converted to Double";

            _UAWriteValueArgumentsPtr WriteValueArguments3Ptr(__uuidof(UAWriteValueArguments));
            WriteValueArguments3Ptr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            WriteValueArguments3Ptr->NodeDescriptor->NodeId->ExpandedText =
L"nsu=http://test.org/UA/Data/;s=UnknownNode";
            WriteValueArguments3Ptr->Value = L"ABC";

            CComSafeArray<VARIANT> arguments(3);
            arguments.SetAt(0, _variant_t((IDispatch*)WriteValueArguments1Ptr));
            arguments.SetAt(1, _variant_t((IDispatch*)WriteValueArguments2Ptr));
            arguments.SetAt(2, _variant_t((IDispatch*)WriteValueArguments3Ptr));
            CComVariant vArguments(arguments);

            // Obtain values. By default, the Value attributes of the nodes will be Write.
            CComSafeArray<VARIANT> results;
            results.Attach(ClientPtr->WriteMultipleValues(&vArguments));

            // Display results
            for (int i = results.GetLowerBound(); i <= results.GetUpperBound(); i++)
            {
                _UAWriteResultPtr ResultPtr = results[i];

                if (ResultPtr->Succeeded)
                    _tprintf(_T("Result %d success\n"), i);
                else
                    _tprintf(_T("Result %d: %s\n"), i, (LPCTSTR)CW2CT(ResultPtr->Exception-
>GetBaseException()->Message));
            }
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Object Pascal

```
// This example shows how to write values into 3 nodes at once, test for
// success of each write and display the exception message in case of failure.

class procedure WriteMultipleValues.TestSuccess;
var
  Arguments: OleVariant;
  Client: OpcLabs_EasyOpcUA_TLB._EasyUAClient;
  I: Cardinal;
  WriteResult: _UAWriteResult;
  WriteValueArguments1, WriteValueArguments2, WriteValueArguments3: _UAWriteValueArguments;
  Results: OleVariant;
begin
  WriteValueArguments1 := CoUAWriteValueArguments.Create;
  WriteValueArguments1.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10221';
  WriteValueArguments1.Value := 23456;

  WriteValueArguments2 := CoUAWriteValueArguments.Create;
  WriteValueArguments2.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
```

```
  WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;i=10226';
  WriteValueArguments2.Value := 'This string cannot be converted to Double';

  WriteValueArguments3 := CoUAWriteValueArguments.Create;
  WriteValueArguments3.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText :=
'nsu=http://test.org/UA/Data/;s=UnknownNode';
  WriteValueArguments3.Value := 'ABC';

  Arguments := VarArrayCreate([0, 2], varVariant);
  Arguments[0] := WriteValueArguments1;
  Arguments[1] := WriteValueArguments2;
  Arguments[2] := WriteValueArguments3;

  // Instantiate the client object
  Client := CoEasyUAClient.Create;

  // Modify values of nodes
  TVarData(Results).VType := varArray or varVariant;
  TVarData(Results).VArray := PVarArray(Client.WriteMultipleValues(Arguments));

  // Display results
  for I := VarArrayLowBound(Results, 1) to VarArrayHighBound(Results, 1) do
  begin
    WriteResult := IInterface(Results[I]) as _UAWriteResult;
    if WriteResult.Succeeded then
      WriteLn('Result ', I, ' success')
    else
      WriteLn('Result ', I, ': ', WriteResult.Exception.GetBaseException.Message);
  end;
end;
```

PHP

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$WriteValueArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$WriteValueArguments1->Value = 23456;

$WriteValueArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226";
$WriteValueArguments2->Value = "This string cannot be converted to Double";

$WriteValueArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments");
$WriteValueArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteValueArguments3->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode";
$WriteValueArguments3->Value = "ABC";
```

```
$arguments[0] = $WriteValueArguments1;
$arguments[1] = $WriteValueArguments2;
$arguments[2] = $WriteValueArguments3;

// Modify values of nodes
$results = $Client->WriteMultipleValues($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $WriteResult = $results[$i];
    // The UA Test Server does not support this, and therefore a failure will occur.
    if ($WriteResult->Succeeded)
        printf("Result %d success\n", $i);
    else
        printf("Result %d: %s \n", $i, $WriteResult->Exception->GetBaseException()->Message);
}
```

PowerScript

```
// This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
// message in case of failure.

mle_outputtext.Text = ""

// Instantiate the client object
OLEObject client
client = CREATE OLEObject
client.ConnectToNewObject("OpcLabs.EasyOpc.UA.EasyUAClient")

// Prepare arguments.

OLEObject writeValueArguments1
writeValueArguments1 = CREATE OLEObject
writeValueArguments1.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAwriteValueArguments")

writeValueArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
writeValueArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10221"
writeValueArguments1.Value = 23456

OLEObject writeValueArguments2
writeValueArguments2 = CREATE OLEObject
writeValueArguments2.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAwriteValueArguments")

writeValueArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
writeValueArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10226"
writeValueArguments2.Value = "This string cannot be converted to Double"

OLEObject writeValueArguments3
writeValueArguments3 = CREATE OLEObject
writeValueArguments3.ConnectToNewObject("OpcLabs.EasyOpc.UA.OperationModel.UAwriteValueArguments")

writeValueArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
writeValueArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode"
writeValueArguments3.Value = "ABC"

OLEObject writeValueArgumentsList
```

```
writeValueArgumentsList = CREATE OLEObject
writeValueArgumentsList.ConnectToNewObject("OpcLabs.BaseLib.Collections.ElasticVector")
writeValueArgumentsList.Add(writeValueArguments1)
writeValueArgumentsList.Add(writeValueArguments2)
writeValueArgumentsList.Add(writeValueArguments3)

// Modify value of nodes

OLEObject operationResultList
operationResultList = client.WriteValueList(writeValueArgumentsList)

// Display results
Int i
FOR i = 0 TO operationResultList.Count - 1
    OLEObject operationResult
    operationResult = operationResultList.Item[i]
    IF operationResult.Succeeded THEN
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": success" + "~r~n"
    ELSE
        mle_outputtext.Text = mle_outputtext.Text + "Result " + String(i) + ": " +
operationResult.Exception.GetBaseException().Message + "~r~n"
    END IF
NEXT

mle_outputtext.Text = mle_outputtext.Text + "~r~n"
mle_outputtext.Text = mle_outputtext.Text + "Finished." + "~r~n"
```

Visual Basic (VB 6.)

```
Rem This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
Rem message in case of failure.

Private Sub WriteMultipleValues_TestSuccess_Command_Click()
    OutputText = ""

    ' Instantiate the client object
    Dim Client As New EasyUAClient

    Dim WriteValueArguments1 As New UAWriteValueArguments
    WriteValueArguments1.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    WriteValueArguments1.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10221"
    WriteValueArguments1.SetValue 23456

    Dim WriteValueArguments2 As New UAWriteValueArguments
    WriteValueArguments2.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    WriteValueArguments2.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;i=10226"
    WriteValueArguments2.SetValue "This string cannot be converted to Double"

    Dim WriteValueArguments3 As New UAWriteValueArguments
    WriteValueArguments3.endpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
    WriteValueArguments3.NodeDescriptor.NodeId.expandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode"
    WriteValueArguments3.SetValue "ABC"

    Dim arguments(2) As Variant
    Set arguments(0) = WriteValueArguments1
    Set arguments(1) = WriteValueArguments2
```

```
    Set arguments(2) = WriteValueArguments3

    ' Modify values of nodes
    Dim results As Variant
    results = Client.WriteMultipleValues(arguments)

    ' Display results
    Dim i: For i = LBound(results) To UBound(results)
        Dim Result As UAWriteResult: Set Result = results(i)
        If Result.Succeeded Then
            OutputText = OutputText & "Result " & i & " success" & vbCrLf
        Else
            OutputText = OutputText & "Result " & i & ": " &
Result.Exception.GetBaseException().Message & vbCrLf
        End If
    Next
End Sub
```

VBScript

```
Rem This example shows how to write values into 3 nodes at once, test for success of each write
and display the exception
Rem message in case of failure.

Option Explicit

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim WriteValueArguments1: Set WriteValueArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteValueArguments1.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10221"
WriteValueArguments1.Value = 23456

Dim WriteValueArguments2: Set WriteValueArguments2 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteValueArguments2.NodeDescriptor.NodeId.ExpandedText = "nsu=http://test.org/UA/Data/;i=10226"
WriteValueArguments2.Value = "This string cannot be converted to Double"

Dim WriteValueArguments3: Set WriteValueArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteValueArguments")
WriteValueArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteValueArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;s=UnknownNode"
WriteValueArguments3.Value = "ABC"

Dim arguments(2)
Set arguments(0) = WriteValueArguments1
Set arguments(1) = WriteValueArguments2
Set arguments(2) = WriteValueArguments3

' Modify values of nodes
Dim results: results = Client.WriteMultipleValues(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim WriteResult: Set WriteResult = results(i)
    If WriteResult.Succeeded Then
```

```
        WScript.Echo "Result " & i & " success"
    Else
        WScript.Echo "Result " & i & ": " & WriteResult.Exception.GetBaseException().Message
    End If
Next
```

13.2.13.52 Examples - OPC Unified Architecture - Write multiple values, timestamps and status code

C#

```
// This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
// write and display the exception message in case of failure.

using System;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class WriteMultiple
    {
        public static void TestSuccess()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            // Modify data of nodes' attributes
            OperationResult[] operationResultArray = client.WriteMultiple(new[]
                {
                    new UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                        new UAAttributeData(23456, UASeverity.GoodOrSuccess,
DateTime.UtcNow)),
                    new UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226",
                        new UAAttributeData(2.34567890, UASeverity.GoodOrSuccess,
DateTime.UtcNow)),
                    new UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227",
                        new UAAttributeData("ABC", UASeverity.GoodOrSuccess,
DateTime.UtcNow))
                });
```

```
            // The target server may not support this, and in such case failures will
occur.

            for (int i = 0; i < operationResultArray.Length; i++)
                if (operationResultArray[i].Succeeded)
                    Console.WriteLine("Result {0}: success", i);
                else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray[i].Exception.GetBaseException().Message);
        }
    }
}
```

VB.NET

```
' This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
' write and display the exception message in case of failure.

Imports System
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class WriteMultiple
        Public Shared Sub TestSuccess()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            ' Modify data of nodes' attributes
            Dim operationResultArray() As OperationResult = client.WriteMultiple(New
UAWriteArguments() _
              {
                  New UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                                       New UAAttributeData(23456,
UASeverity.GoodOrSuccess, Date.UtcNow)),
                  New UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10226",
                                       New UAAttributeData(2.3456789,
UASeverity.GoodOrSuccess, Date.UtcNow)),
                  New UAWriteArguments(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10227",
                                       New UAAttributeData("ABC",
UASeverity.GoodOrSuccess, Date.UtcNow))
              }
            )
```

```
            ' The target server may not support this, and in such case failures will
occur.

            For i As Integer = 0 To operationResultArray.Length - 1
                If operationResultArray(i).Succeeded Then
                    Console.WriteLine("Result {0}: success", i)
                Else
                    Console.WriteLine("Result {0}: {1}", i,
operationResultArray(i).Exception.GetBaseException().Message)
                End If
            Next i
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
// write and display the exception message in case of failure.

$GoodOrSuccess = 0;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

$StatusCode = new COM("OpcLabs.EasyOpc.UA.UAStatusCode");
$StatusCode->Severity = $GoodOrSuccess;

$WriteArguments1 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments");
$WriteArguments1->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteArguments1->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221";
$AttributeData1 = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData1->Value = 23456;
$AttributeData1->StatusCode = $StatusCode;
$AttributeData1->SourceTimestamp = (time() - 25569)/86400.0;    // works in PHP v5.6,
does not work in PHP 7.3.10
$WriteArguments1->AttributeData = $AttributeData1;

$WriteArguments2 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments");
$WriteArguments2->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteArguments2->NodeDescriptor->NodeId->ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226";
$AttributeData2 = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData2->Value = 2.3456789;
$AttributeData2->StatusCode = $StatusCode;
$AttributeData2->SourceTimestamp = (time() - 25569)/86400.0;    // works in PHP v5.6,
does not work in PHP 7.3.10
$WriteArguments2->AttributeData = $AttributeData2;

$WriteArguments3 = new COM("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments");
$WriteArguments3->EndpointDescriptor->UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer";
$WriteArguments3->NodeDescriptor->NodeId->ExpandedText =
```

```
"nsu=http://test.org/UA/Data/;i=10227";
$AttributeData3 = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData3->Value = "ABC";
$AttributeData3->StatusCode = $StatusCode;
$AttributeData3->SourceTimestamp = (time() - 25569)/86400.0;    // works in PHP v5.6,
does not work in PHP 7.3.10
$WriteArguments3->AttributeData = $AttributeData3;

$arguments[0] = $WriteArguments1;
$arguments[1] = $WriteArguments2;
$arguments[2] = $WriteArguments3;

// Modify data of nodes' attributes
$results = $Client->WriteMultiple($arguments);

// Display results
for ($i = 0; $i < count($results); $i++)
{
    $WriteResult = $results[$i];
    // The target server may not support this, and in such case failures will occur.
    if ($WriteResult->Succeeded)
        printf("Result %d success\n", $i);
    else
        printf("Result %d: %s \n", $i, $WriteResult->Exception->GetBaseException()-
>Message);
}
```

VBScript

```
Rem This example shows how to write data (a value, timestamps and status code) into 3
nodes at once, test for success of each
Rem write and display the exception message in case of failure.

Option Explicit

Const GoodOrSuccess = 0

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

Dim StatusCode: Set StatusCode = CreateObject("OpcLabs.EasyOpc.UA.UAStatusCode")
StatusCode.Severity = GoodOrSuccess

Dim WriteArguments1: Set WriteArguments1 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments")
WriteArguments1.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteArguments1.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10221"
Dim AttributeData1: Set AttributeData1 =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData1.Value = 23456
AttributeData1.StatusCode = StatusCode
AttributeData1.SourceTimestamp = Now
WriteArguments1.AttributeData = AttributeData1

Dim WriteArguments2: Set WriteArguments2 =
```

```
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments")
WriteArguments2.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteArguments2.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10226"
Dim AttributeData2: Set AttributeData2 =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData2.Value = 2.3456789
AttributeData2.StatusCode = StatusCode
AttributeData2.SourceTimestamp = Now
WriteArguments2.AttributeData = AttributeData2

Dim WriteArguments3: Set WriteArguments3 =
CreateObject("OpcLabs.EasyOpc.UA.OperationModel.UAWriteArguments")
WriteArguments3.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
WriteArguments3.NodeDescriptor.NodeId.ExpandedText =
"nsu=http://test.org/UA/Data/;i=10227"
Dim AttributeData3: Set AttributeData3 =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData3.Value = "ABC"
AttributeData3.StatusCode = StatusCode
AttributeData3.SourceTimestamp = Now
WriteArguments3.AttributeData = AttributeData3

Dim arguments(2)
Set arguments(0) = WriteArguments1
Set arguments(1) = WriteArguments2
Set arguments(2) = WriteArguments3

' Modify data of nodes' attributes
Dim results: results = Client.WriteMultiple(arguments)

' Display results
Dim i: For i = LBound(results) To UBound(results)
    Dim WriteResult: Set WriteResult = results(i)
    ' The target server may not support this, and in such case failures will occur.
    If WriteResult.Succeeded Then
        WScript.Echo "Result " & i & " success"
    Else
        WScript.Echo "Result " & i & ": " &
WriteResult.Exception.GetBaseException().Message
    End If
Next
```

13.2.13.53 Examples - OPC Unified Architecture - Write single value, timestamps and status code

C#

```
// This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.
```

```
using System;
using OpcLabs.EasyOpc.UA;
using OpcLabs.EasyOpc.UA.OperationModel;

namespace UADocExamples._EasyUAClient
{
    class Write
    {
        public static void Main1()
        {
            UAEndpointDescriptor endpointDescriptor =
                "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer";
            // or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
            // or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            // Instantiate the client object
            var client = new EasyUAClient();

            try
            {
                // Modify data of a node's attribute
                client.Write(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                    new UAAttributeData(12345, UASeverity.GoodOrSuccess,
DateTime.UtcNow));

                // The target server may not support this, and in such case a failure
will occur.
            }
            catch (UAException uaException)
            {
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.

Imports OpcLabs.EasyOpc.UA
Imports OpcLabs.EasyOpc.UA.OperationModel

Namespace UADocExamples._EasyUAClient
    Friend Class Write
        Public Shared Sub Main1()

            ' Define which server we will work with.
            Dim endpointDescriptor As UAEndpointDescriptor =
                    "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
            ' or "http://opcua.demo-this.com:51211/UA/SampleServer" (not in .NET
Standard)
```

```
            ' or "https://opcua.demo-this.com:51212/UA/SampleServer/"

            ' Instantiate the client object
            Dim client = New EasyUAClient()

            Try
                ' Modify data of a node's attribute
                client.Write(endpointDescriptor,
"nsu=http://test.org/UA/Data/;i=10221",
                                    New UAAttributeData(12345, UASeverity.GoodOrSuccess,
Date.UtcNow)) ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"

                ' The target server may not support this, and in such case a failure
will occur.
            Catch uaException As UAException
                Console.WriteLine("*** Failure: {0}",
uaException.GetBaseException().Message)
            End Try
        End Sub
    End Class
End Namespace
```

PHP

```
// This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.

$GoodOrSuccess = 0;

// Instantiate the client object
$Client = new COM("OpcLabs.EasyOpc.UA.EasyUAClient");

// Modify data of a node's attribute
$StatusCode = new COM("OpcLabs.EasyOpc.UA.UAStatusCode");
$StatusCode->Severity = $GoodOrSuccess;
$AttributeData = new COM("OpcLabs.EasyOpc.UA.UAAttributeData");
$AttributeData->Value = 12345;
$AttributeData->StatusCode = $StatusCode;
$AttributeData->SourceTimestamp = (time() - 25569)/86400.0;

// Perform the operation
try
{
    $Client->Write(
        "http://opcua.demo-this.com:51211/UA/SampleServer", // or
"opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
        "nsu=http://test.org/UA/Data/;i=10221",
        $AttributeData);
    // The target server may not support this, and in such case a failure will occur.
}
catch (com_exception $e)
{
    printf("*** Failure: %s\n", $e->getMessage());
}
```

VBScript

```
Rem This example shows how to write data (a value, timestamps and status code) into a
single attribute of a node.

Option Explicit

Const GoodOrSuccess = 0

' Instantiate the client object
Dim Client: Set Client = CreateObject("OpcLabs.EasyOpc.UA.EasyUAClient")

' Modify data of a node's attribute
Dim StatusCode: Set StatusCode = CreateObject("OpcLabs.EasyOpc.UA.UAStatusCode")
StatusCode.Severity = GoodOrSuccess
Dim AttributeData: Set AttributeData =
CreateObject("OpcLabs.EasyOpc.UA.UAAttributeData")
AttributeData.Value = 12345
AttributeData.StatusCode = StatusCode
AttributeData.SourceTimestamp = Now

' Perform the operation
On Error Resume Next
Client.Write "http://opcua.demo-this.com:51211/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=10221", _
            AttributeData ' or "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer"
' The target server may not support this, and in such case a failure will occur.
If Err.Number <> 0 Then
    WScript.Echo "*** Failure: " & Err.Source & ": " & Err.Description
    WScript.Quit
End If
On Error Goto 0
```

13.2.14 Examples - OPC XML-DA

13.2.14.1 Examples - OPC XML-DA - Browse nodes recursively

C#

```
// This example shows how to recursively browse the nodes in the OPC XML-DA address
space.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class BrowseNodes
    {
```

```
        public static void RecursiveXml()
        {
            var stopwatch = new Stopwatch();
            stopwatch.Start();

            var client = new EasyDAClient();
            _branchCount = 0;
            _leafCount = 0;

            try
            {
                BrowseFromNode(client, "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx", "");
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
                return;
            }

            stopwatch.Stop();
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds);
            Console.WriteLine("Branch count: {0}", _branchCount);
            Console.WriteLine("Leaf count: {0}", _leafCount);
        }

        private static void BrowseFromNode(
            EasyDAClient client,
            ServerDescriptor serverDescriptor,
            DANodeDescriptor parentNodeDescriptor)
        {
            Debug.Assert(client != null);
            Debug.Assert(serverDescriptor != null);
            Debug.Assert(parentNodeDescriptor != null);

            // Obtain all node elements under parentNodeDescriptor
            var browseParameters = new DABrowseParameters();    // no filtering
whatsoever
            DANodeElementCollection nodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters);
            // Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            // it, here omitted for brevity.

            foreach (DANodeElement nodeElement in nodeElementCollection)
            {
                Debug.Assert(nodeElement != null);

                Console.WriteLine(nodeElement);

                // If the node is a branch, browse recursively into it.
                if (nodeElement.IsBranch)
                {
                    _branchCount++;
```

```
                    BrowseFromNode(client, serverDescriptor, nodeElement);
                }
                else
                {
                    _leafCount++;
                }
            }
        }

        private static int _branchCount;
        private static int _leafCount;
    }
}
```

VB.NET

```
' This example shows how to recursively browse the nodes in the OPC XML-DA address
space.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.AddressSpace
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class BrowseNodes

        Shared Sub RecursiveXml()
            Dim stopwatch = New Stopwatch()
            stopwatch.Start()

            Dim client = New EasyDAClient()
            _branchCount = 0
            _leafCount = 0

            Try
                BrowseFromNode(client, "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx", "")
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try

            stopwatch.Stop()
            Console.WriteLine("Browsing has taken (milliseconds): {0}",
stopwatch.ElapsedMilliseconds)
            Console.WriteLine("Branch count: {0}", _branchCount)
            Console.WriteLine("Leaf count: {0}", _leafCount)
        End Sub

        Private Shared Sub BrowseFromNode( _
            client As EasyDAClient,
            serverDescriptor As ServerDescriptor,
            parentNodeDescriptor As DANodeDescriptor)

            Debug.Assert(client IsNot Nothing)
```

```
            Debug.Assert(serverDescriptor IsNot Nothing)
            Debug.Assert(parentNodeDescriptor IsNot Nothing)

            ' Obtain all node elements under parentNodeDescriptor
            Dim browseParameters = New DABrowseParameters() ' no filtering whatsoever
            Dim nodeElementCollection As DANodeElementCollection =
                client.BrowseNodes(serverDescriptor, parentNodeDescriptor,
browseParameters)
            ' Remark: that BrowseNodes(...) may also throw OpcException; a production
code should contain handling for
            ' it, here omitted for brevity.

            For Each nodeElement As DANodeElement In nodeElementCollection
                Debug.Assert(nodeElement IsNot Nothing)

                Console.WriteLine(nodeElement)

                ' If the node is a branch, browse recursively into it.
                If nodeElement.IsBranch Then
                    _branchCount += 1
                    BrowseFromNode(client, serverDescriptor, nodeElement)
                Else
                    _leafCount += 1
                End If
            Next nodeElement
        End Sub

        Private Shared _branchCount As Integer
        Private Shared _leafCount As Integer
    End Class
End Namespace
```

13.2.14.2 Examples - OPC XML-DA - Callback using a lambda

C#

```
// This example shows how subscribe to changes of a single item in an OPC XML-DA server
and display the value of the item
// with each change, using a callback method specified using lambda expression.

using System;
using System.Threading;
using System.Diagnostics;
using OpcLabs.EasyOpc.DataAccess;

namespace DocExamples.DataAccess.Xml
{
    class SubscribeItem
    {
        public static void CallbackLambdaXml()
        {
            // Instantiate the client object
            var client = new EasyDAClient();

            Console.WriteLine("Subscribing...");
```

```
            // The callback is a lambda expression the displays the value
            client.SubscribeItem(
                "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                "Dynamic/Analog Types/Int",
                1000,
                (sender, eventArgs) =>
                {
                    Debug.Assert(eventArgs != null);

                    if (eventArgs.Succeeded)
                    {
                        Debug.Assert(eventArgs.Vtq != null);
                        Console.WriteLine(eventArgs.Vtq.ToString());
                    }
                    else
                        Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief);
                },
                state: null);

            Console.WriteLine("Processing item changed events for 30 seconds...");
            Thread.Sleep(30 * 1000);

            Console.WriteLine("Unsubscribing...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Waiting for 2 seconds...");
            Thread.Sleep(2 * 1000);
        }
    }
}
```

VB.NET

```
' This example shows how subscribe to changes of a single item in an OPC XML-DA server
and display the value of the item
' with each change, using a callback method specified using lambda expression.

Imports OpcLabs.EasyOpc.DataAccess

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class SubscribeItem
        Shared Sub CallbackLambdaXml()
            ' Instantiate the client object
            Dim client = New EasyDAClient()

            Console.WriteLine("Subscribing...")
            ' The callback is a lambda expression the displays the value
            client.SubscribeItem(
                    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                    "Dynamic/Analog Types/Int",
                    1000,
                    Sub(sender, eventArgs)
                        Debug.Assert(eventArgs IsNot Nothing)
                        If eventArgs.Succeeded Then
                            Debug.Assert(eventArgs.Vtq IsNot Nothing)
                            Console.WriteLine(eventArgs.Vtq.ToString())
```

```
                        Else
                            Console.WriteLine("*** Failure: {0}",
eventArgs.ErrorMessageBrief)
                        End If
                    End Sub,
                    state:=Nothing)

            Console.WriteLine("Processing item changed events for 30 seconds...")
            Threading.Thread.Sleep(30 * 1000)

            Console.WriteLine("Unsubscribing...")
            client.UnsubscribeAllItems()

            Console.WriteLine("Waiting for 2 seconds...")
            Threading.Thread.Sleep(2 * 1000)
        End Sub
    End Class
End Namespace
```

13.2.14.3 Examples - OPC XML-DA - Change update rate of a single subscription

C#

```
// This example shows how change the update rate of an existing OPC XML-DA
subscription.

using System;
using System.Threading;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class ChangeItemSubscription
    {
        public static void Main1Xml()
        {
            using (var client = new EasyDAClient())
            {
                client.ItemChanged += client_ItemChanged;

                Console.WriteLine("Subscribing...");
                int handle = client.SubscribeItem(
                    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                    "Dynamic/Analog Types/Int",
                    2000,
                    state: null);

                Console.WriteLine("Waiting for 20 seconds...");
                Thread.Sleep(20 * 1000);
```

```
                Console.WriteLine("Changing subscription...");
                client.ChangeItemSubscription(handle, new DAGroupParameters(500));

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);

                Console.WriteLine("Unsubscribing...");
                client.UnsubscribeAllItems();

                Console.WriteLine("Waiting for 10 seconds...");
                Thread.Sleep(10 * 1000);
            }
        }

        // Item changed event handler
        static void client_ItemChanged(object sender, EasyDAItemChangedEventArgs e)
        {
            if (e.Succeeded)
                Console.WriteLine(e.Vtq);
            else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief);
        }
    }
}
```

VB.NET

```
' This example shows how change the update rate of an existing OPC XML-DA subscription.

Imports System.Threading
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess.Xml

    Partial Friend Class ChangeItemSubscription
        Public Shared Sub Main1Xml()
            Using client = New EasyDAClient()
                AddHandler client.ItemChanged, AddressOf client_ItemChanged

                Console.WriteLine("Subscribing...")
                Dim handle As Integer = client.SubscribeItem(
                        "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                        "Dynamic/Analog Types/Int",
                        2000,
                        state:=Nothing)

                Console.WriteLine("Waiting for 20 seconds...")
                Thread.Sleep(20 * 1000)

                Console.WriteLine("Changing subscription...")
                client.ChangeItemSubscription(handle, New DAGroupParameters(500))

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)

                Console.WriteLine("Unsubscribing...")
```

```
                client.UnsubscribeAllItems()

                Console.WriteLine("Waiting for 10 seconds...")
                Thread.Sleep(10 * 1000)
            End Using
        End Sub

        ' Item changed event handler
        Private Shared Sub client_ItemChanged(ByVal sender As Object, ByVal e As
EasyDAItemChangedEventArgs)
            If e.Succeeded Then
                Console.WriteLine(e.Vtq)
            Else
                Console.WriteLine("*** Failure: {0}", e.ErrorMessageBrief)
            End If
        End Sub
    End Class
End Namespace
```

13.2.14.4 Examples - OPC XML-DA - Event pull

C#

```
// This example shows how to subscribe to OPC XML-DA item changes and obtain the events
by pulling them.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class PullItemChanged
    {
        public static void Main1Xml()
        {
            // In order to use event pull, you must set a non-zero queue capacity
upfront.
            var client = new EasyDAClient { PullItemChangedQueueCapacity = 1000 };

            Console.WriteLine("Subscribing item changes...");
            client.SubscribeItem(
                "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                "Dynamic/Analog Types/Int",
                1000,
                state: null);

            Console.WriteLine("Processing item changes for 1 minute...");
            int endTick = Environment.TickCount + 60 * 1000;
            do
            {
                EasyDAItemChangedEventArgs eventArgs = client.PullItemChanged(2 *
1000);
                if (eventArgs != null)
                    // Handle the notification event
```

```
                Console.WriteLine(eventArgs);
            } while (Environment.TickCount < endTick);

            Console.WriteLine("Unsubscribing item changes...");
            client.UnsubscribeAllItems();

            Console.WriteLine("Finished.");
        }
    }
}
```

VB.NET

```
' This example shows how to subscribe to OPC XML-DA item changes and obtain the events
by pulling them.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class PullItemChanged
        Public Shared Sub Main1Xml()
            ' In order to use event pull, you must set a non-zero queue capacity
upfront.
            Dim client = New EasyDAClient() With {.PullItemChangedQueueCapacity = 1000}

            Console.WriteLine("Subscribing item changes...")
            client.SubscribeItem(
                    "http://opcxml.demo-this.com/XmlDaSampleServer/Service.asmx",
                    "Dynamic/Analog Types/Int",
                    1000,
                    state:=Nothing)

            Console.WriteLine("Processing item changes for 1 minute...")
            Dim endTick As Integer = Environment.TickCount + 60 * 1000
            Do
                Dim eventArgs As EasyDAItemChangedEventArgs = client.PullItemChanged(2
* 1000)
                If Not eventArgs Is Nothing Then
                    ' Handle the notification event
                    Console.WriteLine(eventArgs)
                End If
            Loop While Environment.TickCount < endTick

            Console.WriteLine("Unsubscribing item changes...")
            client.UnsubscribeAllItems()

            Console.WriteLine("Finished.")
        End Sub
    End Class
End Namespace
```

13.2.14.5 Examples - OPC XML-DA - Read multiple items

C#

```
// This example shows how to read 4 items from an OPC XML-DA server at once, and
display their values, timestamps
// and qualities.

using System;
using System.Diagnostics;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class ReadMultipleItems
    {
        public static void Main1Xml()
        {
            var client = new EasyDAClient();

            DAVtqResult[] vtqResults = client.ReadMultipleItems(
                new ServerDescriptor { UrlString = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx" },
                new DAItemDescriptor[]
                    {
                        "Dynamic/Analog Types/Double",
                        "Dynamic/Analog Types/Double[]",
                        "Dynamic/Analog Types/Int",
                        "SomeUnknownItem"
                    });

            for (int i = 0; i < vtqResults.Length; i++)
            {
                Debug.Assert(vtqResults[i] != null);

                if (vtqResults[i].Exception != null)
                {
                    Console.WriteLine("vtqResults[{0}] *** Failure: {1}", i,
vtqResults[i].ErrorMessageBrief);
                    continue;
                }
                Console.WriteLine("vtqResults[{0}].Vtq: {1}", i, vtqResults[i].Vtq);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to read 4 items from an OPC XML-DA server at once, and display
their values, timestamps
' and qualities.

Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.DataAccess.OperationModel
```

```
Namespace DocExamples.DataAccess.Xml
    Partial Friend Class ReadMultipleItems
        Public Shared Sub Main1Xml()
            Dim client = New EasyDAClient()

            Dim vtqResults() As DAVtqResult = client.ReadMultipleItems(
                    New ServerDescriptor() With {.UrlString = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx"},
                    New DAItemDescriptor() _
                    { _
                        "Dynamic/Analog Types/Double",
                        "Dynamic/Analog Types/Double[]",
                        "Dynamic/Analog Types/Int",
                        "SomeUnknownItem"
                    })

            For i = 0 To vtqResults.Length - 1
                Debug.Assert(vtqResults(i) IsNot Nothing)

                If vtqResults(i).Exception IsNot Nothing Then
                    Console.WriteLine("vtqResult[{0}] *** Failure: {1}", i,
vtqResults(i).ErrorMessageBrief)
                    Continue For
                End If
                Console.WriteLine("vtqResult[{0}].Vtq: {1}", i, vtqResults(i).Vtq)
            Next i
        End Sub
    End Class
End Namespace
```

13.2.14.6 Examples - OPC XML-DA - Write a single value

C#

```
// This example shows how to write a value into a single OPC XML-DA item.

using System;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.OperationModel;

namespace DocExamples.DataAccess.Xml
{
    class WriteItemValue
    {
        public static void Main1Xml()
        {
            var client = new EasyDAClient();

            try
            {
                client.WriteItemValue("http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx", "Static/Analog Types/Int", 12345);
```

```
            }
            catch (OpcException opcException)
            {
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to write a value into a single OPC XML-DA item.

Imports OpcLabs.EasyOpc.DataAccess
Imports OpcLabs.EasyOpc.OperationModel

Namespace DocExamples.DataAccess.Xml
    Partial Friend Class WriteItemValue
        Public Shared Sub Main1Xml()
            Dim client = New EasyDAClient()

            Try
                client.WriteItemValue("http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx", "Static/Analog Types/Int", 12345)
            Catch opcException As OpcException
                Console.WriteLine("*** Failure: {0}",
opcException.GetBaseException().Message)
                Exit Sub
            End Try
        End Sub
    End Class
End Namespace
```

13.2.14.7 Examples - OPC XML-DA - Get values of multiple properties

The example below combines the ability to browse for items with getting the data type property for each of the items obtained, for OPC XML-DA Servers.

C#

```
// This example shows how to obtain a data type of all OPC XML-DA items under a branch.

using System;
using System.Linq;
using OpcLabs.BaseLib.ComInterop;
using OpcLabs.BaseLib.OperationModel;
using OpcLabs.EasyOpc;
using OpcLabs.EasyOpc.DataAccess;
using OpcLabs.EasyOpc.DataAccess.AddressSpace;

namespace DocExamples.DataAccess.Xml
```

```
{
    class GetMultiplePropertyValues
    {
        public static void DataTypeXml()
        {
            var client = new EasyDAClient();
            ServerDescriptor serverDescriptor = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx";

            // Browse for all leaves under the "Simulation" branch
            DANodeElementCollection nodeElementCollection =
client.BrowseLeaves(serverDescriptor, "Static/Analog Types");

            // Create list of node descriptors, one for each leaf obtained
            DANodeDescriptor[] nodeDescriptorArray = nodeElementCollection
                .Where(element => !element.IsHint)  // filter out hint leafs that do
not represent real OPC items (rare)
                .Select(element => new DANodeDescriptor(element))
                .ToArray();

            // Get the value of DataType property; it is a 16-bit signed integer
            ValueResult[] valueResultArray =
client.GetMultiplePropertyValues(serverDescriptor,
                nodeDescriptorArray, DAPropertyIds.DataType);

            for (int i = 0; i < valueResultArray.Length; i++)
            {
                DANodeDescriptor nodeDescriptor = nodeDescriptorArray[i];

                // Check if there has been an error getting the property value
                ValueResult valueResult = valueResultArray[i];
                if (valueResult.Exception != null)
                {
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message);
                    continue;
                }

                // Convert the data type to VarType
                var varType = (VarType)(short)valueResult.Value;

                // Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType);
            }
        }
    }
}
```

VB.NET

```
' This example shows how to obtain a data type of all OPC XML-DA items under a branch.

Imports OpcLabs.BaseLib.ComInterop
Imports OpcLabs.BaseLib.OperationModel
Imports OpcLabs.EasyOpc
Imports OpcLabs.EasyOpc.DataAccess
```

```
Namespace DocExamples.DataAccess.Xml
    Friend Class GetMultiplePropertyValues
        Public Shared Sub DataTypeXml()
            Dim client = New EasyDAClient()
            Dim serverDescriptor As ServerDescriptor = "http://opcxml.demo-
this.com/XmlDaSampleServer/Service.asmx"

            ' Browse for all leaves under the "Simulation" branch
            Dim nodeElementCollection = client.BrowseLeaves(serverDescriptor,
"Static/Analog Types")

            ' Create list of node descriptors, one for each leaf obtained
            ' filter out hint leafs that do not represent real OPC items (rare)
            Dim nodeDescriptorArray() As DANodeDescriptor = nodeElementCollection _
                .Where(Function(element) Not element.IsHint) _
                .Select(Function(element) New DANodeDescriptor(element)) _
                .ToArray()

            ' Get the value of DataType property; it is a 16-bit signed integer
            Dim valueResultArray() As ValueResult =
client.GetMultiplePropertyValues(serverDescriptor,
                    nodeDescriptorArray, DAPropertyIds.DataType)

            For i = 0 To valueResultArray.Length - 1
                Dim nodeDescriptor = nodeDescriptorArray(i)

                ' Check if there has been an error getting the property value
                Dim valueResult As ValueResult = valueResultArray(i)
                If valueResult.Exception IsNot Nothing Then
                    Console.WriteLine("{0} *** Failure: {1}", nodeDescriptor.NodeId,
valueResult.Exception.Message)
                    Continue For
                End If

                ' Convert the data type to VarType
                Dim varType = CType(CShort(valueResult.Value), VarType)

                ' Display the obtained data type
                Console.WriteLine("{0}: {1}", nodeDescriptor.ItemId, varType)
            Next i
        End Sub
    End Class
End Namespace
```

13.2.15 Examples - Reactive Programming

13.2.15.1 Examples - Reactive Programming - Create OPC Data Access observable

The following code fragment creates the observable using one of the Create methods:

Creating OPC DA observable

```
DAItemChangedObservable<double> ramp =
     DAItemChangedObservable.Create<double>(
         "", "OPCLabs.KitServer.2", "Demo.Ramp", 1000);
```

13.2.15.2 Examples - Reactive Programming - Create OPC Data Access write observer

The following code fragment creates the observer using one of the Create methods:

Creating OPC DA write observer

```
DAWriteItemValueObserver<int> observer =
     DAWriteItemValueObserver.Create<int>(
         "", "OPCLabs.KitServer.2", "Simulation.Register_I4");
```

13.2.15.3 Examples - Reactive Programming - Transfer of OPC Data Access item values

In the example below, your application is required to continuously monitor the value of an OPC item, and when it is available (there is no failure obtaining the value), make some computations with it (we will multiply the value by 1000), and write the results into a different OPC item. This logic can be expressed by following code:

Reactive transfer of OPC DA item values

```
DAItemChangedObservable.Create<int>(
         "", "OPCLabs.KitServer.2", "Simulation.Incrementing (1 s)", 100)
     .Where(e => e.Exception == null)
     .Select(e => e.Vtq.Value * 1000)
     .Subscribe(DAWriteItemValueObserver.Create<int>(
         "", "OPCLabs.KitServer.2", "Simulation.Register_I4"));
```

Let's dissect what this example does:

1. It creates an observable sequence for significant changes in OPC-DA item "Simulation.Incrementing (1 s)".
2. The "Where" clause filters (from the observable sequence) only the changes that have a null Exception, i.e. those that carry a valid DAVtq object (value/timestamp/quality).
3. The "Select" clause takes the actual value from the Vtq property (it is of type DAVtq<int>), and returns the value multiplied by 1000.
4. An observer that writes incoming vales into the "Simulation.Register_I4" OPC-DA item is created.
5. The observer is subscribed to the transformed (processed) observable sequence.

13.2.16 Examples - User Interface

13.2.16.1 Examples - User Interface - Computer browser dialog

C++

```
// This example shows how to let the user browse for computers on the network.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _ComputerBrowserDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _ComputerBrowserDialogPtr DialogPtr(__uuidof(ComputerBrowserDialog));

            //
            DialogResult dialogResult = DialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(DialogPtr->SelectedName));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for computers on the network.

class procedure ShowDialog.Main;
var
  Dialog: ComputerBrowserDialog;
begin
  // Instantiate the dialog object
  Dialog := CoComputerBrowserDialog.Create;

  Dialog.ShowDialog(nil);
  WriteLn(Dialog.SelectedName);
end;
```

Object Pascal

```
// This example shows how to let the user browse for computers on the network.
```

```
class procedure ShowDialog.Main;
var
  ComputerBrowserDialog: OpcLabs_BaseLibForms_TLB._ComputerBrowserDialog;
begin
  // Instantiate the dialog object
  ComputerBrowserDialog := CoComputerBrowserDialog.Create;

  ComputerBrowserDialog.ShowDialog(nil);
  WriteLn(ComputerBrowserDialog.SelectedName);
end;
```

PHP

```
// This example shows how to let the user browse for computers on the network.

$Dialog = new COM("OpcLabs.BaseLib.Forms.Browsing.ComputerBrowserDialog");
printf("%d\n", $Dialog->ShowDialog);

// Display results
printf("%s\n", $Dialog->SelectedName);
```

Python

```
# This example shows how to let the user browse for computers on the network.

import win32com.client

dialog =
win32com.client.Dispatch('OpcLabs.BaseLib.Forms.Browsing.ComputerBrowserDialog')
print(dialog.ShowDialog())

# Display results
print(dialog.SelectedName)
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for computers on the network.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim Dialog As New ComputerBrowserDialog
    Dim DialogResult
    DialogResult = Dialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & Dialog.SelectedName & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for computers on the network.

Option Explicit
```

```
Const DialogResult_OK = 1

Dim Dialog: Set Dialog =
CreateObject("OpcLabs.BaseLib.Forms.Browsing.Specialized.ComputerBrowserDialog")
Dim dialogResult: dialogResult = Dialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo Dialog.SelectedName
```

13.2.16.2 Examples - User Interface - OPC Classic generic browsing dialog

Object Pascal

```
// This example shows how to let the user browse for an OPC Data Access node.

class procedure ShowDialog.Main;
var
  BrowseDialog: OpcLabs_EasyOpcForms_TLB._OpcBrowseDialog;
begin
  // Instantiate the dialog object
  BrowseDialog := CoOpcBrowseDialog.Create;

  BrowseDialog.ShowDialog(nil);

  // Display results
  WriteLn(BrowseDialog.Outputs.CurrentNodeElement.DANodeElement.ToString);
end;
```

VBScript

```
Rem This example shows how to let the user browse for an OPC Data Access node.

Option Explicit

Const DialogResult_OK = 1

Dim BrowseDialog: Set BrowseDialog =
CreateObject("OpcLabs.EasyOpc.Forms.Browsing.OpcBrowseDialog")
Dim dialogResult: dialogResult = BrowseDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If
```

```
' Display results
WScript.Echo BrowseDialog.Outputs.CurrentNodeElement.DANodeElement
```

13.2.16.3 Examples - User Interface - OPC Classic server dialog

VBScript

```
Rem This example shows how to let the user browse for an OPC "Classic" server.

Option Explicit

Const DialogResult_OK = 1

Dim ServerDialog: Set ServerDialog =
CreateObject("OpcLabs.EasyOpc.Forms.Browsing.OpcServerDialog")
'ServerDialog.Location = ""
Dim dialogResult: dialogResult = ServerDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo ServerDialog.ServerElement
```

13.2.16.4 Examples - User Interface - OPC Data Access item dialog

Object Pascal

```
// This example shows how to let the user browse for an OPC Data Access item.

class procedure ShowDialog.Main;
var
  ItemDialog: OpcLabs_EasyOpcForms_TLB._DAItemDialog;
begin
  // Instantiate the dialog object
  ItemDialog := CoDAItemDialog.Create;

  ItemDialog.ServerDescriptor.ServerClass := 'OPCLabs.KitServer.2';

  ItemDialog.ShowDialog(nil);

  // Display results
```

```
  WriteLn(ItemDialog.NodeElement.ToString);
end;
```

VBScript

```
Rem This example shows how to let the user browse for an OPC Data Access item.

Option Explicit

Const DialogResult_OK = 1

Dim ItemDialog: Set ItemDialog =
CreateObject("OpcLabs.EasyOpc.DataAccess.Forms.Browsing.DAItemDialog")
ItemDialog.ServerDescriptor.ServerClass = "OPCLabs.KitServer.2"
Dim dialogResult: dialogResult = ItemDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo "NodeElement: " & ItemDialog.NodeElement
```

13.2.16.5 Examples - User Interface - OPC Unified Architecture data dialog

C++

```
// This example shows how to let the user browse for an OPC-UA data node (a Data
Variable or a Property).

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UADataDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UADataDialogPtr DataDialogPtr(__uuidof(UADataDialog));

            //
            DataDialogPtr->EndpointDescriptor->UrlString = L"http://opcua.demo-
this.com:51211/UA/SampleServer";
            DataDialogPtr->UserPickEndpoint = true;

            //
            DialogResult dialogResult = DataDialogPtr->ShowDialog(NULL);
```

```
            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("EndpointDescriptor: %s\n"), (LPCTSTR)CW2CT(DataDialogPtr-
>EndpointDescriptor->ToString));
            _tprintf(_T("NodeElement: %s\n"), (LPCTSTR)CW2CT(DataDialogPtr-
>NodeElement->ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for an OPC-UA data node
// (a Data Variable or a Property).

class procedure ShowDialog.Main;
var
  DataDialog: UADataDialog;
begin
  // Instantiate the dialog object
  DataDialog := CoUADataDialog.Create;

  DataDialog.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  DataDialog.UserPickEndpoint := True;

  DataDialog.ShowDialog(nil);

  // Display results
  WriteLn('EndpointDescriptor: ', DataDialog.EndpointDescriptor.ToString);
  WriteLn('NodeElement: ', DataDialog.NodeElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for an OPC-UA data node
// (a Data Variable or a Property).

class procedure ShowDialog.Main;
var
  DataDialog: OpcLabs_EasyOpcForms_TLB._UADataDialog;
begin
  // Instantiate the dialog object
  DataDialog := CoUADataDialog.Create;

  DataDialog.EndpointDescriptor.UrlString := 'http://opcua.demo-
this.com:51211/UA/SampleServer';
  DataDialog.UserPickEndpoint := True;

  DataDialog.ShowDialog(nil);

  // Display results
  WriteLn('EndpointDescriptor: ', DataDialog.EndpointDescriptor.ToString);
  WriteLn('NodeElement: ', DataDialog.NodeElement.ToString);
```

```
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for an OPC-UA data node (a Data
Variable or a Property).

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim DataDialog As New UADataDialog
    DataDialog.EndpointDescriptor.UrlString = "opc.tcp://opcua.demo-
this.com:51210/UA/SampleServer"
    DataDialog.UserPickEndpoint = True

    Dim DialogResult
    DialogResult = DataDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & "EndpointDescriptor: " & DataDialog.EndpointDescriptor &
vbCrLf
    OutputText = OutputText & "NodeElement: " & DataDialog.NodeElement & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for an OPC-UA data node (a Data
Variable or a Property).

Option Explicit

Const DialogResult_OK = 1

Dim DataDialog: Set DataDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UADataDialog")
DataDialog.EndpointDescriptor.UrlString = "http://opcua.demo-
this.com:51211/UA/SampleServer"
DataDialog.UserPickEndpoint = True

Dim dialogResult: dialogResult = DataDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo "EndpointDescriptor: " & DataDialog.EndpointDescriptor
WScript.Echo "NodeElement: " & DataDialog.NodeElement
```

13.2.16.6 Examples - User Interface - OPC Unified Architecture endpoint dialog

C++

```
// This example shows how to let the user browse for an OPC-UA server endpoint.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UAEndpointDialog
{
    void ShowDialog1::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UAEndpointDialogPtr EndpointDialogPtr(__uuidof(UAEndpointDialog));

            //
            EndpointDialogPtr->DiscoveryHost = L"opcua.demo-this.com";

            //
            DialogResult dialogResult = EndpointDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(EndpointDialogPtr->DiscoveryElement-
>ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for an OPC-UA server endpoint.

class procedure ShowDialog.Main;
var
  EndpointDialog: UAEndpointDialog;
begin
  // Instantiate the dialog object
  EndpointDialog := CoUAEndpointDialog.Create;

  EndpointDialog.DiscoveryHost := 'opcua.demo-this.com';

  EndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn(EndpointDialog.DiscoveryElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for an OPC-UA server endpoint.

class procedure ShowDialog.Main;
var
  EndpointDialog: OpcLabs_EasyOpcForms_TLB._UAEndpointDialog;
begin
  // Instantiate the dialog object
  EndpointDialog := CoUAEndpointDialog.Create;

  EndpointDialog.DiscoveryHost := 'opcua.demo-this.com';

  EndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn(EndpointDialog.DiscoveryElement.ToString);
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for an OPC-UA server endpoint.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim EndpointDialog As New UAEndpointDialog
    EndpointDialog.DiscoveryHost = "opcua.demo-this.com"

    Dim DialogResult
    DialogResult = EndpointDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & EndpointDialog.DiscoveryElement & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for an OPC-UA server endpoint.

Option Explicit

Const DialogResult_OK = 1

Dim EndpointDialog: Set EndpointDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UAEndpointDialog")
EndpointDialog.DiscoveryHost = "opcua.demo-this.com"

Dim dialogResult: dialogResult = EndpointDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If
```

```
' Display results
WScript.Echo EndpointDialog.DiscoveryElement
```

13.2.16.7 Examples - User Interface - OPC Unified Architecture generic browsing dialog

C++

```
// This example shows how to let the user browse for an OPC-UA node.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UABrowseDialog
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UABrowseDialogPtr BrowseDialogPtr(__uuidof(UABrowseDialog));

            //
            BrowseDialogPtr->InputsOutputs->CurrentNodeDescriptor->EndpointDescriptor-
>Host = L"opcua.demo-this.com";
            BrowseDialogPtr->Mode->AnchorElementType = UAElementType_Host;

            //
            DialogResult dialogResult = BrowseDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("%s\n"), (LPCTSTR)CW2CT(BrowseDialogPtr->Outputs-
>CurrentNodeElement->NodeElement->ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for an OPC-UA node.

class procedure ShowDialog.Main;
var
  BrowseDialog: UABrowseDialog;
begin
  // Instantiate the dialog object
  BrowseDialog := CoUABrowseDialog.Create;
  BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host :=
```

```
'opcua.demo-this.com';
  BrowseDialog.Mode.AnchorElementType := UAElementType_Host;

  BrowseDialog.ShowDialog(nil);

  // Display results
  WriteLn(BrowseDialog.Outputs.CurrentNodeElement.NodeElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for an OPC-UA node.

class procedure ShowDialog.Main;
var
  BrowseDialog: OpcLabs_EasyOpcForms_TLB._UABrowseDialog;
begin
  // Instantiate the dialog object
  BrowseDialog := CoUABrowseDialog.Create;
  BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host :=
'opcua.demo-this.com';
  BrowseDialog.Mode.AnchorElementType := UAElementType_Host;

  BrowseDialog.ShowDialog(nil);

  // Display results
  WriteLn(BrowseDialog.Outputs.CurrentNodeElement.NodeElement.ToString);
end;
```

PHP

```
// This example shows how to let the user browse for an OPC-UA node.

$UAElementType_Host = 1;

$BrowseDialog = new COM("OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog");
$BrowseDialog->InputsOutputs->CurrentNodeDescriptor->EndpointDescriptor->Host =
"opcua.demo-this.com";
$BrowseDialog->Mode->AnchorElementType = $UAElementType_Host;

printf("%d\n", $BrowseDialog->ShowDialog);

// Display results
printf("%s\n", $BrowseDialog->Outputs->CurrentNodeElement->NodeElement);
```

Python

```
# This example shows how to let the user browse for an OPC-UA node.

import win32com.client

UAElementType_Host = 1

browseDialog =
win32com.client.Dispatch('OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog')
browseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host = "opcua.demo-
this.com"
```

```
browseDialog.Mode.AnchorElementType = UAElementType_Host

print(browseDialog.ShowDialog())

# Display results
print(browseDialog.Outputs.CurrentNodeElement.NodeElement)
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for an OPC-UA node.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim BrowseDialog As New UABrowseDialog
    BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host =
"opcua.demo-this.com"

    Dim DialogResult
    DialogResult = BrowseDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & BrowseDialog.outputs.CurrentNodeElement.NodeElement &
vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for an OPC-UA node.

Option Explicit

Const DialogResult_OK = 1

Const UAElementType_Host = 1

Dim BrowseDialog: Set BrowseDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog")
BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host = "opcua.demo-
this.com"
BrowseDialog.Mode.AnchorElementType = UAElementType_Host

Dim dialogResult: dialogResult = BrowseDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
WScript.Echo BrowseDialog.Outputs.CurrentNodeElement.NodeElement
```

13.2.16.8 Examples - User Interface - OPC Unified Architecture generic browsing dialog with multi-select

VBScript

```
Rem This example shows how to let the user browse for multiple OPC-UA nodes.

Option Explicit

Const DialogResult_OK = 1

Const UAElementType_Host = 1

Dim BrowseDialog: Set BrowseDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UABrowseDialog")
BrowseDialog.InputsOutputs.CurrentNodeDescriptor.EndpointDescriptor.Host = "opcua.demo-
this.com"
BrowseDialog.Mode.AnchorElementType = UAElementType_Host
BrowseDialog.Mode.MultiSelect = True


Dim dialogResult: dialogResult = BrowseDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
Dim SelectionElements: Set SelectionElements = BrowseDialog.Outputs.SelectionElements
Dim i: For i = 0 To SelectionElements.Count - 1
    Dim Element: Set Element = SelectionElements(i)
    WScript.Echo "SelectionElements(" & i & "): " & Element.NodeElement
Next
```

13.2.16.9 Examples - User Interface - OPC Unified Architecture host and endpoint dialog

C++

```
// This example shows how to let the user browse for a host (computer) and an endpoint
of an OPC-UA server residing on it.

#include "stdafx.h"    // Includes "QuickOpc.h", and other commonly used files
#include "ShowDialog.h"

namespace _UAHostAndEndpointDialog
```

```
{
    void ShowDialog::Main()
    {
        // Initialize the COM library
        CoInitializeEx(NULL, COINIT_MULTITHREADED);
        {
            //
            _UAHostAndEndpointDialogPtr
HostAndEndpointDialogPtr(__uuidof(UAHostAndEndpointDialog));

            //
            HostAndEndpointDialogPtr->EndpointDescriptor->Host = L"opcua.demo-
this.com";

            //
            DialogResult dialogResult = HostAndEndpointDialogPtr->ShowDialog(NULL);

            // Display results
            _tprintf(_T("%d\n"), dialogResult);
            _tprintf(_T("HostElement: %s\n"), (LPCTSTR)CW2CT(HostAndEndpointDialogPtr-
>HostElement->ToString));
            _tprintf(_T("DiscoveryElement: %s\n"),
(LPCTSTR)CW2CT(HostAndEndpointDialogPtr->DiscoveryElement->ToString));
        }
         // Release all interface pointers BEFORE calling CoUninitialize()
        CoUninitialize();
    }
}
```

Free Pascal

```
// This example shows how to let the user browse for a host (computer) and
// an endpoint of an OPC-UA server residing on it.

class procedure ShowDialog.Main;
var
  HostAndEndpointDialog: UAHostAndEndpointDialog;
begin
  // Instantiate the dialog object
  HostAndEndpointDialog := CoUAHostAndEndpointDialog.Create;

  HostAndEndpointDialog.EndpointDescriptor.Host := 'opcua.demo-this.com';

  HostAndEndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn('HostElement: ', HostAndEndpointDialog.HostElement.ToString);
  WriteLn('DiscoveryElement: ', HostAndEndpointDialog.DiscoveryElement.ToString);
end;
```

Object Pascal

```
// This example shows how to let the user browse for a host (computer) and
// an endpoint of an OPC-UA server residing on it.

class procedure ShowDialog.Main;
var
```

```
  HostAndEndpointDialog: OpcLabs_EasyOpcForms_TLB._UAHostAndEndpointDialog;
begin
  // Instantiate the dialog object
  HostAndEndpointDialog := CoUAHostAndEndpointDialog.Create;

  HostAndEndpointDialog.EndpointDescriptor.Host := 'opcua.demo-this.com';

  HostAndEndpointDialog.ShowDialog(nil);

  // Display results
  WriteLn('HostElement: ', HostAndEndpointDialog.HostElement.ToString);
  WriteLn('DiscoveryElement: ', HostAndEndpointDialog.DiscoveryElement.ToString);
end;
```

Visual Basic (VB 6.)

```
Rem This example shows how to let the user browse for a host (computer) and an endpoint
of an OPC-UA server residing on it.

Private Sub ShowDialog_Main_Command_Click()
    OutputText = ""

    Dim HostAndEndpointDialog As New UAHostAndEndpointDialog
    HostAndEndpointDialog.EndpointDescriptor.Host = "opcua.demo-this.com"

    Dim DialogResult
    DialogResult = HostAndEndpointDialog.ShowDialog

    ' Display results
    OutputText = OutputText & DialogResult & vbCrLf
    OutputText = OutputText & "HostElement: " & HostAndEndpointDialog.HostElement &
vbCrLf
    OutputText = OutputText & "DiscoveryElement: " &
HostAndEndpointDialog.DiscoveryElement & vbCrLf
End Sub
```

VBScript

```
Rem This example shows how to let the user browse for a host (computer) and an endpoint
of an OPC-UA server residing on it.

Option Explicit

Const DialogResult_OK = 1

Dim HostAndEndpointDialog: Set HostAndEndpointDialog =
CreateObject("OpcLabs.EasyOpc.UA.Forms.Browsing.UAHostAndEndpointDialog")
HostAndEndpointDialog.EndpointDescriptor.Host = "opcua.demo-this.com"

Dim dialogResult: dialogResult = HostAndEndpointDialog.ShowDialog
WScript.Echo dialogResult

If dialogResult <> DialogResult_OK Then
    WScript.Quit
End If

' Display results
```

```
WScript.Echo "HostElement: " & HostAndEndpointDialog.HostElement
WScript.Echo "DiscoveryElement: " & HostAndEndpointDialog.DiscoveryElement
```

13.2.17 Examples - Header file for C++

C++

```
// $Header: $
// Copyright (c) CODE Consulting and Development, s.r.o., Plzen. All rights reserved.
// This files imports all QuickOPC libraries (and system libraries they require),
creating Compiler COM Support wrappers.
// It also contains additional definitions for creating event sinks.
#pragma once



// #import system libraries

// mscorlib
#pragma warning(push)
#pragma warning(disable:4278)    // 'ReportEvent': identifier in type library
'BED7F4EA-1A96-11D2-8F08-00A0C9A6186D' is already a macro; use the 'rename' qualifier
#if _MSC_VER >= 1300
#define IMPORT_MSCORLIB "libid:BED7F4EA-1A96-11D2-8F08-00A0C9A6186D"
#else
#define IMPORT_MSCORLIB "mscorlib.tlb"
#endif
#import IMPORT_MSCORLIB
#pragma warning(pop)
using namespace mscorlib;

// System.Drawing
#if _MSC_VER >= 1300
#define IMPORT_SYSTEM_DRAWING "libid:D37E2A3E-8545-3A39-9F4F-31827C9124AB"
#else
#define IMPORT_SYSTEM_DRAWING "System.Drawing.tlb"
#endif
#import IMPORT_SYSTEM_DRAWING
using namespace System_Drawing;

// System.Windows.Forms
#pragma warning(push)
#pragma warning(disable:4192)    // automatically excluding 'IDataObject' while
importing type library 'System.Windows.Forms.tlb'
#if _MSC_VER >= 1300
#define IMPORT_SYSTEM_WINDOWS_FORMS "libid:215D64D2-031C-33C7-96E3-61794CD1EE61"
#else
#define IMPORT_SYSTEM_WINDOWS_FORMS "System.Windows.Forms.tlb"
#endif
#import IMPORT_SYSTEM_WINDOWS_FORMS
#pragma warning(pop)
using namespace System_Windows_Forms;
```

```
// #import QuickOPC libraries

// OpcLabs.BaseLib
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_BASELIB    "libid:ecf2e77d-3a90-4fb8-b0e2-f529f0cae9c9"
#else
#define IMPORT_OPCLABS_BASELIB    "OpcLabs.BaseLib.tlb"
#endif
#import IMPORT_OPCLABS_BASELIB \
    rename("password", "Password") \
    rename("value", "Value")
using namespace OpcLabs_BaseLib;

// OpcLabs.BaseLibForms
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_BASELIBFORMS    "libid:A0D7CA1E-7D8C-4D31-8ECB-84929E77E331"
#else
#define IMPORT_OPCLABS_BASELIBFORMS    "OpcLabs.BaseLibForms.tlb"
#endif
#import IMPORT_OPCLABS_BASELIBFORMS
using namespace OpcLabs_BaseLibForms;

// OpcLabs.EasyOpcClassic
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_EASYOPCCLASSIC    "libid:1F165598-2F77-41C8-A9F9-EAF00C943F9F"
#else
#define IMPORT_OPCLABS_EASYOPCCLASSIC    "OpcLabs.EasyOpcClassic.tlb"
#endif
#import IMPORT_OPCLABS_EASYOPCCLASSIC \
    rename("itemId", "ItemId") \
    rename("machineName", "MachineName") \
    rename("requestedUpdateRate", "RequestedUpdateRate") \
    rename("serverClass", "ServerClass")
using namespace OpcLabs_EasyOpcClassic;

// OpcLabs.EasyOpcUA
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_EASYOPCUA    "libid:E15CAAE0-617E-49C6-BB42-B521F9DF3983"
#else
#define IMPORT_OPCLABS_EASYOPCUA    "OpcLabs.EasyOpcUA.tlb"
#endif
#import IMPORT_OPCLABS_EASYOPCUA \
    rename("applicationUriString", "ApplicationUriString") \
    rename("attributeData", "AttributeData") \
    rename("browsePath", "BrowsePath") \
    rename("endpointDescriptor", "EndpointDescriptor") \
    rename("expandedText", "ExpandedText") \
    rename("inputArguments", "InputArguments") \
    rename("inputTypeCodes", "InputTypeCodes") \
    rename("nodeDescriptor", "NodeDescriptor") \
    rename("nodeId", "NodeId") \
    rename("password", "Password") \
    rename("serverCapabilities", "ServerCapabilities") \
    rename("value", "Value")
using namespace OpcLabs_EasyOpcUA;
```

```
// OpcLabs.EasyOpcForms
#if _MSC_VER >= 1300
#define IMPORT_OPCLABS_EASYOPCFORMS    "libid:2C654FA0-6CD6-496D-A64E-CE2D2925F388"
#else
#define IMPORT_OPCLABS_EASYOPCFORMS    "OpcLabs.EasyOpcForms.tlb"
#endif
#import IMPORT_OPCLABS_EASYOPCFORMS
using namespace OpcLabs_EasyOpcForms;



// DISPID-s

// EasyAEClientConfigurationEvents
#define DISPID_EASYAECLIENTCONFIGURATIONEVENTS_LOGENTRY    1001

// EasyAEClientEvents
#define DISPID_EASYAECLIENTEVENTS_EVENTINGFAILURE 1001
#define DISPID_EASYAECLIENTEVENTS_NOTIFICATION 1002

// EasyDAClientConfigurationEvents
#define DISPID_EASYDACLIENTCONFIGURATIONEVENTS_LOGENTRY    1001

// EasyDAClientEvents
#define DISPID_EASYDACLIENTEVENTS_EVENTINGFAILURE 1001
#define DISPID_EASYDACLIENTEVENTS_ITEMCHANGED 1002

// EasyUAClientConfigurationEvents
#define DISPID_EASYUACLIENTCONFIGURATIONEVENTS_LOGENTRY    1001

// EasyUAClientEvents
#define DISPID_EASYUACLIENTEVENTS_DATACHANGENOTIFICATION  1001
#define DISPID_EASYUACLIENTEVENTS_EVENTINGFAILURE  1002
#define DISPID_EASYUACLIENTEVENTS_EVENTNOTIFICATION  1003
#define DISPID_EASYUACLIENTEVENTS_SERVERCONDITIONCHANGED  1004

// EasyUASubscriberEvents
#define DISPID_EASYUASUBSCRIBEREVENTS_DATASETMESSAGE  1001
#define DISPID_EASYUASUBSCRIBEREVENTS_EVENTINGFAILURE  1002
#define DISPID_EASYUASUBSCRIBEREVENTS_RESOLVERACCESS  1003
#define DISPID_EASYUASUBSCRIBEREVENTS_SUBSCRIPTIONRESOLVED  1004
```

14 Troubleshooting

This section is under construction and will be expanded in future versions.

14.1 Troubleshooting the Setup

In case of non-obvious installation problems, it might be helpful to view a detailed installation log. The installer creates a log file automatically. By default, it is located in the user's temporary directory, has a **".txt"** extension, and its name starts with **"Setup Log"** and it includes the date of the installation and a sequence number.

For example, the installation log file location and name might be:

"C:\Users\TestUser\AppData\Local\Temp\Setup Log 2014-11-03 #001.txt"

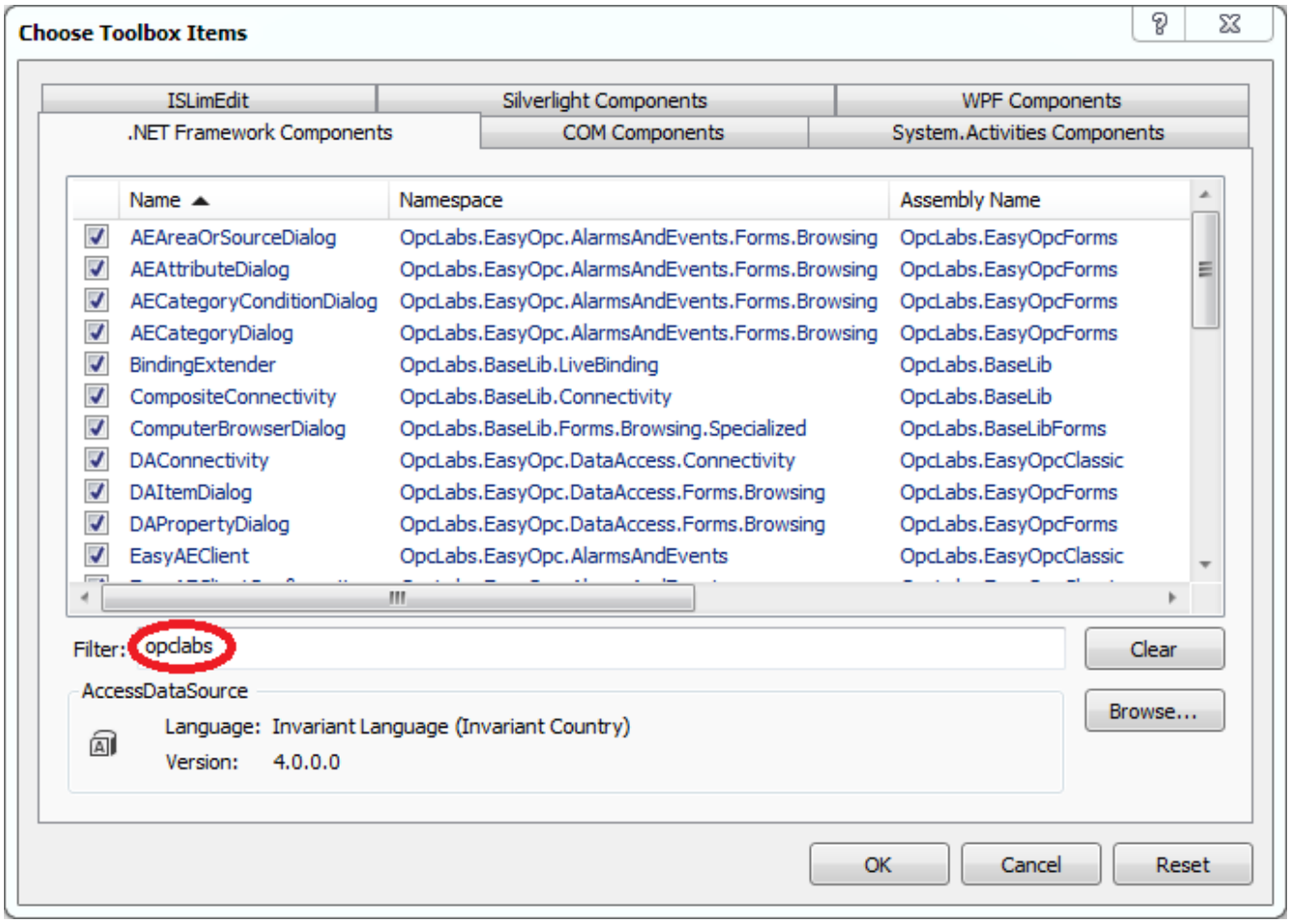
> ⚠ The "AppData" folder is hidden by default.

14.2 Troubleshooting Visual Studio Toolbox

QuickOPC toolkit items do not appear in the Toolbox

There should be tab labeled QuickOPC in the Visual Studio Toolbox, and contain the toolbox items described in this document. The actual items displayed depend on the type of project and designer. In some cases, you may not have the QuickOPC tab or any QuickOPC items in the Toolbox, or if the set of items is incomplete.

1. Check that you are designing a supported project and item - such as Windows Forms form or control, or a WPF window. Make sure you have a document open in the editor that allows placing components to it – such as a Windows form (alternatively, "Show All" can be selected in the Toolbox).
2. Verify that your project (in its Properties) is targeting a supported version of .NET Framework. If not, retarget the project to a supported version. Commonly, if you do not see the right components, it means that your project is not targeting the .NET Framework version 4.7 or later.
3. Perform a "Reset Toolbox" command (right-click in the toolbox area first), and restart Visual Studio.
4. If everything else fails, you can add the OPC Data Client components to the Toolbox manually (you only need to this once): Right-click in the **Toolbox** area, and select "Choose Items...".



In the "**Choose Toolbox Items**" dialog, select "**.NET Framework Components**" tab (for Windows Forms or similar projects), or the "**WPF Components**" tab (for WPF), and type ***opclabs*** into the **Filter** field.

Then, check all boxes next to the displayed components, and press OK. The component icons should appear in the **Toolbox**. Note: they will only be visible in the proper context, i.e. when you have selected an appropriate parent component (usually, a Windows Form) in the designer, or when you check "Show All" in the Toolbox context menu.

You can create a new toolbox tab named "QuickOPC", and move the component icons into this tab. To create a new tab, right-click in the Toolbox area, and select "Add Tab".

14.3 Using packet capture files with OPC UA PubSub

OPC Data Client can read the OPC UA PubSub network traffic from a file, instead of using live data. This is extremely handy for various kinds of troubleshooting, diagnostics, analysis and similar tasks. It allows you to capture the data generated by the publisher once, and then replay it as many times as you like.

> In order to work with packet capture files, the OpcLabs.Pcap assembly and its dependencies are needed in runtime. For more information, see **OpcLabs.Pcap Usage (Section 5.1.5.1.5)**.

The network capture needs to be in the PCAP format. Various tools, including Wireshark, can read and write this format. The current versions of Wireshark, however, store network capture in a newer format (PCAPNG) by default. Using the "Save as type" drop-down in Wireshark's "Save file as" dialog, it is, however, easy to save your network captures in the PCAP format, or even convert existing PCAPNG captures to PCAP.

For UDP message mapping, call the UseUdpCaptureFile Extension Method on the PubSub connection descriptor object, passing it the name of the capture file to be used. For Ethernet message mapping, call the UseEthernetCaptureFile Extension Method instead.

The default file name extension is ".pcap".

C#

```
// This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to
the message
// oriented middleware (receiving the messages from the network).
//
// The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made available, together with
its
// dependencies) for the capture files to work. Refer to the documentation for more information.

using System;
using System.Collections.Generic;
using System.Threading;
using OpcLabs.EasyOpc.UA.PubSub;
using OpcLabs.EasyOpc.UA.PubSub.Extensions;
using OpcLabs.EasyOpc.UA.PubSub.OperationModel;

namespace UADocExamples.PubSub._EasyUASubscriber
{
    partial class SubscribeDataSet
    {
        public static void CaptureFile()
        {
            // Define the PubSub connection we will work with. Uses implicit conversion from a string.
            UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF";
            // Use packets from the specified Ethernet capture file. The file itself is at the root of the
project, and we
            // have specified that it has to be copied to the project's output directory.
            // Note that .pcap is the default file name extension, and can thus be omitted.
            pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-Ethernet.pcap");

            // Alternative setup for Ethernet with VLAN tagging:
            //UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF:2";
            //pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-EthernetVlan.pcap");

            // Alternative setup for UDP over IPv4:
            //UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://239.0.0.1";
            //pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP.pcap");

            // Alternative setup for UDP over IPv6:
            //UAPubSubConnectionDescriptor pubSubConnectionDescriptor = "opc.udp://[ff02::1]";
            //pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP6.pcap");

            // Instantiate the subscriber object.
            var subscriber = new EasyUASubscriber();

            // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit)
publisher Id 31.
            var subscribeDataSetArguments = new UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31));

            Console.WriteLine("Subscribing...");
            subscriber.SubscribeDataSet(subscribeDataSetArguments, (sender, args) =>
            {
                // Display the dataset.
                if (args.Succeeded)
                {
                    // An event with null DataSetData just indicates a successful connection.
                    if (!(args.DataSetData is null))
                    {
                        Console.WriteLine();
                        Console.WriteLine($"Dataset data: {args.DataSetData}");
                        foreach (KeyValuePair<string, UADataSetFieldData> pair in
args.DataSetData.FieldDataDictionary)
                            Console.WriteLine(pair);
                    }
                }
                else
```

```
                {
                    Console.WriteLine();
                    Console.WriteLine($"*** Failure: {args.ErrorMessageBrief}");
                }
            });

            Console.WriteLine("Processing dataset message events for 20 seconds...");
            Thread.Sleep(20 * 1000);

            Console.WriteLine("Unsubscribing...");
            subscriber.UnsubscribeAllDataSets();

            Console.WriteLine("Waiting for 1 second...");
            // Unsubscribe operation is asynchronous, messages may still come for a short while.
            Thread.Sleep(1 * 1000);

            Console.WriteLine("Finished.");
        }

        // Example output:
        //
        //Subscribing...
        //Processing dataset message events for 20 seconds...
        //
        //Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
        //[#0, True {System.Boolean} @2019-10-31T16:04:59.145,266,700,00; Good]
        //[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
        //[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
        //[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
        //
        //Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields:
100
        //[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
        //...
    }
}
```

VB.NET

```
' This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to the
message
' oriented middleware (receiving the messages from the network).
'
' The OpcLabs.Pcap assembly needs to be referenced in your project (Or otherwise made available, together with
its
' dependencies) for the capture files to work. Refer to the documentation for more information.

Imports OpcLabs.EasyOpc.UA.PubSub
Imports OpcLabs.EasyOpc.UA.PubSub.Extensions
Imports OpcLabs.EasyOpc.UA.PubSub.OperationModel

Namespace UADocExamples.PubSub._EasyUASubscriber
    Partial Friend Class SubscribeDataSet
        Public Shared Sub CaptureFile()

            ' Define the PubSub connection we will work with. Uses implicit conversion from a string.
            Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF"
            ' Use packets from the specified Ethernet capture file. The file itself is at the root of the
project, and we
            ' have specified that it has to be copied to the project's output directory.
            ' Note that .pcap is the default file name extension, and can thus be omitted.
            pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-Ethernet.pcap")
```

```
            ' Alternative setup for Ethernet with VLAN tagging
            'Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.eth://FF-FF-FF-FF-FF-FF:2"
            'pubSubConnectionDescriptor.UseEthernetCaptureFile("UADemoPublisher-EthernetVlan.pcap")

            ' Alternative setup for UDP over IPv4
            'Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://239.0.0.1"
            'pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP.pcap")

            ' Alternative setup for UDP over IPv6
            'Dim pubSubConnectionDescriptor As UAPubSubConnectionDescriptor = "opc.udp://[ff02::1]"
            'pubSubConnectionDescriptor.UseUdpCaptureFile("UADemoPublisher-UDP6.pcap")

            ' Instantiate the subscriber object.
            Dim subscriber = New EasyUASubscriber()

            ' Define the arguments for subscribing to the dataset, where the filter Is (unsigned 64-bit)
publisher Id 31.
            Dim subscribeDataSetArguments = New UASubscribeDataSetArguments(
                pubSubConnectionDescriptor, UAPublisherId.CreateUInt64(31))

            Console.WriteLine("Subscribing...")
            subscriber.SubscribeDataSet(subscribeDataSetArguments,
                Sub(sender, eventArgs)
                    '  Display the dataset.
                    If eventArgs.Succeeded Then
                        ' An event with null DataSetData just indicates a successful connection.
                        If Not eventArgs.DataSetData Is Nothing Then
                            Console.WriteLine()
                            Console.WriteLine("Dataset data: {0}", eventArgs.DataSetData)
                            For Each pair As KeyValuePair(Of String, UADataSetFieldData) In
eventArgs.DataSetData.FieldDataDictionary
                                Console.WriteLine(pair)
                            Next
                        End If
                    Else
                        Console.WriteLine()
                        Console.WriteLine("*** Failure: {0}", eventArgs.ErrorMessageBrief)
                    End If
                End Sub)

            Console.WriteLine("Processing dataset message events for 20 seconds...")
            Threading.Thread.Sleep(20 * 1000)

            Console.WriteLine("Unsubscribing...")
            subscriber.UnsubscribeAllDataSets()

            Console.WriteLine("Waiting for 1 second...")
            ' Unsubscribe operation is asynchronous, messages may still come for a short while.
            Threading.Thread.Sleep(1 * 1000)

            Console.WriteLine("Finished...")
        End Sub
    End Class

    ' Example output
    '
    'Subscribing...
    'Processing dataset message events for 20 seconds...
    '
    'Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
    '[#0, True {System.Boolean} @2019-10-31T1604:59.145,266,700,00; Good]
    '[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
    '[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
    '[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
    '
    'Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields: 100
    '[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
```

```
    '[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
    '...

End Namespace
```

Object Pascal

```
// This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to
the message
// oriented middleware (receiving the messages from the network).
//
// In order to produce network messages for this example, run the UADemoPublisher tool. For documentation, see
// http://kb.opclabs.com/UADemoPublisher_Basics . In some cases, you may have to specify the interface name to be
used.
//
// The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made available, together with
its
// dependencies) for the capture files to work. Refer to the documentation for more information.

type
  TSubscriberEventHandlers72 = class
    procedure OnDataSetMessage(
      ASender: TObject;
      sender: OleVariant;
      const eventArgs: _EasyUADataSetMessageEventArgs);
  end;

class procedure SubscribeDataSet.CaptureFile;
var
  ConnectionDescriptor: _UAPubSubConnectionDescriptor;
  SubscribeDataSetArguments: _EasyUASubscribeDataSetArguments;
  Subscriber: TEasyUASubscriber;
  SubscriberEventHandlers: TSubscriberEventHandlers72;
begin
  // Define the PubSub connection we will work with.
  SubscribeDataSetArguments := CoEasyUASubscribeDataSetArguments.Create;
  ConnectionDescriptor := SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor;
  ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString := 'opc.eth://FF-FF-FF-FF-FF-FF';
  // Use packets from the specified Ethernet capture file. The file itself is at the root of the project, and we
  // have specified that it has to be copied to the project's output directory.
  // Note that .pcap is the default file name extension, and can thus be omitted.
  ConnectionDescriptor.UseEthernetCaptureFile('UADemoPublisher-Ethernet.pcap');

  // Alternative setup for Ethernet with VLAN tagging:
  //ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.eth://FF-FF-FF-FF-FF-FF:2';
  //ConnectionDescriptor.UseEthernetCaptureFile('UADemoPublisher-EthernetVlan.pcap');

  // Alternative setup for UDP over IPv4:
  //ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.udp://239.0.0.1';
  //ConnectionDescriptor.UseUdpCaptureFile('UADemoPublisher-UDP.pcap');

  // Alternative setup for UDP over IPv6:
  //ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = 'opc.udp://[ff02::1]';
  //ConnectionDescriptor.UseUdpCaptureFile('UADemoPublisher-UDP6.pcap');

  // Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.

SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier(UAPublisherIdType_UInt64,
31);

  // Instantiate the subscriber object and hook events.
  Subscriber := TEasyUASubscriber.Create(nil);
  SubscriberEventHandlers := TSubscriberEventHandlers72.Create;
  Subscriber.OnDataSetMessage := SubscriberEventHandlers.OnDataSetMessage;

  WriteLn('Subscribing...');
  Subscriber.SubscribeDataSet(SubscribeDataSetArguments);

  WriteLn('Processing dataset message for 20 seconds...');
  PumpSleep(20*1000);
```

```
  WriteLn('Unsubscribing...');
  Subscriber.UnsubscribeAllDataSets;

  WriteLn('Waiting for 1 second...');
  // Unsubscribe operation is asynchronous, messages may still come for a short while.
  PumpSleep(1*1000);

  WriteLn('Finished.');
  FreeAndNil(Subscriber);
  FreeAndNil(SubscriberEventHandlers);
end;

procedure TSubscriberEventHandlers72.OnDataSetMessage(
  ASender: TObject;
  sender: OleVariant;
  const eventArgs: _EasyUADataSetMessageEventArgs);
var
  Count: Cardinal;
  DictionaryEntry2: _DictionaryEntry2;
  Element: OleVariant;
  FieldDataDictionaryEnumerator: IEnumVariant;
begin
  // Display the dataset.
  if eventArgs.Succeeded then
  begin
    // An event with null DataSetData just indicates a successful connection.
    if eventArgs.DataSetData <> nil then
    begin
      WriteLn;
      WriteLn('Dataset data: ', eventArgs.DataSetData.ToString);
      FieldDataDictionaryEnumerator := eventArgs.DataSetData.FieldDataDictionary.GetEnumerator;
      while (FieldDataDictionaryEnumerator.Next(1, Element, Count) = S_OK) do
      begin
        DictionaryEntry2 := IUnknown(Element) as _DictionaryEntry2;
        WriteLn(DictionaryEntry2.ToString);
      end;
    end;
  end
  else begin
    WriteLn;
      WriteLn('*** Failure: ', eventArgs.ErrorMessageBrief);
  end;
end;

// Example output:
//
//Subscribing...
//Processing dataset message events for 20 seconds...
//
//Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
//[#0, True {System.Boolean} @2019-10-31T16:04:59.145,266,700,00; Good]
//[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
//[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
//[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
//
//Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields: 100
//[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
//...
```

VBScript

```
Rem This example shows how to feed the packet capture file into the PubSub subscriber, instead of connecting to
```

```
the message
Rem oriented middleware (receiving the messages from the network).
Rem
Rem The OpcLabs.Pcap assembly needs to be referenced in your project (or otherwise made available, together with
its
Rem dependencies) for the capture files to work. Refer to the documentation for more information.

Option Explicit

Const UAPublisherIdType_UInt64 = 4

' Define the PubSub connection we will work with.
Dim SubscribeDataSetArguments: Set SubscribeDataSetArguments =
CreateObject("OpcLabs.EasyOpc.UA.PubSub.OperationModel.EasyUASubscribeDataSetArguments")
Dim ConnectionDescriptor: Set ConnectionDescriptor =
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.ConnectionDescriptor
ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.eth://FF-FF-FF-FF-FF-FF"
' Use packets from the specified Ethernet capture file. The file itself is at the root of the project, and we
' have specified that it has to be copied to the project's output directory.
' Note that .pcap is the default file name extension, and can thus be omitted.
ConnectionDescriptor.UseEthernetCaptureFile "UADemoPublisher-Ethernet.pcap"

' Alternative setup for Ethernet with VLAN tagging:
'ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.eth://FF-FF-FF-FF-FF-FF:2"
'ConnectionDescriptor.UseEthernetCaptureFile "UADemoPublisher-EthernetVlan.pcap"

' Alternative setup for UDP over IPv4:
'ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://239.0.0.1"
'ConnectionDescriptor.UseUdpCaptureFile "UADemoPublisher-UDP.pcap"

' Alternative setup for UDP over IPv6:
'ConnectionDescriptor.ResourceAddress.ResourceDescriptor.UrlString = "opc.udp://[ff02::1]"
'ConnectionDescriptor.UseUdpCaptureFile "UADemoPublisher-UDP6.pcap"

' Define the arguments for subscribing to the dataset, where the filter is (unsigned 64-bit) publisher Id 31.
SubscribeDataSetArguments.DataSetSubscriptionDescriptor.Filter.PublisherId.SetIdentifier
UAPublisherIdType_UInt64, 31

' Instantiate the subscriber object and hook events.
Dim Subscriber: Set Subscriber = CreateObject("OpcLabs.EasyOpc.UA.PubSub.EasyUASubscriber")
WScript.ConnectObject Subscriber, "Subscriber_"

WScript.Echo "Subscribing..."
Subscriber.SubscribeDataSet SubscribeDataSetArguments

WScript.Echo "Processing dataset message events for 20 seconds..."
WScript.Sleep 20*1000

WScript.Echo "Unsubscribing..."
Subscriber.UnsubscribeAllDataSets

WScript.Echo "Waiting for 1 second..."
' Unsubscribe operation is asynchronous, messages may still come for a short while.
WScript.Sleep 1*1000

WScript.Echo "Finished."



Sub Subscriber_DataSetMessage(Sender, e)
    ' Display the dataset.
    If e.Succeeded Then
        ' An event with null DataSetData just indicates a successful connection.
        If Not (e.DataSetData Is Nothing) Then
            WScript.Echo
            WScript.Echo "Dataset data: " & e.DataSetData
            Dim Pair: For Each Pair in e.DataSetData.FieldDataDictionary
                WScript.Echo Pair
            Next
        End If
    Else
        WScript.Echo
        WScript.Echo "*** Failure: " & e.ErrorMessageBrief
```

```
    End If
End Sub


' Example output:
'
'Subscribing...
'Processing dataset message events for 20 seconds...
'
'Dataset data: 2019-10-31T16:04:59.145,266,700,00; Good; Data; publisher=(UInt64)31, writer=1, fields: 4
'[#0, True {System.Boolean} @2019-10-31T16:04:59.145,266,700,00; Good]
'[#1, 0 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
'[#2, 767 {System.Int32} @2019-10-31T16:04:59.145,266,700,00; Good]
'[#3, 10/31/2019 4:04:59 PM {System.DateTime} @2019-10-31T16:04:59.145,266,700,00; Good]
'
'Dataset data: 2019-10-31T16:04:59.170,047,500,00; Good; Data; publisher=(UInt64)31, writer=3, fields: 100
'[#0, 0 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#1, 100 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#2, 200 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#3, 300 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#4, 400 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#5, 500 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#6, 600 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#7, 700 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#8, 800 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#9, 900 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'[#10, 1000 {System.Int64} @2019-10-31T16:04:59.170,047,500,00; Good]
'...
```

When reading UDP message captures, there is a built-in IP fragment reassembler (for both IPv4 and IPv6), and thus you can work with messages whose requirements exceed the network's MTU.

The packets in the capture file are used to provide "timed" replay of the network communication. This means that when the capture is replayed, the software not only mimics the actual data in the packets, but it also attempts to maintain the same timing of the packets as it appears in the capture. An initial delay is added before the replay starts, in order to allow for startup tasks on the receiver side to finalize.

> ⚠ The features discussed here, or some of them, may not be available in all editions of the product. Check the Product Editions page for differences between the editions. The trial license has all features enabled (and is limited in period for which it provides valid data), but licenses for specific commercial editions may have functionality limitations.

15 Additional Resources

If you are migrating from earlier version, please read the "**What's New**" document (available from the Start menu or the Launcher application).

Study the **Reference (Section 16)** documentation (also available from Start menu).

Explore the **Examples (Section 13)** and Bonus Material associated with the product.

Check the vendor's Web page for updates, news, related products and other information.

16 Reference

The reference is provided online. The printable documentation (PDF booklet) only contains introductory chapters for each kind of reference material, and links to the online content.

We have put together the references to the most commonly used types in the **Concise Reference (Section 16.1)** page.

16.1 Concise Reference

This page contains links to reference material for the most commonly used types.

OpcLabs.BaseLib Assembly

- ServerDescriptor Class

OpcLabs.BaseLib Namespace

- EndpointDescriptor Class
- ObjectDescriptor Class
- ResourceDescriptor Class

OpcLabs.BaseLib.ComInterop Namespace

- VarType Class
- VarTypes Enumeration

OpcLabs.BaseLib.OperationModel Namespace

- HandleArguments Class
- OperationArguments Class
- OperationEventArgs Class
- OperationEventHandler Delegate
- OperationException Class
- OperationResult Class
- ValueArguments Class
- ValueArrayResult Class
- ValueResult Class

OpcLabs.BaseLib.Widgets Namespace

- NotifyingWidget Class
- Widget Class

OpcLabs.EasyOpcClassic Assembly

OpcLabs.EasyOpc Namespace

OpcLabs.EasyOpc.AlarmsAndEvents Namespace

- AEAttributeSetDictionary Class
- AEAttributeValueDictionary Class
- AEBrowseFilter Enumeration
- AEEventData Class
- AEEventType Class
- AEEventTypes Enumeration
- AENodeDescriptor Class
- AESubscriptionFilter Class
- AESubscriptionParameters Class
- EasyAEClient Class
- IEasyAEClient Interface
- IEasyAEClientExtension Class

OpcLabs.EasyOpc.DataAccess Namespace

- DAGroupParameters Class
- DAItemDescriptor Class
- DANodeDescriptor Class
- DAPropertyId Class
- DAQualities Enumeration
- DAQuality Class
- DAReadParameters Class
- DAVtq Class
- EasyDAClient Class
- IEasyDAClient Interface
- IEasyDAClientExtension Class

OpcLabs.EasyOpc.DataAccess.Engine Namespace

- DAClientMode Class
- DAReadWriteMethod Enumeration
- EasyDAAdaptableParameters Class
- EasyDAClientHoldPeriods Class
- EasyDAClientMode Class
- EasyDAClientParameters Class
- EasyDAClientTimeouts Class
- EasyDAClientUpdateRates Class
- EasyDAEngineParameters Class
- EasyDAInstanceParameters Class
- EasyDASharedParameters Class
- EasyDATopicParameters Class

OpcLabs.EasyOpcUA Assembly

OpcLabs.EasyOpcU.UA Namespace

- EasyUAClient Class
- IEasyUAClient Interface
- IEasyUAClientExtension Class
- UAAttributeData Class
- UAAttributeId Enumeration
- UADataValue Class
- UAEndpointDescriptor Class
- UAIndexRange Class
- UAIndexRangeList Class
- UAMonitoringParameters Class
- UANodeDescriptor Class

OpcLabs.EasyOpcU.UA.AddressSpace Namespace

- UANamespaceQualified Class
- UANodeId Class
- UANodeIdType Enumeration
- UAQualifiedName Class

OpcLabs.EasyOpcU.UA.Discovery Namespace

- IUADiscoveryElement Interface
- UAApplicationTypes Enumeration
- UADiscoveryElement Class
- UADiscoveryElementCollection Class

OpcLabs.EasyOpcU.UA.Engine Namespace

- EasyUAAdaptableParameters Class
- EasyUAEngineParameters Class
- EasyUAInstanceParameters Class
- EasyUASharedParameters Class
- UACertificateAcceptancePolicy Class
- UAClientAdaptableParameters Class
- UAClientApplicationParameters Class
- UAEngineException Class
- UAClientEngineParameters Class
- UAClientMonitoredItemParameters Class
- UAClientSessionParameters Class
- UAClientSubscriptionParameters Class
- UADiscoveryParameters Class
- UAEndpointSelectionPolicy Class

OpcLabs.EasyOpc.DataAccess.OperationModel Namespace

- DAHandleGroupArguments Class
- DAItemArguments Class
- DAItemGroupArguments Class
- DAItemValueArguments Class
- DAItemVtqArguments Class
- DAPropertyArguments Class
- DAReadItemArguments Class
- DAVtqResult Class
- EasyDAItemChangedEventArgs Class
- EasyDAItemSubscriptionArguments Class

OpcLabs.EasyOpc.Engine Namespace

- EasyClientParameters Class
- EasyEngineParameters Class
- EasyMachineParameters Class

OpcLabs.EasyOpc.OperationModel Namespace

- OpcException Class

- UAEnhancedSessionParameters Class
- UAHostParameters Class
- UAMessageSecurityModes Enumeration
- UASmartEngineParameters Class
- UAUserInteractionParameters Class

OpcLabs.EasyOpcU.UA.Navigation Namespace

- IUABrowsePathElement Interface
- UABrowsePath Class
- UABrowsePathElement Class
- UABrowsePathElementCollection Class
- UABrowsePathElementType Enumeration

OpcLabs.EasyOpcU.UA.OperationModel Namespace

- EasyUADataChangeNotificationEventArgs Class
- EasyUAMonitoredItemArguments Class
- EasyUAMonitoredItemArgumentsDictionary Class
- EasyUAMonitoredItemChangedEventArgs Class
- EasyUASubscriptionChangeArguments Class
- UAAttributeArguments Class
- UAAttributeDataResult Class
- UABrowseArguments Class
- UAException Class
- UAMonitoredItemArguments Class
- UAMonitoredItemArgumentsCollection Class
- UANodeArguments Class
- UANodeElementCollectionResult Class
- UAReadArguments Class
- UAWriteArguments Class
- UAWriteArgumentsBase Class
- UAWriteResult Class
- UAWriteValueArguments Class

16.2 .NET Assemblies Reference

The reference is both for .NET Framework and .NET Standard development platforms, although parts of the documentation may not apply, or may slightly differ depending on the platform.

In This Topic

List of .NET assemblies

- OpcLabs.BaseLib Assembly
- OpcLabs.BaseLibForms Assembly
- OpcLabs.BaseLibPresentation Assembly
- OpcLabs.EasyOpcClassic Assembly
- OpcLabs.EasyOpcForms Assembly
- OpcLabs.EasyOpcUA Assembly
- OpcLabs.OpcComplexEventProcessing Assembly

16.3 COM Type Libraries Reference

List of COM type libraries

- OpcLabs_BaseLib ActiveX DLL
- OpcLabs_BaseLibForms ActiveX DLL
- OpcLabs_EasyOpcClassic ActiveX DLL
- OpcLabs_EasyOpcForms ActiveX DLL
- OpcLabs_EasyOpcUA ActiveX DLL

The links above may not work in Microsoft Help Viewer (Visual Studio) due to a bug in the software we use to create the documentation and help. Following links to the Web-based (online) help can be used instead:

- OpcLabs.BaseLib ActiveX DLL
- OpcLabs.BaseLibForms ActiveX DLL
- OpcLabs.EasyOpcClassic ActiveX DLL
- OpcLabs.EasyOpcForms ActiveX DLL
- OpcLabs.EasyOpcUA ActiveX DLL

Usage notes

The COM objects of OPC Data Client are based upon the underlying .NET objects. In order to avoid duplicities and mismatches, the documentation is primarily maintained for the .NET objects. The .NET objects are exposed using the "interop" mechanism to the COM world using an automated translation provided by Microsoft. Therefore, a documentation that applies to a .NET type or member that is exposed to COM can be assumed to apply to the corresponding .NET type or member as well.

The bulk of the reference documentation for COM type libraries is generated from the type libraries themselves. This means that only a limited descriptive text (typically, one line) is available with each type or member. You need to look to the reference documentation for the .NET assemblies in order to find the more detailed documentation.

In addition, some languages or COM-based tools do not make direct use of the type libraries, and therefore require some additional effort to use the COM objects - such as knowing their ProgIDs, dealing with interface IDs (IIDs), etc.

Below are some rules and conventions that help with using the reference documentation for COM.

Determining the assembly name

This is not usually needed, but may be useful if you are writing a setup program for your application, and want to deploy only the .NET assemblies that are needed to run the COM objects that you are actually using.

The .NET assembly has the same name as its corresponding type library, except that the .DLL extension is used instead of .TLB.

For example, in order to use types from the **OpcLabs.EasyOpcUA.tlb** type library, the **OpcLabs.EasyOpcUA.dll** assembly is needed. Note that due to assembly dependencies, some more assemblies may be needed as well.

In This Topic

Determining the name of the corresponding .NET object

For a COM object, you need to locate the corresponding .NET object in order to find more detailed documentation related to the object.

The .NET objects that corresponds to the COM object has the same simple name, but is qualified with some namespace. Use the Search functionality of the help system to find the reference documentation for the .NET object, typing in the name of the COM object you are interested in.

For example, the EasyUAClient COM object corresponds to OpcLabs.EasyOpc.UA.EasyUAClient object in .NET.

Determining a CLSID of the object

The CLSID may be needed to create a COM object in some languages (e.g. C++).

In order to determine the CLSID of the COM object, first find the reference documentation for its corresponding .NET object. The CLSID is then listed as the GuidAttribute of this .NET object.

Determining a ProgID of the object

The CLSID may be needed to create a COM object in some languages (e.g. VBScript).

In order to determine the CLSID of the COM object, first find the reference documentation for its corresponding .NET object. The ProgID is then same as the fully qualified name of that type (i.e. including the namespace). For example, for the EasyUAClient COM object, the ProgID is "**OpcLabs.EasyOpc.UA.EasyUAClient**".

Determining the default COM interface

The default COM interface is the interface that implements the members of the COM object. You need to locate the reference documentation for the default interface to find more detailed information related to the object.

The default COM interface has the same name as the COM object, but is preceded with an underscore ('_'). In order to view the reference documentation for the default COM interface, first find the reference documentation for the .NET object that corresponds to the COM object you are interested in. On the Overview page of that .NET object, you will see the list of interfaces that the object implements. Click on the interface that starts with an undescore and has the same name as the object.

For example, the default COM interface for EasyDAClient object is named _EasyDAClient.

Determining the IID of the default COM interface

The IID may be needed to access the COM object in some languages (e.g. C++).

In order to determine the IID of the default COM interface, first find the reference documentation for this interface. The IID is then listed as the GuidAttribute of this interface.

16.4 Format Strings Reference

Formatting converts objects to strings, usually for display purposes. OPC Data Client follows the design of formatting in the .NET Framework (https://msdn.microsoft.com/en-us/library/26etazsy(v=vs.110).aspx).

Certain OPC Data Client objects implement the IFormattable interface, and can therefore participate in formatting schemes provided by the .NET Framework – e.g. their formatting can be controlled by corresponding format specifiers in the String.Format method overloads. This appendix describes the formatting of OPC Data Client types that have this kind of support.

In this part of the Reference:

- OperationResult Format Strings
- ValueResult Format Strings

- DAQuality Format Strings
- DAVtq Format Strings
- DAVtqResult Format Strings

- UAAttributeData Format Strings
- UAAttributeDataResult Format Strings

16.5 Point Types Reference

A point represents a discrete real-time value source or target. Points are used

- in Live Binding Model, and
- in Connectivity RTD Server (Excel Option).

This reference currently focuses on describing the so-called friendly syntax used in Connectivity RTD Server for various types of points.

In this part of the Reference:

- **OPC-DA Item Point (Section 16.5.1.1)**
- **OPC-DA Property Point (Section 16.5.1.2)**

- **OPC-UA Attribute Point (Section 16.5.1.3)**

16.5.1 Point Types Reference

16.5.1.1 OPC-DA Item Point

An **OPC-DA Item Point** is a connectivity point that represents an OPC Data Access (OPC-DA) item.

Point, Parameters, and Arguments Types

For operations listed below, following classes are used for the point itself, the operation parameters and operation arguments:

In This Topic

Operation	Point Type	Parameters Type	Arguments Type
Read	DAItemPoint Class	DAItemPointReadParameters Class	DAItemPointReadArguments Class
Subscribe		DAItemPointSubscribeParameters Class	DAItemPointSubscribeArguments Class
Write		DAItemPointWriteParameters Class	DAItemPointWriteArguments Class

Type Aliases

You can use following type aliases for this point type:

- **opcdaitem**
- **di**

Friendly Topic Syntax

In order to describe the point and its subscription parameters, use the following sequence:

```
ServerDescriptorString, ItemId [, RequestedUpdateRate]
```

Where:

- *ServerDescriptorString* is a descriptor string for the server.
- *ItemId* is an OPC item identifier (empty string for the root).
- *RequestedUpdateRate* is the requested update rate (how often should the updates be received). In milliseconds. Defaults to 1000 milliseconds.

If the point or parameters differ from the default in other properties, the friendly topic syntax cannot be used.

Remarks

The *ServerDescriptorString* may contain one of the following:

- A server class, i.e. ProgID (or {CLSID}) of the object.
- A string in the form "\\machineName\serverClass".
- A URI of the object.

Examples

```
"OPCLabs.KitServer.2", "Demo.Ramp"

"OPCLabs.KitServer.2", "Demo.Ramp", "5000"

"opcda://MY-PC/OPCLabs.KitServer.2", "Demo.Single"
```

XML Topic Syntax

The XML topic syntax is capable to fully describe the point and parameters. It is currently not documented, and is subject

to change.

16.5.1.2 OPC-DA Property Point

An **OPC-DA Property Point** is a connectivity point that represents an OPC Data Access (OPC-DA) property.

In This Topic

Point, Parameters, and Arguments Types

For operations listed below, following classes are used for the point itself, the operation parameters and operation arguments:

Operation	Point Type	Parameters Type	Arguments Type
Read	DAPropertyPoint Class	Object (see Note 1)	DAPropertyPointReadArguments Class
Subscribe		Object (see Note 1)	DAPropertyPointReadArguments Class (see Note 2)

The Write operation is not supported. The Subscribe operation make a on-time Read (in OPC-DA terms, Property "Get").

Note 1: No specific parameters are needed.

Note 2: Arguments for the Subscribe operation reuse the type used with the Read.

Type Aliases

You can use following type aliases for this point type:

- **opcdaproperty**
- **dp**

Friendly Topic Syntax

In order to describe the point and its subscription parameters, use the following sequence:

```
ServerDescriptorString, ItemId, { QualifiedName | PropertyId }
```

Where:

- *ServerDescriptorString* is a descriptor string for the server.
- *ItemId* is an OPC item identifier (empty string for the root).
- *QualifiedName* is a qualified name of the property (used with OPC XML-DA). Must contain a colon (:).
- *PropertyId* is a symbolic or numeric value of the property ID. Case insensitive.

If the point or parameters differ from the default in other properties, the friendly topic syntax cannot be used.

Remarks

The *ServerDescriptorString* may contain one of the following:

- A server class, i.e. ProgID (or {CLSID}) of the object.
- A string in the form "\\machineName\serverClass".
- A URI of the object.

For valid symbolic values of the *PropertyId*, see DAPropertyIds Enumeration Fields.

Examples

```
"OPCLabs.KitServer.2", "Demo.Ramp", "DataType"

"opcda://MY-PC/OPCLabs.KitServer.2", "Demo.Single", "5"
```

XML Topic Syntax

The XML topic syntax is capable to fully describe the point and parameters. It is currently not documented, and is subject to change.

16.5.1.3 OPC-UA Attribute Point

An **OPC-UA Attribute Point** is a connectivity point that represents an OPC Unified Architecture (OPC-UA) attribute on a node.

In This Topic

Point, Parameters, and Arguments Types

For operations listed below, following classes are used for the point itself, the operation parameters and operation arguments:

Operation	Point Type	Parameters Type	Arguments Type
Read	UAAttributePoint Class	UAAttributePointReadParameters Class	UAAttributePointReadArguments Class
Subscribe		UAAttributePointSubscribeParameters Class	UAAttributePointSubscribeArguments Class
Write		UAAttributePointWriteParameters Class	UAAttributePointWriteArguments Class

Type Aliases

You can use following type aliases for this point type:

- **opcuaattribute**
- **ua**

Friendly Topic Syntax

In order to describe the point and its subscription parameters, use the following sequence:

```
EndpointDescriptorString, NodeId [, [AttributeId] [, SamplingInterval]]
```

Where:

- *EndpointDescriptorString* is a descriptor string for the endpoint, i.e. a URL string of the resource.
- *NodeId* is a node Id. Contains an identifier for a node in a server's address space, together with a complete namespace URI.
- *AttributeId* identifies an attribute of a node. It can be symbolic or numeric value. Case insensitive. Defaults to the **Value** attribute.
- *SamplingInterval* is the fastest rate at which the monitored items should be accessed and evaluated (in milliseconds). Defaults to 1000 milliseconds.

If the point or parameters differ from the default in other properties, the friendly topic syntax cannot be used.

Remarks

For valid symbolic values of the *AttributeId*, see UAAttributeId Enumeration Fields.

Examples

```
   "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=11218"

   "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=11218", "DataType"

   "opc.tcp://opcua.demo-this.com:51210/UA/SampleServer",
"nsu=http://test.org/UA/Data/;i=11218", , "5000"
```

XML Topic Syntax

The XML topic syntax is capable to fully describe the point and parameters. It is currently not documented, and is subject to change.

17 Index