OPC DA/AE/HDA Client SDK .NET

Develop OPC DA / AE / HDA Clients with C# / VB.NET

Introduction

Document Control

Version	Date	Comment
8.2.0	16. Mar. 2018	Initial version based on product version 8.2

Purpose and audience of document

Microsoft's .NET Framework is an application development environment that supports multiple languages and provides a large set of standard programming APIs. This document defines an Application Programming Interface (API) for OPC UA Client and Server development based on the .NET programming model.

The OPC UA specification can be downloaded from the web site of the OPC Foundation. But only [UA Part 1] (Overview and Concepts) is available to the public. All other parts can only be downloaded from OPC Foundation members and may be used only as long as the user is an active OPC Foundation member. Because of this fact the Visual OPC .NET API hides most of the OPC UA specifications to provide the possibility to delevlop OPC UA Clients and OPC UA Servers in the .NET environment without the need to be an OPC Foundation member. The API does support OPC Unified Architecture.

This document is intended as reference material for developers of OPC UA compliant Client and Server applications. It is assumed that the reader is familiar with the Microsoft's .NET Framework and the needs of the Process Control industry.

Summary

This document gives a short overview of the functionality of the Visual OPC .NET product family. The goal of this document is to give an introduction and can be used as base for your own implementations

Referenced OPC Documents

Documents
This document partly uses extracts taken from the OPC UA specifications to be able to give at least a short introduction into the specifications. The specifications itself are available from: http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC#Specifications
OPC Unified Architecture Textbook, written by Wolfgang Mahnke, Stefan-Helmut Leitner and Matthias Damm: http://www.amazon.com/OPC-Unified-Architecture-Wolfgang-Mahnke/dp/3540688986/ref=sr_1_1?ie=UTF8&s=books&qid=1209506074&sr=8-1
[UA Part 1] OPC UA Specification: Part 1 – Concepts http://www.opcfoundation.org/UA/Part1/
[UA Part 2] OPC UA Specification: Part 2 – Security http://www.opcfoundation.org/UA/Part2/
[UA Part 3] OPC UA Specification: Part 3 – Address Space Model http://www.opcfoundation.org/UA/Part3/
[UA Part 4] OPC UA Specification: Part 4 – Services http://www.opcfoundation.org/UA/Part4/
[UA Part 5] OPC UA Specification: Part 5 – Information Model http://www.opcfoundation.org/UA/Part5/
[UA Part 6] OPC UA Specification: Part 6 – Mappings http://www.opcfoundation.org/UA/Part6/
[UA Part 7] OPC UA Specification: Part 7 – Profiles http://www.opcfoundation.org/UA/Part7/
[UA Part 8] OPC UA Specification: Part 8 – Data Access http://www.opcfoundation.org/UA/Part8/
[UA Part 9] OPC UA Specification: Part 9 – Alarm & Conditions http://www.opcfoundation.org/UA/Part9/
[UA Part 11] OPC UA Specification: Part 11 – Historical Access http://www.opcfoundation.org/UA/Part11/
[UA Part 13] OPC UA Specification: Part 13 – Aggregates http://www.opcfoundation.org/UA/Part13/

Other Referenced Documents

SOAP Part 1: SOAP Version 1.2 Part 1: Messaging Framework

http://www.w3.org/TR/soap12-part1/

SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts

http://www.w3.org/TR/soap12-part2/

XML Encryption: XML Encryption Syntax and Processing

http://www.w3.org/TR/xmlenc-core/

XML Signature:: XML-Signature Syntax and Processing

http://www.w3.org/TR/xmldsig-core/

WS Security: SOAP Message Security 1.1

http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

WS Addressing: Web Services Addressing (WS-Addressing)

http://www.w3.org/Submission/ws-addressing/

WS Trust: Web Services Trust Language (WS-Trust)

http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

WS Secure Conversation: Web Services Secure Conversation Language (WS-SecureConversation)

http://specs.xmlsoap.org/ws/2005/02/sc/WS-SecureConversation.pdf

SSL/TLS: RFC 2246: The TLS Protocol Version 1.0

http://www.ietf.org/rfc/rfc2246.txt

X200 : ITU-T X.200 – Open Systems Interconnection – Basic Reference Model

http://www.itu.int/rec/T-REC-X.200-199407-I/en

:X509: X.509 Public Key Certificate Infrastructure

http://www.itu.int/rec/T-REC-X.509-200003-I/e

HTTP: RFC 2616: Hypertext Transfer Protocol - HTTP/1.1

http://www.ietf.org/rfc/rfc2616.txt

HTTPS: RFC 2818: HTTP Over TLS

http://www.ietf.org/rfc/rfc2818.txt

IS Glossary: Internet Security Glossary

http://www.ietf.org/rfc/rfc2828.txt

NIST 800-12: Introduction to Computer Security

http://csrc.nist.gov/publications/nistpubs/800-12/

NIST 800-57: Part 3: Application-Specific Key Management Guidance

http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_PART3_key-management_Dec2009.pdf

NERC CIP: CIP 002-1 through CIP 009-1, by North-American Electric Reliability Council

http://www.nerc.com/page.php?cid=2|20

IEC 62351: Data and Communications Security

http://www.iec.ch/heb/d_mdoc-e050507.htm

SPP-ICS: System Protection Profile – Industrial Control System, by Process Control Security Requirements Forum (PCSRF)

http://www.isd.mel.nist.gov/projects/processcontrol/SPP-ICSv1.0.pdf

SHA-1: Secure Hash Algorithm RFC

http://tools.ietf.org/html/rfc3174

PKI: Public Key Infrastructure article in Wikipedia

http://en.wikipedia.org/wiki/Public_key_infrastructure

X509 PKI: Internet X.509 Public Key Infrastructure

http://www.ietf.org/rfc/rfc3280.txt

EEMUA : 2nd Edition EEMUA 191 - Alarm System - A guide to design, management and procurement (Appendixes 6, 7, 8, 9).

http://www.eemua.co.uk/

T

Table of Contents

Disclaimer

Trademark Notice

1 Overview

If your application requires access to the OPC technology you have to decide which of the OPC Specifications you want to support. The decision depends on many factors like the used OPC specifications supported by third party products or the system architcture you want to use. Technosoftware GmbH is specialized in software consulting and development services for technical and industrial applications based on OPC and the founder of Technosoftware GmbH was involved in the design and development of many software products with OPC connectivity with more than 15 years extensive knowledge about OPC.

The OPC 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.

It's **time to adapt** this specification for use in your applications with keeping in mind that you may need to support other OPC specifications as well. The Classic OPC specifications are widely used and it is important that your application can support these specifications. We recommend that you design your application to use the OPC Unified Architecture in the first place, with the option to have access to the Classic OPC Specifications in the second place. A short overview of the mainly used Classic OPC Specifications should give you an understanding of the requirements your application design have to fullfill.

1.1 OPC Technology

1.1.1 Classic OPC Specifications

The Classic OPC specifications are DCOM based specifications like OPC Data Access (DA), OPC Alarms&Events (AE) and OPC Historical Access (HDA). Each of these specifications is a separate specification and several more exists.

According to the different requirements within industrial applications, three major Classic OPC specifications have been developed. Support of one or more of these specifications should be provided by most of the OPC Client or OPC Server applications.

1.1.1.1 Data Access (DA)

An OPC DA Server allows OPC DA Clients to retrieve information about several objects: the server, the group and the items.

- The OPC server object maintains information about the server and acts as a container for OPC group objects.
- The OPC group object maintains information about itself and provides the mechanism for containing and logically organizing OPC items.
- The OPC items represent connections to data sources within the server.

The OPC DA Specification defines two read/write interfaces:

- Synchronous
  The client can perform a synchronous read from cache (simple and reasonably efficient). This may be appropriate for fairly simple clients that are reading relatively small amounts of data.
- Asynchronous
  The client can ‘subscribe’ to cached data using IAdviseSink or IOPCDataCallback which is more complex but very efficient. Asynchronous access is recommended because it minimizes the use of CPU and NETWORK resources.

In all cases the OPC DA Server gives the client access to current values of the OPC items. The OPC DA Server only holds current information in cache. Old information is overwritten. As a result of this it cannot be guaranteed that an OPC DA Client retrieves all changes in values (also not in asynchronous mode).

For such cases there exist two more OPC specifications, the OPC Alarms&Events and the OPC Historical Data Access Specification.

Technosoftware GmbH offers the following products supporting the Classic OPC Data Access Specification:

- **OPC DA/AE/HDA Client SDK .NET**
  The OPC DA/AE/HDA Client SDK .NET offers a fast and easy access to the OPC client technology. Develop client applications supporting the Classic OPC specifications DA, AE and HDA with C#/VB.NET for the .NET framework. For new developments we recommend the OPC Client SDK .NET with the DA/AE/HAD Add-On.
- **OPC UA Server Gateway .NET**
  The OPC UA Server Gateway .NET allows any OPC DA, AE or HDA Client to access OPC UA servers.

1.1.1.2 Alarms&Events (AE)

The OPC AE interface provides a mechanism for OPC AE clients to be notified when a specified event and/or alarm condition occurs. The browser interface also allows OPC AE clients to determine the list of events and conditions supported by an OPC AE Server as well as to get their current status.

Within OPC, an alarm is an abnormal condition and is thus a special case of a condition. A condition is a named state of the OPC Event Server or of one of its contained objects that is of interest to an OPC AE client. For example, the tag Temperature may have the following conditions associated with it: HighAlarm, HighHighAlarm, Normal, LowAlarm, and LowLowAlarm.

On the other hand, an event is a detectable occurrence that is of significance to the OPC Server, the device it represents, and its OPC AE clients. An event may or may not be associated with a condition. For example, the transition into HighAlarm and Normal conditions are events, which are associated with conditions. However, operator actions, system configuration changes, and system errors are examples of events, which are not related to specific conditions. OPC AE clients may subscribe to be notified of the occurrence of specified events.

The OPC AE specification provides methods enabling the OPC AE client to:

- Determine the types of events that are supported by the OPC AE server.
- Enter subscriptions to specified events so that OPC AE clients can receive notifications of their occurrences. Filters may be used to define a subset of desired events.
- Access and manipulate conditions implemented by the OPC AE server.

Technosoftware GmbH offers the following products supporting the Classic OPC Alarms&Events Specification:

- **OPC DA/AE/HDA Client SDK .NET**
  The OPC DA/AE/HDA Client SDK .NET offers a fast and easy access to the OPC client technology. Develop client applications supporting the Classic OPC specifications DA, AE and HDA with C#/VB.NET for the .NET framework. For new developments we recommend the OPC Client SDK .NET with the DA/AE/HAD Add-On.
- **OPC UA Server Gateway .NET**
  The OPC UA Server Gateway .NET allows any OPC DA, AE or HDA Client to access OPC UA servers.

1.1.1.3 Historical Data Access (HDA)

Historical engines today produce an added source of information that should be distributed to users and software clients that are interested in this information. Currently most historical systems use their own proprietary interfaces for dissemination of data. There is no capability to augment or use existing historical solutions with other capabilities in a plug-n-play environment. This requires the developer to recreate the same infrastructure for their products, as all other vendors have had to develop independently with no interoperability with any other systems.

In keeping with the desire to integrate data at all levels of business, historical information can be considered to be another type of data.

There are several types of Historian servers. Some key types supported by the HDA specification are:

- Simple Trend data servers.

These servers provided little else then simple raw data storage. (Data would typically be the types of data available from an OPC Data Access server, usually provided in the form of a duple [Time Value & Quality])

- Complex data compression and analysis servers. These servers provide data compression as well as raw data storage. They are capable of providing summary data or data analysis functions, such as average values, minimums and maximums etc. They can support data updates and history of the updates. They can support storage of annotations along with the actual historical data storage.

Technosoftware GmbH offers the following products supporting the Classic OPC Historical Data Access Specification:

- **OPC DA/AE/HDA Client SDK .NET**
  The OPC DA/AE/HDA Client SDK .NET offers a fast and easy access to the OPC client technology. Develop client applications supporting the Classic OPC specifications DA, AE and HDA with C#/VB.NET for the .NET framework. For new developments we recommend the OPC Client SDK .NET with the DA/AE/HAD Add-On.
- **OPC UA Server Gateway .NET**
  The OPC UA Server Gateway .NET allows any OPC DA, AE or HDA Client to access OPC UA servers.

1.1.2 OPC XML-DA Specification

The OPC XML-DA specification is a web services based specification and is a step between Classic OPC and OPC Unified Architecture. The functionality is restricted to OPC DA.

Technosoftware GmbH does not offer any products supporting the OPC XML-DA Specification.

1.1.3 OPC .NET 4.0 (WCF)

OPC .NET 4.0 (WCF) is not a specification like the others mentioned here, it bridges the gap between Microsoft.NET and the world of Classic OPC. OPC .NET 4.0 is an OPC standard C# Application Programming Interface (API) designed to simplify client access to OPC Classic servers. It also includes a formal WCF Interface; however, there is no compliance or certification program for this interface. The Classic OPC interfaces are still the primary means to ensure multi-vendor interoperability.

Technosoftware GmbH does not offer any products supporting the OPC .NET 4.0 API. We suggest to support the Classic OPC specifications through a wrapper to the OPC Unified Architecture, like included with the OPC UA Client SDK .NET – DA/AE/HDA Add-On. This gives you support for the Classic OPC specifications as well as the new OPC Unified Architecture.

1.1.4 OPC Unified Architecture Specification

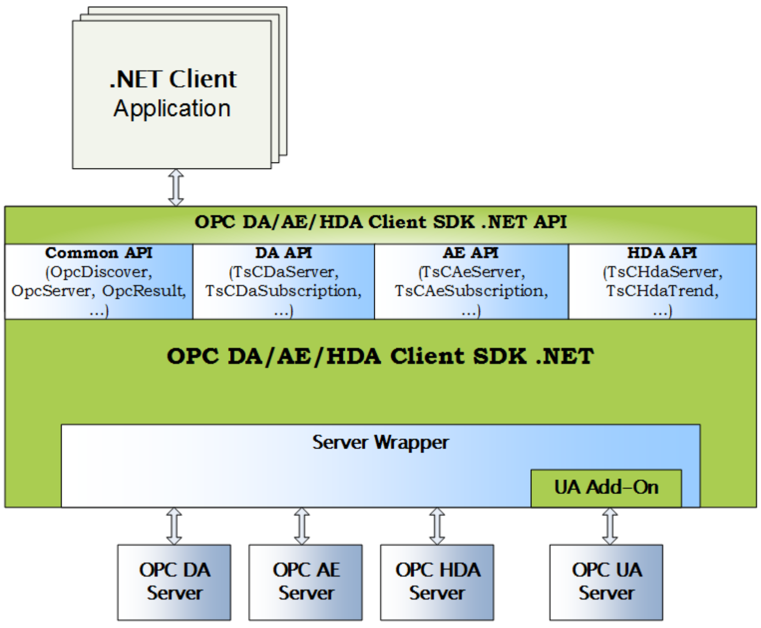
OPC Unified Architecture (UA) is a platform-independent standard through which various kinds of systems and devices can communicate by sending Messages between Clients and Servers over various types of networks. It supports robust, secure communication that assures the identity of Clients and Servers and resists attacks. OPC UA defines standard sets of Services that Servers may provide, and individual Servers specify to Clients what Service sets they support. Information is conveyed using standard and vendor- defined data types, and Servers define object models that Clients can dynamically discover. Servers can provide access to both current and historical data, as well as Alarms and Events to notify Clients of important changes. OPC UA can be mapped onto a variety of communication protocols and data can be encoded in various ways to trade off portability and efficiency.

The OPC Foundation provides deliverables for its member companies. These include a .NET based OPC UA Stack, an ANSI C based OPC UA Stack and a Java based OPC UA Stack. The .NET based OPC UA Stack is the base of all Technosoftware GmbH's .NET based products.

Technosoftware GmbH offers the following products supporting the OPC Unified Architecture Specification:

- **OPC UA Client SDK .NET**
  The OPC UA Client SDK .NET offers a fast and easy access to the OPC client technology. Develop client applications supporting OPC Unified Architecture with C#/VB.NET for the .NET framework.
- **OPC UA Client Gateway .NET**
  The OPC UA Client Gateway .NET allows any OPC UA Clients to access Classic OPC DA, AE and HDA servers.
- **OPC UA Server SDK .NET**
  The OPC UA Server SDK .NET offers a fast and easy access to the OPC Unified Architecture (UA) technology. Develop OPC UA 1.01 / 1.02 / 1.03 compliant Servers with C#/VB.NET for the .NET framework.
- **OPC UA Server Gateway .NET**
  The OPC UA Server Gateway .NET allows any OPC DA, AE or HDA Client to access OPC UA servers.

2 OPC DA/AE/HDA Client SDK .NET



.NET Client Application
OPC DA/AE/HDA Client SDK .NET API
Common API (OpcDiscover, OpcServer, OpcResult, ...)
DA API (TsCDaServer, TsCDaSubscription, ...)
AE API (TsCAeServer, TsCAeSubscription, ...)
HDA API (TsCHdaServer, TsCHdaTrend, ...)
OPC DA/AE/HDA Client SDK .NET
Server Wrapper
UA Add-On
OPC DA Server
OPC AE Server
OPC HDA Server
OPC UA Server

The OPC DA/AE/HDA Client SDK .NET offers a fast and easy access to the OPC client technology. Develop client applications supporting the Classic OPC specifications DA, AE and HDA with C#/VB.NET for the .NET framework.

OPC Clients can be developed fast and easily without the need to spend a lot of time learning how to implement the Classic OPC specifications DA, AE or HDA.

The OPC DA/AE/HDA Client SDK .NET API defines classes which can be used to implement an OPC client capable to access OPC DA, AE and HDA servers supporting different profiles with the same API. These classes manage client side state information; provide higher level abstractions for OPC tasks such as managing sessions and subscriptions or saving and restoring connection information for later use.

A large set of sample controls can be used as reference on how to implement different use cases. The sample controls as well as a client using them is included in source code.

The UA Add-On extends the OPC DA/AE/HDA Client SDK .NET with an OPC UA server wrapper. With this add-on OPC UA servers can also be accessed through the same API. Because of the Classic OPC oriented API the UA server access is limited to the Data Access, Alarms&Conditions and the Historical Access features of the UA server.

Features:

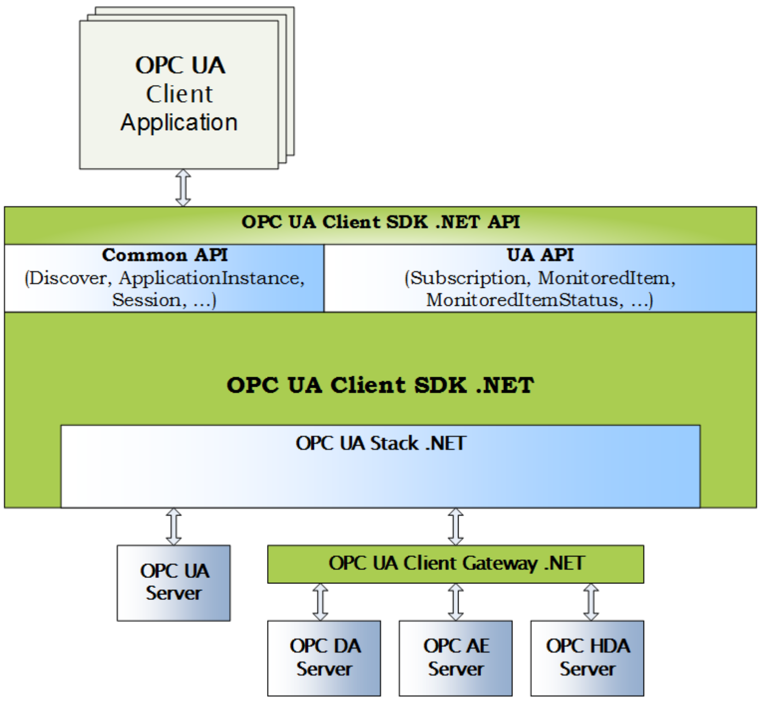
- Clients are fully compliant to OPC DA 2.05, OPC DA 3.0, OPC AE 1.10 and OPC HDA 1.20.
- Supports 32bit and 64bit mode with the same assembly (build with Any CPU).
- Simple error handling.
- The OPC DA/AE/HDA Client SDK .NET API allows access to OPC servers through the same set of classes and methods.
- Easy to understand and well documented sample clients show how to use the API.
- Source of several sample controls included showing different use cases.
- Increased maintainability and usability using consistent coding guidelines and standard naming conventions.
- Includes Documentation and Samples.
- No additional royalties or run-times fees required.

3 OPC UA SDKs .NET Product Line

The OPC UA SDK .NET product line consists of toolkits for OPC Client and OPC Server Development as well as tools to give Unified Architecture applications access to the Classic OPC specifications.

3.1 OPC UA Client SDK .NET

If If your application requires access to the OPC technology you have to decide which of the OPC Specifications you want to support. The decision depends on many factors like the used OPC specifications supported by third party products or the system architecture you want to use. Technosoftware GmbH is specialized in software consulting and development services for technical and industrial applications based on OPC and the founder of Technosoftware GmbH was involved in the design and development of many software products with OPC connectivity with more then 20 years extensive knowledge about OPC.


OPC UA Client Application
OPC UA Client SDK .NET API
Common API
(Discover, ApplicationInstance, Session, ...)
UA API
(Subscription, MonitoredItem, MonitoredItemStatus, ...)
OPC UA Client SDK .NET
OPC UA Stack .NET
OPC UA Server
OPC UA Client Gateway .NET
OPC DA Server
OPC AE Server
OPC HDA Server

The OPC 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.

It’s Time to Adapt this specification for use in your applications and we recommend considering designing your application to use the OPC Unified Architecture.

We recommend considering designing your application to use the OPC Unified Architecture in the first place by using our new **OPC UA Client SDK .NET** toolkit which allows development of client applications supporting OPC UA and through the **OPC UA Client Gateway .NET** also the Classic OPC specifications DA, AE and HDA.

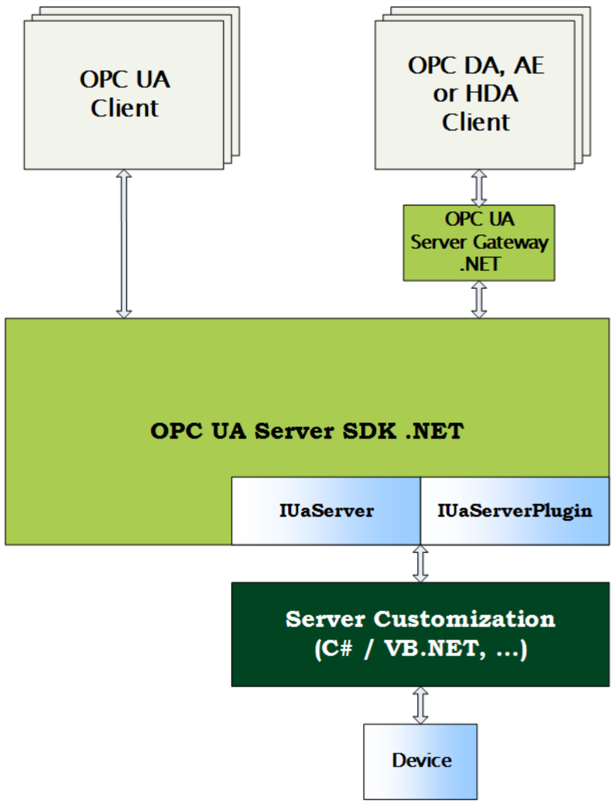
The **OPC UA Client SDK .NET** offers a fast and easy access to the OPC UA Client technology. Develop OPC compliant UA Clients with C#/VB.NET for the .NET 4.6.1 frameworks.

The **OPC UA Client SDK .NET** API defines classes which can be used to implement an OPC client capable to access OPC servers supporting different profiles with the same API. These classes manage client side state information; provide higher level abstractions for OPC tasks such as managing sessions and subscriptions or saving and restoring connection information for later use.

A large set of sample controls can be used as reference on how to implement different use cases. The sample controls as well as a client using them is included in source code.

The **OPC UA Client Gateway.NET** extends the OPC UA Client SDK .NET with an OPC DA/AE/HDA server wrapper. Because of the functionality of the use Classic OPC Server access is limited to the DA, AE and the HDA features of the Classic OPC server.

3.2 OPC UA Server SDK .NET



If your application requires access to the OPC technology you have to decide which of the OPC Specifications you want to support. The decision depends on many factors like the used OPC specifications supported by third party products or the system architecture you want to use. Technosoftware GmbH is specialized in software consulting and development services for technical and industrial applications based on OPC and the founder of Technosoftware GmbH was involved in the design and development of many software products with OPC connectivity with more then 20 years extensive knowledge about OPC.

The OPC 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.

It's Time to Adapt this specification for use in your applications with keeping in mind that you may need to support other OPC specifications as well. The Classic OPC specifications are widely used and it is important that your application can support these specifications.

In some cases it may still usefull to have direct access to the Classic OPC specifications. For this you can use the **OPC UA Server Gateway .NET** which gives OPC DA, OPC AE and OPC HDA client applications direct access to the OPC UA server.

The OPC UA Server SDK .NET offers a fast and easy access to the OPC Unified Architecture (UA) technology. Develop OPC UA 1.01, OPC UA 1.02 and OPC UA 1.03 compliant Servers with C#/VB.NET or any other compiler capable of generating a .NET assembly.

The developer can concentrate on his application and servers can be developed fast and easily without the need to spend a lot of time learning how to implement the OPC Unified Architecture specification. The server API is easy to use and many OPC specific functions are handled by the framework.

The included Model Compiler can be used to create the necessary C# classes of Information Model's specified in XML and CSV based files. At the moment the XML files must be edited by a text editor but a Model Designer is planned for the future. This will then improve code quality and programming efficiency.

The **OPC UA Server Gateway .NET** extends the OPC UA Server SDK .NET with an OPC UA server wrapper. With this Add-On OPC UA servers can also be accessed with OPC DA, AE and HDA clients. Because of the Classic OPC oriented interfaces the UA server access is limited to the Data Access, Alarms&Conditions and the Historical Access features of the UA server.

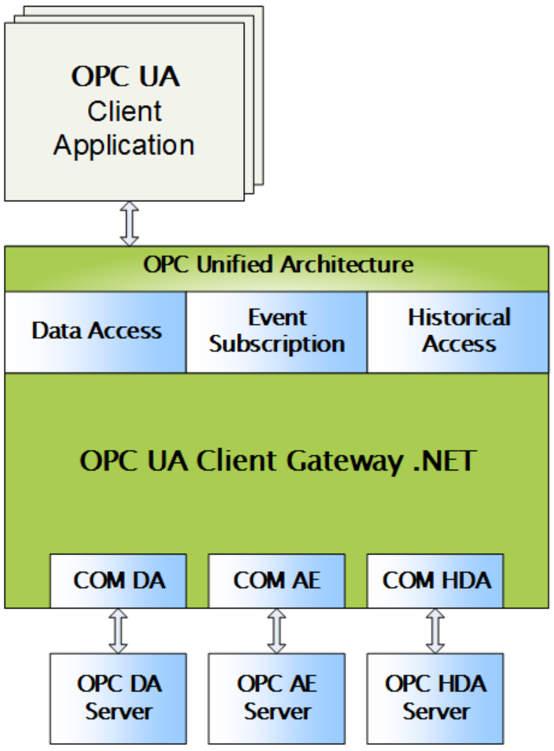
3.3 OPC UA Client Gateway .NET

You have a Classic OPC Server which is no longer supported or updated by the server vendor but need to integrate it into OPC UA? You want to eliminate the need to configure DCOM for network operation? The OPC UA Client Gateway.NET is the right choice for this.

The OPC UA Client Gateway.NET allows any OPC UA Clients to access Classic OPC DA, AE and HDA servers.

The OPC UA Client Gateway.NET can be configured with an integrated configuration tool. With this tool you can specify the Classic OPC Server to be used as well as starting node (folder) within the UA address space to be used for mapping the Classic OPC Server address space.

In general Technosoftware has tested the OPC UA Client Gateway .NET with Classic OPC servers and OPC UA clients from different vendors.



Features:

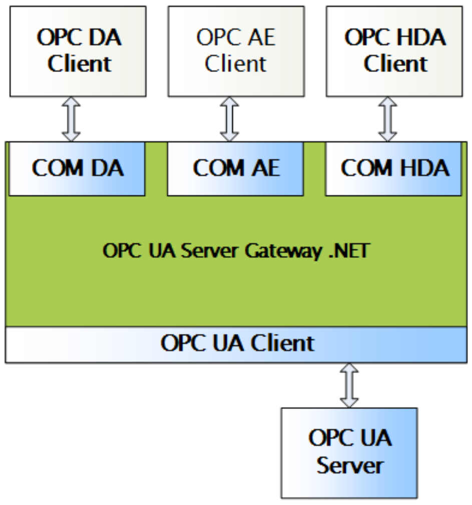
- OPC UA Client Gateway .NET is fully compliant to OPC UA 1.01 / 1.02.
- OPC UA Client Gateway .NET is fully compliant to OPC DA 2.05, OPC DA 3.0, OPC AE 1.10 and OPC HDA 1.20 servers.
- Supports 32bit and 64bit mode with the same assembly (build with Any CPU).
- OPC DA, AE and HDA address space and functionality is mapped to separate nodes within the UA address space.
- Automatically reconnect a Classic OPC Server in case connection is lost.
- Can be configured to use multiple Classic OPC Servers, each server's address space mapped to its own folder.
- Increased maintainability and usability.
- Easy to understand and well documented.

3.4 OPC UA Server Gateway SDK .NET

You have a Classic OPC Client which is no longer supported or updated by the client vendor but need to integrate it into OPC UA? You want to eliminate the need to configure DCOM for network operation? The OPC UA Proxy .NET is the right choice for this.

The OPC UA Server Gateway .NET allows any OPC DA, AE or HDA Client to access OPC UA servers. The OPC UA Server Gateway .NET can be configured with an integrated configuration tool. With this tool you can specify the OPC UA Server to be used as well as the Classic OPC Server it should be mapped to.

In general Technosoftware has tested the proxy with Classic OPC Clients and OPC UA servers from different vendors.



Features:

- Fully compliant to OPC DA 2.05, OPC DA 3.0, OPC AE 1.10 and OPC HDA 1.20.
- Fully compliant to OPC Unified Architecture 1.01 / 1.02 Data Access, Alarms& Conditions and Historical Access Profile
- Supports 32bit and 64bit mode with the same assembly (build with Any CPU).
- OPC DA, AE and HDA address space and functionality is mapped to separate Classic OPC servers.
- Increased maintainability and usability.
- Easy to understand and well documented.

4 Concepts

This chapter describes the base concepts of the OPC DA/AE/HDA Client SDK .NET and contains a lot of information. If you don't understand something don't read over it but raise a question IMMEDIATELY as this document contains no unimportant information.

4.1 APIs

The classes and interfaces defined fall into two categories separating functionality that is intended to be used inside the client process (which implies a property access may not directly result in network activity) and functionality that is intended to be used across processes.

The first category called **Client API** consists of base classes that provide transparent access to all server functions but also maintain local state information such as server assigned handles for items.

The second category called **Server API** consists primarily of interfaces and serializable objects that the individual implementations should never need to sub-class. The **Server API** is intended only for implementation by servers and the framework internal protocol specific wrappers. Client applications should never need to access the **Server API** directly.

Note that the **Client API** and **Server API** classes and interfaces are tightly coupled. For this reason, they are not separated into distinct modules or namespaces. Classes which are part of the **Client API** may be sub-classed and have properties that may be accessed without causing network activity.

For example, the OpcServer class is part of the **Client API** and wraps a remote object that implements the IOpcServer interface. Client applications should create an instance of a OpcServer class for each distinct connection they wish to establish with a remote server. The class OpcServer provides implementations of all methods defined for IOpcServer however, this class also provides other properties and methods that make the object easier to use within a client application.

4.2 Naming Conventions

The OPC DA/AE/HDA Client SDK .NET uses the following name space structure for its classes. The main name space contains all classes used by more then one specification. It is recommended to use these classes to allow clients access to several specifications with the same code.

For each specification there is another name space which contains all classes for client and server applications used by one specification.

The name spaces use the following naming conventions for all classes, enumeration, structures, interfaces and delegates:

- The names used in the main name space starts with "Opc" at the beginning to show that the definition is used by several specifications.
- The names used in the other name spaces uses no specific characters to start with.

For compliance with the OPC OPC DA/AE/HDA Client SDK .NET .NET and for easy migration the following rules apply to all specifications supported also by the OPC OPC DA/AE/HDA Client SDK .NET .NET:

- All names in the main and other name spaces uses the two character "Ts" at the beginning to identify a Technosoftware specific definition
- The third character can be "C" for a client specific definition, "S" for server specific definition or something else
- If not "C" or "S" is the third character, one of the following is used:
    - "Opc" shows that the definition is used by several specifications
    - "Da", "Xda", "Ae", "Hda" shows that the definition is used for one specification but is used for client as well as server development

The following table shows the most used definitions and the corresponding namespaces:

Name starts with	Namespace	Description
Opc	OpcClientSdk	Used by more then one specification
TsDa	OpcClientSdk.Da	Client and Server specific OPC Data Access definition
TsCDa	OpcClientSdk.Da	Client specific OPC Data Access definition
TsCAe	OpcClientSdk.Ae	Client specific OPC Alarms&Events definition
TsCHda	OpcClientSdk.Hda	Client specific OPC Historical Data Access definition
TsCCpx	OpcClientSdk.Cpx	Client specific OPC Complex Data definition

4.3 Requirements

The system requirements of the OPC DA/AE/HDA Client SDK .NET and the build applications are described in the following chapters.

4.3.1 Run-time Requirements

- Windows 10 (32bit and 64bit), Windows 7 SP1 (32bit and 64bit), Windows 8.1 (32bit and 64bit),
- Windows Server 2016 Standard and Datacenter (32bit and 64bit), Windows Server 2012 R2 (32bit and 64bit),
- .NET Framework 4.6.1
- Microsoft Visual Studio 2017 (C#/VB.NET)

Please be sure that the required software components are installed. If they are not yet installed please install the software as follows:

4.3.1.1 Install the .NET Framework

Install the .NET Framework redistributable which can be downloaded from www.microsoft.com or is installed with Microsoft Visual Studio .NET 2017.

4.3.1.2 OPC Core Components

Within the directory bin\redist\OPCCoreComponentsRedistributable and x64\redist\OPCCoreComponentsRedistributable are setup applications (setup.exe) which installs the OPC Core components required by applications based on the OPC DA/AE/HDA Client SDK .NET. If you intend to run client applications on 64bit operating systems in x64 mode you must install the OPC Core Components for x64 bit applications found in the x64\ redist\OPCCoreComponentsRedistributable directory.

The OPC Core Components consists of all shared OPC modules that need to be distributed by multiple vendors. These modules include DCOM proxy/stub libraries, the OPC Server Enumerator, .NET wrappers, etc. The OPC Core Components installer packages bundle modules from all released specifications for convenience.

4.3.1.3 Installation

All COM servers and proxy/stub libraries that have been previously released are installed in the system folder and reference counted in order to ensure compatibility with older install programs. New COM servers and proxy/stub libraries are installed in the common files folder under "OPC Foundation\Bin ".

Shared .NET assemblies are signed with the OPC Foundation's master key-pair and installed in the Global Assembly Cache (GAC).

4.3.1.4 OPC DA/AE/HDA Client SDK .NET Components

The following components are required:

• OpcClientSdk46.dll	OPC DA/AE/HDA Client SDK .NET for .NET Framework 4.6.1

You can find these files in the binaries and x86 directory. It should be copied to your application directory.

4.3.2 Development Requirements

For development one of the following compilers are supported

- Microsoft Visual Studio 2017 (C#/VB.NET)

4.4 Server Identification

The OPC DA/AE/HDA Client SDK .NET uses the following mechanism to identify a server:

- the URL syntax to identify a server. The URL must contain the protocol used to communicate with the server, the host name or IP address and a unique identifier for the server on the host machine.

The OPC DA/AE/HDA Client SDK .NET defines URL syntax for COM servers which have the following form:

```
specification://<Hostname>/<ProgID>[/CLSID]
```

Element	Description
specification://	A prefix that indicates the URL is a for a COM server; **opcda** OPC Data Access **opcae** OPC Alarms&Events **opchda** OPC Historical Data Access **opcua** OPC Unified Architecture
Hostname	The host name or IP address;
ProgID	The programmic identifier for the COM server on the host. This may be a version independent Prog ID;
CLSID	The class identifier for the COM server. This is always a GUID represented as a string with the form {00000000-0000-0000-0000-000000000000}. If this field is omitted then the client must determine the CLSID by resolving the Prog ID on the client's machine.

An example of a URL for a COM server is:

```
opcda://localhost/OpcClientSdk.DASample.80
```

The syntax for a URL for a XML web service is similar:

```
http://<Hostname>[:<Port>]/<Application Path>
```

Element	Description
http://	A prefix that indicates the URL is a for a XML web service;
Hostname	The host name or IP address;
Port	The port used by the web server. The default is 80 if this field is missing.
Application Path	The relative path of the web service on the host.

An example of a URL for an XML web service is:

```
http://localhost//OpcClientSdkXmlDaSampleServer/Service.asmx
```

The URL specifies all the information that a client needs to connect to the server; however, it is not a unique identifier for a server. The same server may have several different URLs that reference it.

4.5 Server Browsing

The OPC DA/AE/HDA Client SDK .NET API has a single OPC server discover class (OpcDiscovery) that can be used for all protocols, however, the mechanism used to discover servers is highly dependent on the protocol; as a result, the OPC DA/AE/HDA Client SDK .NET API provides the class OpcDiscovery which implements different functions for browsing OPC based servers for the different specifications.

For COM based specifications it is an in-process wrapper for the COM based Server Enumerator process (See the appropriate OPC specifications for a more detailed explanation of the OPC Server Enumerator). The client only needs one instance of this object for all hosts since the OPC DA/AE/HDA Client SDK .NET API implementation automatically connects to the COM Server Enumerator services on remote machines as necessary.

The class OpcDiscovery requires that the client choose a specific computer to look for servers. For convenience, OpcDiscovery has the method *GetHostNames* that returns all computers on the local network.

The class OpcDiscovery provides functions just returning lists of available servers for a specification, an example printing out all OPC DA 2.0 servers on the local computer is:

```
List<OpcServer> servers = OpcDiscovery.GetServers(OpcSpecification.OPC_DA_20);

if (servers != null && servers.Count > 0)
{
    CboxListServer.Text = m_sDefaultServer;
    foreach (OpcServer server in servers)
    {
        Console.WriteLine(server.ServerName);
    }
}
```

A client can also use the OpcDiscovery class which returns a set of OpcServer objects that are capable of connecting to the remote server but are not actually connected. The client should use these OpcServer objects for all communication with the remote server.

An example using the OpcDiscovery class getting an OpcServer objects for a specific server on the local computer is:

```
OpcUrl opcUrl = new OpcUrl(OpcSpecification.OPC_DA_20, OpcUrlScheme.DA,
                           "OpcClientSdk.DASample");

OpcServer server = OpcDiscovery.GetServer(opcUrl);
```

4.6 Server Connections

4.6.1 General

The OPC DA/AE/HDA Client SDK .NET API requires that clients first instantiate a sub-class of OpcServer (e.g. TsCDaServer, TsCAeServer, TsCHdaServer) that knows how to instantiate a remote server using a specific protocol and interface version. The knowledge of how to instantiate the remote server is embedded in a sub-class of OpcFactory that is passed to the OpcServer object constructor.

Note that instantiating a remote server object does not necessarily cause any network activity to occur. In some cases, like XML-DA, instantiating a remote server object only create a local proxy class that can be used to send requests to the web service. In the case of COM, instantiating the remote server only creates an instance of an in-process COM wrapper object.

The actual event of creating an instance of the remote server object occurs when the *Connect* method on the OpcServer object is called. This design allows the OpcServer object to use any additional authentication information specified by the client application when creating an instance of the server.

Note that the *Connect* method does allow the client application to specify a different URL when it connects to the server, however, this can only succeed if the URL specifies a server that can be handled by the OpcFactory object passed to the constructor of the OpcServer object.

An example using the TsCDaServer class to connect to an OPC DA server on the local computer is:

```
const string serverUrlDa = "opcda://localhost/Technosoftware.DaSample";

TsCDaServer myDaServer = new TsCDaServer();

// Connect to the DA server
myDaServer.Connect(serverUrlDa);
```

4.6.2 Security

The client application passes user authentication information in the *UserIdentity* property of the OpcConnectData class. This property contains the user name, password and domain for Windows account that the OPC DA/AE/HDA Client SDK .NET API will use to authenticate the client when connecting to the server. The OPC DA/AE/HDA Client SDK .NET API only supports integrated Windows authentication.

An example using the TsCDaServer class to connect to an OPC DA server on a remote computer with an *UserIdentity* is:

```
try
{
        OpcUrl opcUrl = new OpcUrl(OpcSpecification.OPC_DA_20, OpcUrlScheme.DA,
                                   "Technosoftware.DaSample");
        OpcUserIdentity userIdentity = new OpcUserIdentity("username","password");
        TsCDaServer myDaServer = new TsCDaServer();

        myDaServer.Connect(opcUrl, new OpcConnectData(userIdentity));

}
catch (OpcResultException e)
{
        Console.WriteLine("   " + e.Message);
}
catch(Exception e)
{
        Console.WriteLine("   " + e.Message);
}
```

4.6.3 HTTP Proxy

The client application may override the default HTTP proxy server configuration used by the OPC DA/AE/HDA Client SDK .NET API with the *SetProxy* method on the OpcConnectData class. The MSDN documentation for the *WebProxy* class has more information on HTTP proxy server configuration parameters.

4.6.4 Error Handling

4.6.4.1 OpcResult

The OpcResult class is used to indicate that an error occurred while processing a remote method call or while processing a single item during a remote method call. An OpcResult is uniquely identified by an XML qualified name and a 32 bit integer. An OpcResult may only have a value specified for only one of the identifiers. If both identifiers are specified then the 32 bit integer is used to test for equality.

These dual identifiers are necessary to accommodate vendor defined HRESULT codes which may be returned from COM servers and XML qualified name codes that may be returned from XML web services. The OPC DA/AE/HDA Client SDK .NET defines constants for all result codes that are explicitly defined in the OPC specifications. Client applications should generally use these constants when testing for specific result codes.

The OpcResult class defines the Succeeded and Failed methods to determine whether a result code represents critical or a non-critical error. An integer code that is less than zero or a qualified name beginning with 'E_' indicates a critical error. An integer code that is greater than or equal to zero or a qualified name beginning with 'S_' indicates a non-critical error.

4.6.4.2 Exceptions

The OPC DA/AE/HDA Client SDK .NET will throw exceptions whenever it encounters a critical error that prevents processing from containing. If the exception occurs because of an error returned from the server the OPC DA/AE/HDA Client SDK .NET will map that error onto an OpcResult and throw a OpcResultException.

Other types of exceptions, such as ArgumentNullException, may be thrown by the OPC DA/AE/HDA Client SDK .NET API when appropriate.

4.6.4.3 Garbage Collection

Many of the interfaces and classes defined in the OPC DA/AE/HDA Client SDK .NET API may contain references to unmanaged resources such as COM servers. For this reason, the default .NET garbage collection algorithm is not sufficient to ensure to that a client application does not have memory leaks. For this reason, all classes that reference or have a sub-class that could reference unmanaged resources have implemented the IDisposable interface. This behavior implies that the client application must explicitly call the *Dispose* method whenever it is implemented by a class in the OPC DA/AE/HDA Client SDK .NET API. If a client application creates a sub-class from a class that implements the *Dispose* method then it must ensure the base class *Dispose* method is called if it overrides the method.

Note that it is not possible to use explicit destructors to ensure that Dispose is called because the .NET framework does not allow other objects (such as COM servers) to be referenced in the *Dispose* method if it is called from an explicit destructor.

4.6.5 Item Identifiers

The OPC DA/AE/HDA Client SDK .NET API uses the OpcItem class to store attributes used to uniquely identify an OPC item. This class is a base class for a number of other item related classes. This class has four properties:

Element	Description
ItemName	This is a unique string assigned by the server that identifies an item within a server's address space. It may also be qualified with the ItemPath.
ItemPath	This is an additional string qualifier assigned by the server for an item within a server's address space.
ServerHandle	This is a unique value assigned by the server when an item is added to a group owned by a client. This identifier is only unique within the context of the item group and is not persistent. This identifier must be used when referencing items within a group.
ClientHandle	This is a unique value assigned by the client when an item is added to a group owned by the client. This value only has meaning to the client and may not be unique unless the client chooses to make it unique. This value is returned by the server with any item result when completing any group related request.

Depending on the OPC specification the *ServerHandle* and *ClientHandle* properties may be any object. This allows client applications to associate objects directly with items rather than having to maintain a lookup table for a string or integer value. Please be aware that for OPC DA, OPC AE and OPC HDA only integer objects may be used.

The documentation indicates which of the fields are required for any particular method call and what values will be in the response. The OPC DA/AE/HDA Client SDK .NET API allows client applications to control whether the *ItemName*, *ItemPath* and *ClientHandle* are returned with the results to a request.

4.7 Server API

The Server API consists primarily of interfaces and serializable objects that the individual implementations should never need to sub-class. The Server API is intended only for implementation by servers and protocol specific wrappers. Client applications should never need to access the Server API directly.

4.7.1 Interfaces

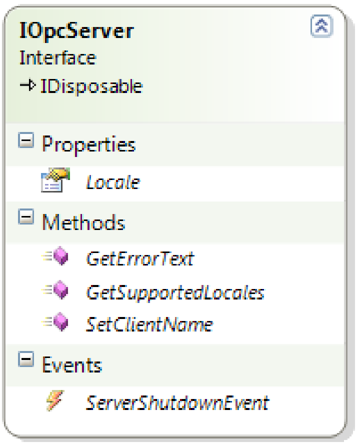
4.7.1.1 IOpcServer Interface

This interface defines functionality that is common to all OPC servers.

This interface should be implemented by .NET based servers or by the in-process wrappers for COM or SOAP/XML protocols. In both cases, a new instance of this object will be created for each client 'context'. The client context defines the scope for client state information which can be modified via this interface. In some cases (e.g. the SOAP/XML stub), a new client context will be created for each HTTP request.

The IOpcServer interface is the base for the following sub-classes:

Name	Description
OpcServer	A base class for an in-process object used to access OPC servers.



4.7.1.1.1 *Properties*

The IOpcServer interface has the following properties:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Locale</td><td>The locale used in any error messages or results returned to the client.
This method has the following parameters:
<table><tr><th>Name</th><th>Description</th></tr><tr><td>[Return Value]</td><td>The locale name in the format "[language code]-[country/region code]".</td></tr></table>All locales are identified with strings that contain a two character abbreviation for a language and an (optional) two character abbreviation for a country/region. Locales that do not have a country/region code are called 'neutral' locales. The complete set of valid language and country/region codes are derived from the ISO 639-1and ISO 3166 standards. This method throws an exception if an error occurs. Possible errors are:
<table><tr><th>Name</th><th>Description</th></tr><tr><td>E_FAIL</td><td>The operation failed.</td></tr><tr><td>E_INVALIDARG</td><td>An argument to the function was invalid. (For example, the error code specified is not valid.)</td></tr></table></td></tr>
</table>

4.7.1.1.2 *Methods*

The IOpcServer interface has the following methods:

Name	Description
GetErrorText	Returns the localized text for the specified result code. This method has the following parameters:

Name	Description
locale	The locale name in the format "[language code]-[country/region code]".
resultID	The result code identifier.
[Return Value]	A message localized for the locale that is the best match for the requested locale.

The server must use the same algorithm that it used in *SetLocale* to determine the best match for the requested locale if it does not support the requested locale for the specified result code. A server may not be able to return a properly localized error message for every result code that it returns, however, a server must always be able to return a message in its default locale. This method throws an exception if an error occurs.

Possible errors are:

Name	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (For example, the error code specified is not valid.)

Name	Description
GetSupportedLocales	Returns the locales supported by the server. This method has the following parameters:

Name	Description
[Return Value]	An array of locales with the format "[language code]-[country/region code]".

All servers must support at least one locale. In addition, the default locale for the server must be the first element in the returned array. The default locale is the locale the server will use if a client requests the invariant (“”) locale. This method throws an exception if an error occurs. Possible errors are:

Name	Description
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid. (For example, the error code specified is not valid.)

Name	Description
SetClientName	Allows the client to optionally register a client name with the server. This is included primarily for debugging purposes. The recommended behavior is that the client set his Node name and EXE name here. This method throws an exception if an error occurs. Possible errors are:

Name	Description
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid. (For example, the error code specified is not valid.)

4.7.1.1.3 Events

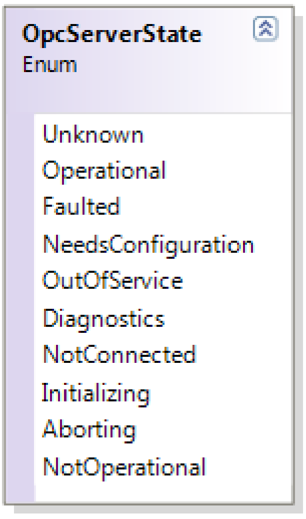
The IOpcServer interface has the following events:

Name	Description
ServerShutdownEvent	An event to receive server shutdown notifications.

4.7.2 Enumerations

4.7.2.1 OpcServerState enumeration

This enumeration defines the set of possible server states.



4.7.2.1.1 Properties

The OpcServerState enumeration has the following properties:

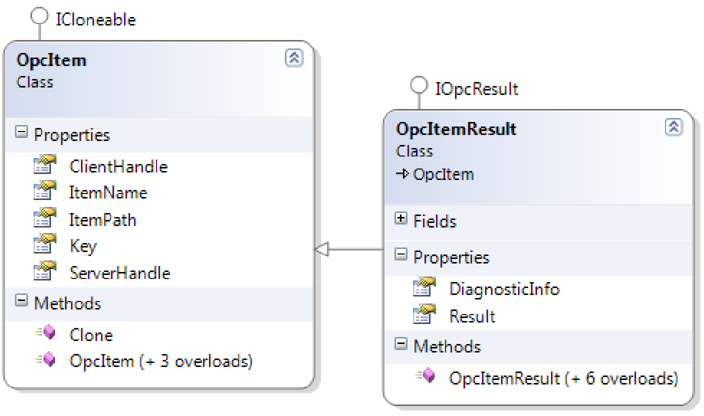
Name	Description
Unknown	The server state is not known.
Operational	The server is running normally. This is the usual state for a server.
Faulted	The server is not operational due to a fault. The server is no longer functioning. The recovery procedure from this situation is vendor specific. An error code of E_FAIL should generally be returned from any other server method.
NeedsConfiguration	The server is running but has no configuration information loaded and thus cannot function normally. Note this state implies that the server needs configuration information in order to function. Servers which do not require configuration information should not return this state.
OutOfService	The server has been temporarily suspended via some vendor specific method and is not getting or sending data. Note that Quality will be returned as OPC_QUALITY_OUT_OF_SERVICE.
Diagnostics	The server is in Diagnostics Mode. The outputs are disconnected from the real hardware but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.
NotConnected	The server is not operational. The outputs are disconnected from the real hardware but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.
Initializing	The server is not operational because it is starting up.
Aborting	The server is operational but it is shutting down and aborting all of its client contexts.
NotOperational	The server is not operational, but the reason is not known.

4.7.3 Classes

4.7.3.1 OpcItem Class

This class represents a unique item identifier.

All OPC servers represent underlying physical data points as uniquely identifiable items within a structured address space. Each of these items may have one or more identifiers associated with it that can be used by a client to access specific data points via the OPC interfaces. The OpcItem class contains different unique identifiers that may be useful in different contexts. This class is intended to be used as a based class for other item related classes.



4.7.3.1.1 Properties

The OpcItem class has the following properties:

Name	Description
ClientHandle	A unique identifier for an item assigned by the client. The ClientHandle only has meaning when an item is added to a set (such as a data subscription) defined by the client. The server will always return the ClientHandle with the results of any operation related to that set of items.
ItemName	The primary identifier for an item within the server namespace. A null or empty string is not a valid value. The ItemName and ItemPath uniquely identify an item within a server.
ItemPath	The secondary identifier for an item within the server namespace. The ItemName and ItemPath uniquely identify an item within a server.
ServerHandle	A unique identifier for an item assigned by the server. The ServerHandle only has meaning when an item is added to a set (such as a data subscription) defined by the client. The client must always provide the ServerHandle when requesting access to an item within the set.

The OpcItemResult class extends this call by adding the following properties:

Name	Description
DiagnosticInfo	Vendor specific diagnostic information (not the localized error text).
Result	The error id for the result of an operation on an item.

4.8 Client API

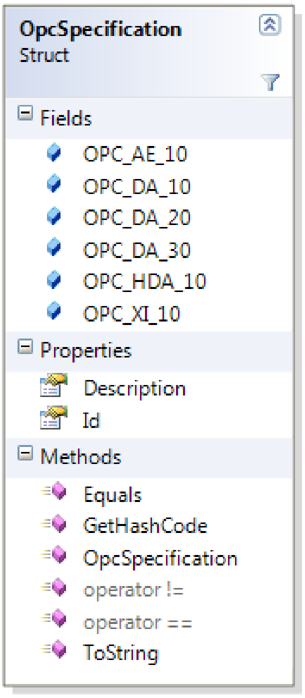
The Client API consists of a set of class that always resides inside the client process. These classes provide access to the remote servers and maintain client side state information. The main objective of the Client API is to provide a convenient, easy to use .NET programming interface for .NET client development. It comprises mainly of serializable classes that implement all of the features of the remote servers as well as objects that track information like server item handles.

This chapter describes the base concepts of the Client API and explains the main classes of it and is important for the understanding of the specification specific classes. Description of the specification specific classes, e.g. TsCDaServer can be found later in this document.

4.8.1 Structures

4.8.1.1 OpcSpecification Structure

This structure defines the different specification and is be used for browsing of OPC servers.



4.8.1.1.1 Properties

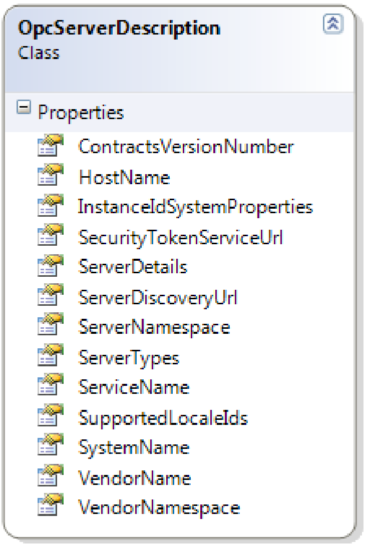
The OpcSpecification structure has the following fields:

Name	Description
OPC_AE_10	OPC Alarms&Events 1.0 OPC Alarms&Events 1.1
OPC_DA_10	OPC Data Access 1.0
OPC_DA_20	OPC Data Access 2.0
OPC_DA_30	OPC Data Access 3.0
OPC_HDA_10	OPC Historical Data Access 1.0
OPC_XI_10	OPC Xi 1.0

4.8.2 Classes

4.8.2.1 OpcServerDescription Class

This class store OPC server information like name of computer and is included in the OpcServer class.



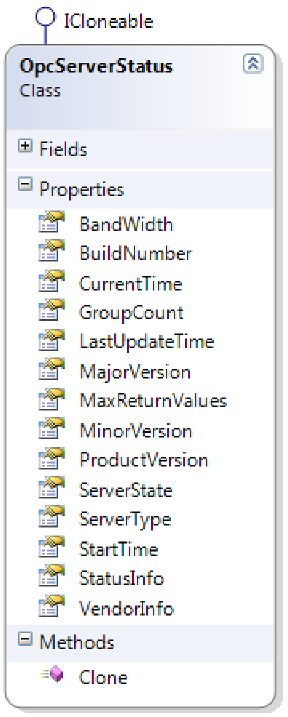
4.8.2.1.1 Properties

The OpcServerDescription class has the following properties:

Name	Description
ContractsVersionNumber	The version number of the Contracts used by this server. Only used by OPC Xi.
HostName	The HostName of the machine in which the server resides (runs).
InstanceIdSystemProperties	The list of InstanceId System property values supported by the server. Only used by OPC Xi.
SecurityTokenServiceUrl	The URL for the Security Token Service. Only used by OPC Xi.
ServerDetails	Detailed information about the server. See OpcServerDetail class.
ServerNamespace	Namespace for server-specific types. Null or empty if not used. Only used by OPC Xi.
ServerTypes	The server types supported by this server. Standard types are defined by the OpcServerType class. Only used by OPC Xi.
ServiceName	The name of the WCF service provided by the server. Only used by OPC Xi.
SupportedLocaleIds	Supported locale ids (the native language is first entry). Only used by OPC Xi.
SystemName	The name of the system that contains the objects accessible through the server. Null or empty if the server provides access to objects from more than one system.Only used by OPC Xi.
VendorName	Name of the server software vendor. Only used by OPC Xi.
VendorNamespace	Namespace for types defined by this vendor. This may or may not be the same as the VendorName. Null or empty if not used.Only used by OPC Xi.

4.8.2.2 Opc ServerStatus class

This class contains properties that describe the current status of an OPC Server.



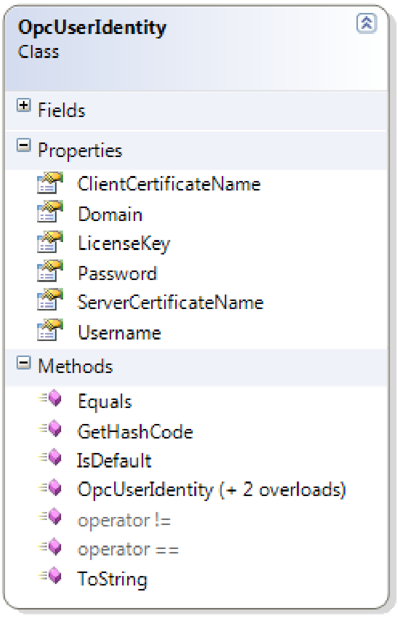
4.8.2.2.1 Properties

The OpcServerStatus class has the following properties:

Name	Description
Bandwith	The behavior of this field is server specific. A suggested use is that it returns the approximate Percent of Bandwidth currently in use by server. If multiple links are in use it could return the 'worst case' link. Note that any value over 100% indicates that the aggregate combination of items and UpdateRate is too high. The server may also return 0xFFFFFFFF if this value is unknown.
BuildNumber	The update rate in ms for the Status Update Thread.
CurrentTime	The current time (UTC) as known by the server.
GroupCount	The total number of groups being managed by the server instance. This is mainly for diagnostic purposes.
LastUpdateTime	The time (UTC) the server sent the last data value update to this client. This value is maintained on an instance basis.
MajorVersion	The major version of the server software
MaxReturnValues	The maximum number of values that can be returned by the server on a per item basis.
MinorVersion	The minor version of the server software
ProductVersion	The 'build number' of the server software
ServerState	The current status of the server. Refer to OpcServerState values in Section **Error! Reference source not found.**.
ServerType	The server for which the status is being reported. The ServerType enumeration is used to identify the server. If the enumeration indicates multiple server types, then this is the status of the entire server. For example, if the server wraps an OPC DA and OPC AE server, then if this ServerType indicates both, the status is for the entire server, and not for an individual wrapped server.
StartTime	Time (UTC) the server was started. This is constant for the server instance and is not reset when the server changes states. Each instance of a server should keep the time when the process started.
StatusInfo	A string that describes the current server state.
VendorInfo	Vendor specific string providing additional information about the server. It is recommended that this mention the name of the company and the type of device(s) supported.

4.8.2.3 OpcUserIdentity Class

This class store user identity like username and password and is used when connecting to an OPC Server.



4.8.2.3.1 Properties

The OpcUserIdentity class has the following properties:

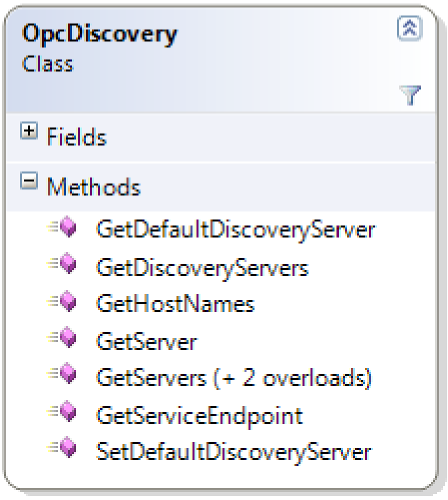
Name	Description
ClientCertificateName	The Client Certificate name for certificate mode authentication of the server access. Only used by OPC Xi.
Domain	The windows domain name.
LicenseKey	Gets or sets a license key to use when activating the server. Not used at the moment.
Password	The password.
ServerCertificateName	The Server Certificate name for certificate mode authentication of the server access. Only used by OPC Xi.
Username	The username.

4.8.2.3.2 Methods

The OpcUserIdentity class has the following methods:

Name	Description
Equals	Determines if the object is equal to the specified value. This method has the following parameters: **Name** / **Description** target: The OpcUserIdentity to compare with. [Return Value]: True if the objects are equal; otherwise false.
GetHashCode	Returns a suitable hash code for the result.
IsDefault	Enumerates hosts that may be accessed for server discovery. This method has the following parameters: **Name** / **Description** [Return Value]: Names of found computers as list of strings.
operator !=	Returns true if the objects are not equal. This method has the following parameters: **Name** / **Description** a: The first OpcUserIdentity to compare. b: The second OpcUserIdentity to compare. [Return Value]: True if the objects are not equal; otherwise false.
operator ==	Returns true if the objects are equal. This method has the following parameters: **Name** / **Description** a: The first OpcUserIdentity to compare. b: The second OpcUserIdentity to compare. [Return Value]: True if the objects are equal; otherwise false.
ToString	Converts the object to a string used for display.

4.8.2.4 OpcDiscovery Class



This class provides functions for browsing OPC based servers for the different specifications.

For COM based specifications it is an in process wrapper for the COM based Server Enumerator process (See the appropriate OPC specifications for a more detailed explanation of the OPC Server Enumerator). The client only needs one instance of this object for all hosts since the OPC DA/AE/HDA Client SDK .NET API implementation automatically connects to the COM Server Enumerator services on remote machines as necessary.

The class OpcDiscovery requires that the client choose a specific computer to look for servers. For convenience, OpcDiscovery has the method *GetHostNames* that returns all computers on the local network.

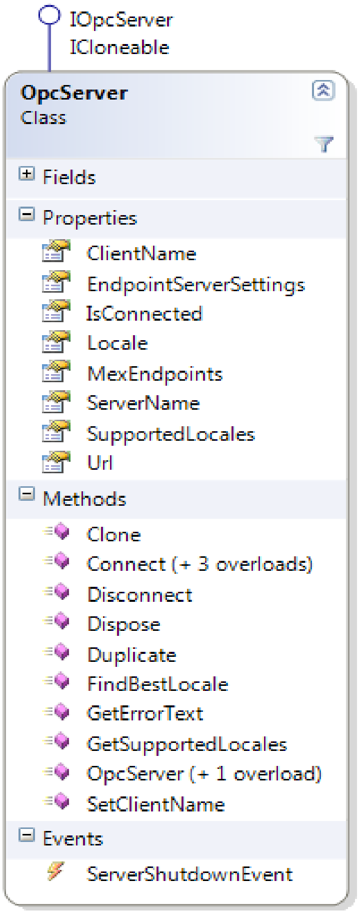
4.8.2.4.1 Methods

The OpcDiscovery class has the following methods:

Name	Description		
GetDefaultDiscoveryServer	Get's the default discovery server for the specified specification. Only used for OPC Xi. This method has the following parameters:		
		Name	**Description**
		specification	Unique identifier for one OPC specification. See also OpcSpecification object.
		[Return Value]	Default discovery server for the specified specification as string.
GetDiscoveryServers	Get's a list of available discovery servers for the specified OPC specification. Only used for OPC Xi. This method has the following parameters:		
		Name	**Description**
		specification	Unique identifier for one OPC specification. See also OpcSpecification object.
		[Return Value]	Names of found discovery servers as list of strings.
GetHostNames	Enumerates hosts that may be accessed for server discovery. This method has the following parameters:		
		Name	**Description**
		[Return Value]	Names of found computers as list of strings.
GetServer	Creates a server object for the specified URL. This method has the following parameters:		
		Name	**Description**
		url	The OpcUrl of the OPC server.
		[Return Value]	The OpcServer object.
GetServers	Returns a list of servers that support the specified specification. This method has the following parameters:		
		Name	**Description**
		specification	Unique identifier for one OPC specification. See also OpcSpecification object.
		[Return Value]	The found OPC servers as list of OpcServer objects..

		Name	Description
GetServers	Returns a list of servers that support the specified specification. This method has the following parameters:	url	The URL of the discovery server to be used as string.
		specification	Unique identifier for one OPC specification. See also OpcSpecification object.
		[Return Value]	The found OPC servers as list of OpcServer objects.
GetServers	Returns a list of servers that support the specified specification. This method has the following parameters:	url	The URL of the discovery server to be used as string.
		userIdentity	The user identity to use when discovering the servers as OpcUserIdentity.
		specification	Unique identifier for one OPC specification. See also OpcSpecification object.
		[Return Value]	The found OPC servers as list of OpcServer objects.
SetDefaultDiscoveryServer	Set's the default discovery server for the specified specification. Only used for OPC Xi. This method has the following parameters:	specification	Unique identifier for one OPC specification. See also OpcSpecification object.
		url	Default discovery server for the specified specification.

4.8.2.5 OpcServer Class



This class is a base class for an in-process object used to access OPC servers and is an in-process wrapper for a remote server (i.e. a server that implements the IOpcServer interface). This class provides a mechanism to cache properties of the remote server locally for fast access and supports serialization (which simplifies the task of saving client configuration information).

The OPC DA/AE/HDA Client SDK .NET API also provides standard sub-classes for different OPC specifications. A client may choose to create their own sub-classes of this class (or its sub-classes) to handle application specific server data.

This object contains references to unmanaged resources (e.g. COM servers), as a result, this object must be explicitly released by calling the *Dispose* method. A call to the *Dispose* method is automatically done by calling the *Disconnect* method.

The OpcServer class is the base class for the following sub-classes:

Name	Description
TsCDaServer	This class is the main interface to access an OPC Data Access server
TsCAeServer	This class is the main interface to access an OPC Alarms&Events server
TsCHdaServer	This class is the main interface to access an OPC Historical Data Access server

4.8.2.5.1 Properties

The OpcServer class has the following properties:

Name	Description
ClientName	A short descriptive name for the server assigned by the client.
EndpointServerSettings	Endpoint configuration settings that are not in the endpoint metadata. OPC Xi only.
IsConnected	Whether the remote server is currently connected.
Locale	Gets/Sets the locale used in any error messages or results returned to the client. The locale name is in the format "[languagecode]-[country/regioncode]"
MexEndpoints	The names of available metaDataExchange endpoints. These names can only be used as a selection choice for the client. The Mex endpoint communication settings must be standardized. OPC Xi only.
ServerName	A short descriptive name for the server.
SupportedLocales	The set of locales supported by the remote server.
Url	The OpcUrl that describes the network location of the server.

4.8.2.5.2 *Methods*

The OpcServer class has the following methods:

Name	Description
Clone	Returns an unconnected copy of the server with the same URL.
Connect	Establishes a physical connection to the remote server. This method has the following parameters: **Name** — **Description** url — The network address for the remote server. It replaces the default URL for the server if the method succeeds. connectData — Any protocol configuration or user authentication information.
Disconnect	Disconnects from the server and releases all network resources. Also Dispose() is called.
Dispose	This must be called explicitly by clients to ensure the remote server is released. Is automatically called by the Disconnect method.
Duplicate	Creates a new instance of a server object with the same factory and url. This method is not the same as the Clone method since it does not copy the values of any properties to the new object (for example, existing subscriptions are not copied). The server object requires that the Dispose method be called explicitly, as a result, client applications which have code (such as a user controls or forms) that access a server in different contexts, may find it simpler to duplicate the server object rather than keep track of references across multiple controls. This method has the following parameters: **Name** — **Description** [Return Value] — A duplicate of the server object.
FindBestLocale	Finds the best matching locale given a set of supported locales. This method has the following parameters: **Name** — **Description** [Return Value] — The locale name in the format "[language code]-[country/region code]". All locales are identified with strings that contain a two character abbreviation for a language and an (optional) two character abbreviation for a country/region. Locales that do not have a country/region code are called ‘neutral’ locales. The complete set of valid language and country/region codes are derived from the ISO 639-1and ISO 3166 standards.

GetErrorText (IOpcServer)	Returns the localized text for the specified result code. This method has the following parameters:

Name	Description
locale	The locale name in the format "[language code]-[country/region code]".
resultID	The result code identifier.
[Return Value]	A message localized for the locale that is the best match for the requested locale.

The server must use the same algorithm that it used in *SetLocale* to determine the best match for the requested locale if it does not support the requested locale for the specified result code. A server may not be able to return a properly localized error message for every result code that it returns, however, a server must always be able to return a message in its default locale. This method throws an exception if an error occurs. Possible errors are:

Name	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (For example, the error code specified is not valid.)

GetSupportedLocales (IOpcServer)	Returns the locales supported by the server This method has the following parameters:

Name	Description
[Return Value]	An array of locales with the format "[language code]-[country/region code]".

All servers must support at least one locale. In addition, the default locale for the server must be the first element in the returned array. The default locale is the locale the server will use if a client requests the invariant (“”) locale. This method throws an exception if an error occurs. Possible errors are:

Name	Description
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid. (For example, the error code specified is not valid.)

SetClientName (IOpcServer)	Allows the client to optionally register a client name with the server. This is included primarily for debugging purposes. The recommended behavior is that the client set his Node name and EXE name here. This method throws an exception if an error occurs. Possible errors are:

Name	Description
E_FAIL	The operation failed.
E_INVALIDARG	An argument to the function was invalid. (For example, the error code specified is not valid.)

4.8.2.5.3 Events

The OpcServer class has the following events:

Name	Description
ServerShutdownEvent (IOpcServer)	An event to receive server shutdown notifications.

5 OPC DA Client Development

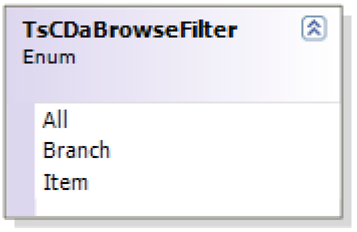
This chapter describes the OPC Data Access specific classes of the **Server API** and **Client API.** Knowledge of the corresponding specifications is required for the understanding of this chapter.

5.1 Server API

5.1.1 Enumerations

5.1.1.1 TsCDaBrowseFilter enumeration

This enumeration defines the type of browse elements to return during a browse.


TsCDaBrowseFilter
Enum
All
Branch
Item

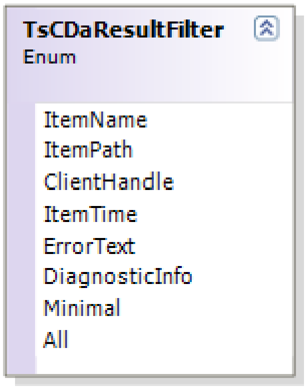
5.1.1.1.1 Properties

The TsCDaBrowseFilter enumeration has the following properties:

Name	Description
All	Return all types of browse elements.
Branch	Return only elements that contain other elements.
Item	Return only elements that represent items.

5.1.1.2 TsCDaResultFilter enumeration

This enumeration defines the filters applied by the server before returning item results.


TsCDaResultFilter
Enum
ItemName
ItemPath
ClientHandle
ItemTime
ErrorText
DiagnosticInfo
Minimal
All

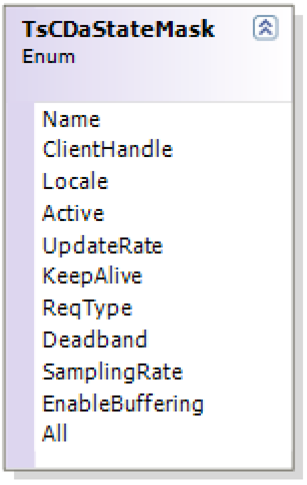
a. Properties

The TsCDaResultFilter enumeration has the following properties:

Name	Description
ItemName	Include the ItemName in the ItemIdentifier if bit is set.
ItemPath	Include the ItemPath in the ItemIdentifier if bit is set.
ClientHandle	Include the ClientHandle in the ItemIdentifier if bit is set.
ItemTime	Include the Timestamp in the ItemValue if bit is set.
ErrorText	Include verbose, localized error text with result if bit is set.
DiagnosticInfo	Include additional diagnostic information with result if bit is set.
Minimal	Include the ItemName and Timestamp if bit is set.
All	Include all information in the results if bit is set.

5.1.1.3 TsCDaStateMask enumeration

This enumeration defines masks to be used when modifying the subscription or item state.



5.1.1.4 Properties

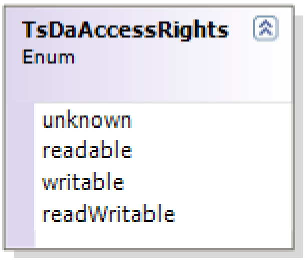
The TsCDaStateMask enumeration has the following properties:

Name	Description
Name	The name of the subscription.
ClientHandle	The client assigned handle for the item or subscription.
Locale	The locale to use for results returned to the client from the subscription.
Active	Whether the item or subscription is active.
UpdateRate	The maximum rate at which data update notifications are sent.
KeepAlive	The longest period between data update notifications. **Note:** This feature is only supported with OPC Data Access 3.0 Servers.
ReqType	The requested data type for the item.
Deadband	The deadband for the item or subscription.
SamplingRate	The rate at which the server should check for changes to an item value.
EnableBufferung	Whether the server should buffer multiple changes to a single item.
All	All fields are valid.

5.1.1.5 TsDaAccessRights enumeration

This enumeration defines the possible item access rights.

Indicates if this item is read only, write only or read/write. This is NOT related to security but rather to the nature of the underlying hardware.



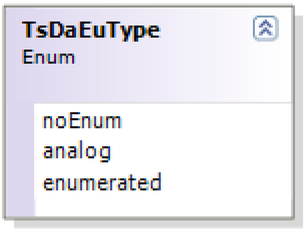
5.1.1.5.1 *Properties*

The TsDaAccessRights enumeration has the following properties:

Name	Description
unknown	The access rights for this item are unknown.
readable	The client can read the data item's value.
writable	The client can change the data item's value.
readWritable	The client can read and change the data item's value.

5.1.1.6 TsDaEuType enumeration

This enumeration defines the possible item engineering unit types.



5.1.1.6.1 *Properties*

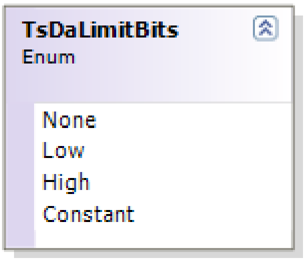
The TsDaEuType enumeration has the following properties:

Name	Description
noEnum	No engineering unit information available
analog	Analog engineering unit - will contain a SAFEARRAY of exactly two doubles (VT_ARRAY | VT_R8) corresponding to the LOW and HI EU range.
enumerated	Enumerated engineering unit - will contain a SAFEARRAY of strings (VT_ARRAY | VT_BSTR) which contains a list of strings (Example: “OPEN”, “CLOSE”, “IN TRANSIT”, etc.). Corresponding to sequential numeric values (0, 1, 2, etc.)

5.1.1.7 TsDaLimitBits enumeration

This enumeration defines the possible limit status bits.

The Limit Field is valid regardless of the Quality and Substatus. In some cases such as Sensor Failure it can provide useful diagnostic information.



TsDaLimitBits
Enum

None
Low
High
Constant

5.1.1.7.1 *Properties*

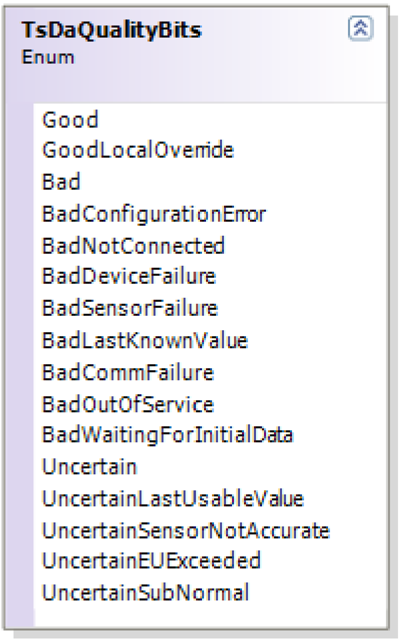
The TsDaLimitBits enumeration has the following properties:

Name	Description
None	The value is free to move up or down
Low	The value has 'pegged' at some lower limit
High	The value has 'pegged' at some high limit
Constant	The value is a constant and cannot move

5.1.1.8 TsDaQualityBits enumeration

This enumeration defines the possible quality status bits.

These flags represent the quality state for an item's data value. This is intended to be similar to but slightly simpler than the Fieldbus Data Quality Specification (section 4.4.1 in the H1 Final Specifications). This design makes it fairly easy for both servers and client applications to determine how much functionality they want to implement. The Limit Field is valid regardless of the Quality and Substatus. In some cases such as Sensor Failure it can provide useful diagnostic information.



TsDaQualityBits
Enum

Good
GoodLocalOverride
Bad
BadConfigurationError
BadNotConnected
BadDeviceFailure
BadSensorFailure
BadLastKnownValue
BadCommFailure
BadOutOfService
BadWaitingForInitialData
Uncertain
UncertainLastUsableValue
UncertainSensorNotAccurate
UncertainEUExceeded
UncertainSubNormal

5.1.1.8.1 *Properties*

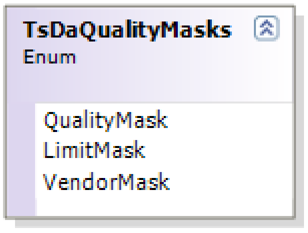
The TsDaQualityBits enumeration has the following properties:

Name	Description
Good	The Quality of the value is Good.
GoodLocalOverride	The value has been Overridden. Typically this means the input has been disconnected and a manually entered value has been 'forced'.
Bad	The value is bad but no specific reason is known.
BadConfigurationError	There is some server specific problem with the configuration. For example the item in question has been deleted from the configuration.
BadNotConnected	The input is required to be logically connected to something but is not. This quality may reflect that no value is available at this time, for reasons like the value may have not been provided by the data source.
BadDeviceFailure	A device failure has been detected.

BadSensorFailure	A sensor failure had been detected (the 'Limits' field can provide additional diagnostic information in some situations).
BadLastKnownValue	Communications have failed. However, the last known value is available. Note that the 'age' of the value may be determined from the time stamp in the item state.
BadCommFailure	Communications have failed. There is no last known value is available.
BadOutOfService	The block is off scan or otherwise locked. This quality is also used when the active state of the item or the group containing the item is InActive.
BadWaitingForInitialData	After Items are added to a group, it may take some time for the server to actually obtain values for these items. In such cases the client might perform a read (from cache), or establish a ConnectionPoint based subscription and/or execute a Refresh on such a subscription before the values are available. This substatus is only available from OPC DA 3.0 or newer servers.
Uncertain	There is no specific reason why the value is uncertain.
UncertainLastUsableValue	Whatever was writing this value has stopped doing so. The returned value should be regarded as 'stale'. Note that this differs from a BAD value with Substatus badLastKnownValue (Last Known Value). That status is associated specifically with a detectable communications error on a 'fetched' value. This error is associated with the failure of some external source to 'put' something into the value within an acceptable period of time. Note that the 'age' of the value can be determined from the time stamp in the item state.
UncertainSensorNotAccurate	Either the value has 'pegged' at one of the sensor limits (in which case the limit field should be set to low or high) or the sensor is otherwise known to be out of calibration via some form of internal diagnostics (in which case the limit field should be none).
UncertainEUExceeded	The returned value is outside the limits defined for this parameter. Note that in this case (per the Fieldbus Specification) the 'Limits' field indicates which limit has been exceeded but does NOT necessarily imply that the value cannot move farther out of range.
UncertainSubNormal	The value is derived from multiple sources and has less than the required number of Good sources.

5.1.1.9 TsDaQualityMasks enumeration

This enumeration defines bit masks for the quality.



5.1.1.9.1 Properties

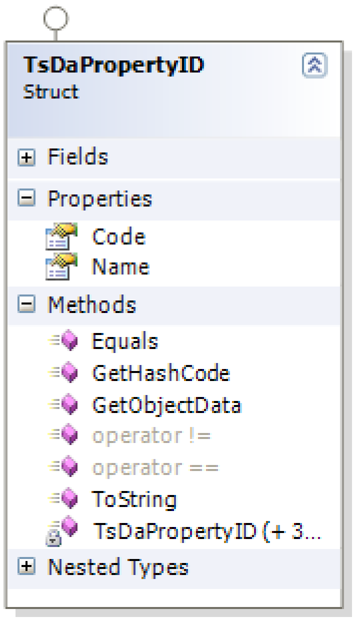
The TsDaQualityMasks enumeration has the following properties:

Name	Description
QualityMask	Quality related bits
LimitMask	Limit related bits
VendorMask	Vendor specific bits

5.1.2 Structures

5.1.2.1 TsDaPropertyID Structure

This structure contains a unique identifier for a property.



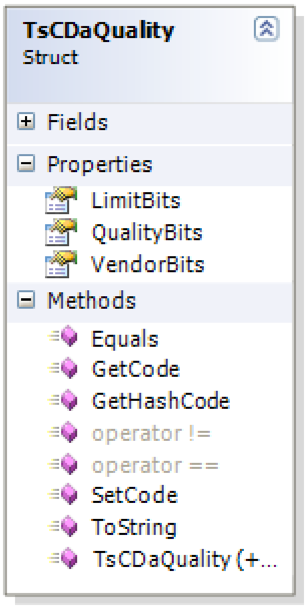
5.1.2.1.1 Properties

The TsDaPropertyID structure has the following fields:

Name	Description
Code	Used for properties identified by a integer.
Name	Used for properties identified by a qualified name.

5.1.2.2 TsCDaQuality Structure

This structure contains the quality field for an item value.



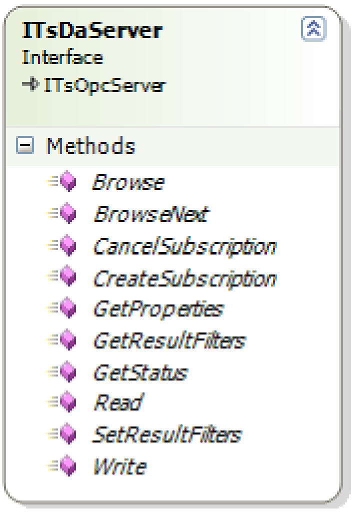
5.1.2.2.1 Properties

The TsCDaQuality structure has the following fields:

Name	Description
LimitBits	The value in the limit bits field.
QualityBits	The value in the quality bits field.
VendorBits	The value in the quality bits field.

5.1.3 Interfaces

5.1.3.1 ITsDaServer Interface



This interface defines functionality specific to OPC Data Access servers.

This interface inherits from the IOpcServer interface.

This interface is intended to merge the functionality defined by the OPC DA specifications into a single interface based on the .NET programming model.

The ITsDaServer interface is the base for the following sub-classes:

Name	Description
TsCDaServer	This class is a base class for an in-process object used to access OPC Data Access servers

5.1.3.1.1 *Methods*

The ITsDaServer interface has the following methods:

Name	Description
Browse	This method fetches the children of a branch that meet the filter criteria. This method has the following parameters:

Name	Description
itemID	The identifier of branch which is the target of the search. The ClientHandle and ServerHandle have no meaning in this context. Passing a null value searches for elements with no parent (e.g. the top of tree).
filters	The filters to use to limit the set of child elements returned. The TsCDaBrowseFilters object is described in section 5.1.4.3.
position	An object used to continue a browse operation A browse operation may not complete if the number of elements exceeds the value of the MaxElementsReturned filter. The client may continue the browse by calling *BrowseNext*, otherwise the client must call *Dispose* on the TsCDaBrowsePosition object to ensure that all resources allocated for the browse are released. A server will typically create sub-classes of the TsCDaBrowsePosition object that contain information used to optimize *BrowseNext* operations. This object has no properties that are visible to clients.
[Return Value]	The set of elements found.

BrowseNext	This method continues a browse operation with previously specified search criteria. This method has the following parameters:

Name	Description
position	An object containing the browse operation state information. This object must be returned from a call to Browse. If the position is invalid for any reason this method throws a OpcResultException exception. If there are no more elements to fetch this method will set the TsCDaBrowsePosition to null. Otherwise, this method will return a new TsCDaBrowsePosition object.
[Return Value]	The set of elements found.

CancelSubscription	This method cancels a subscription and releases all resources allocated for it. Clients must always explicitly cancel all subscriptions that it creates. This method has the following parameters:

Name	Description
subscription	The subscription to cancel.

CreateSubscription	This method creates a new subscription. A subscription allows a client to receive asynchronous notifications from the server whenever an item value changes. All subscriptions that a client creates must be destroyed with the *CancelSubscription* method. The SOAP/XML protocol introduces some complexity with regards to subscriptions because no other method requires that the server maintain state information across method calls. The SOAP/XML stub resolves this issue by managing the references to the ITsDaServer and ITsCDaSubscription objects on behalf of the remote client. This method has the following parameters:

Name	Description
state	The initial state of the subscription. The TsCDaSubscriptionState object is described below.
[Return Value]	The new subscription object.

GetProperties

This method returns the item properties for a set of items.

This method has the following parameters:

Name	Description
itemIDs	A list of item identifiers.
propertyIDs	A list of properties to fetch for each item. If this parameter is null then all available properties are returned.
returnValues	Whether the property values should be returned with the properties.
[Return Value]	A list of properties for each item. The TsCDaItemPropertyCollection object is described below.

The TsCDaItemPropertyCollection object extends *ArrayList* and has the following properties/methods:

Name	Description
ItemName	The primary identifier for the item within the server namespace.
ItemPath	The secondary identifier for the item within the server namespace.
Result	A result code that indicates any item-level errors.
operator[]	Returns the TsCDaItemProperty object at the specified index.

GetResultFilters

This method returns the filters applied by the server to any item results returned to the client.

This method has the following parameters:

Name	Description
[Return Value]	A bit mask indicating which fields should be returned in any item results.

The set of masks is has the following values:

Name	Value	Description
ItemName	0x01	Include the ItemName in the OpcItem if bit is set.
ItemPath	0x02	Include the ItemPath in the OpcItem if bit is set.
ClientHandle	0x04	Include the ClientHandle in the OpcItem if bit is set.
ItemTime	0x08	Include the Timestamp in the *ItemValue* if bit is set.
ErrorText	0x10	Include verbose, localized error text with result if bit is set.
DiagnosticInfo	0x20	Include additional diagnostic information with result if bit is set.
Minimal	0x09	Include the ItemName and Timestamp if bit is set.
All	0xFF	Include all information in the results if bit is set.

Note that the ClientHandle property of and OpcItem has no meaning when used at the server level.

The filters only affect results returned from the Read and Write methods. They are also used as the default for new Subscriptions.

GetStatus

This method returns the current status of the server.

This method has the following parameters:

Name	Description
[Return Value]	The current server status.

The server status is an object that has the following properties:

Name	Description
VendorInfo	The vendor name and product name for the server.
ProductVersion	The vendor's software version number.
ServerState	The current server state (see the server state enumeration below).
StatusInfo	More information about the current server state.
StartTime	The UTC time when the server started.
CurrentTime	The current UTC time at the server.
LastUpdateTime	The last time the server sent a data update to the client.

The server state enumeration has the following values:

Name	Description
Unknown	The server state is not known.
Running	The server is running normally.
Failed	The server is not functioning due to a fatal error.
NoConfig	The server cannot load its configuration information.
Suspended	The server has halted all communication with the underlying hardware.
Test	The server is disconnected from the underlying hardware.
CommFault	The server cannot communicate with the underlying hardware.

Read

The method reads the current values for a set of items.

This method has the following parameters:

Name	Description
items	The set of items to read. Each item must have an ItemName Each item may have an ItemPath, a ReqType or MaxAge.
[Return Value]	The results of the read operation for each item.

The number of item values returned must equal the number of items passed to the method. The client uses the index in the arrays to match a item value with the item. The server indicates errors on individual items by returning the appropriate result code as part of the item value.

It is up to the server to decide whether a cache read is appropriate for a given item, as a result, a server may choose to read directly from the device even if the client requests a MaxAge of *Int32.MaxValue*.

SetResultFilters	This method sets the filters applied by the server to any item results returned to the client. This method has the following parameters:

Name	Description
filters	A bit mask indicating which fields should be returned in any item results.

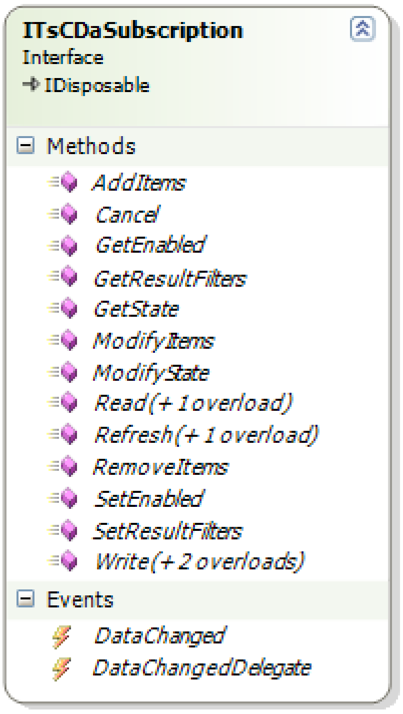
Write	This method writes the value, quality and timestamp for a set of items. This method has the following parameters:

Name	Description
itemValues	The set of item values to write. Each item must have an ItemName and a Value. Each item may have an ItemPath, a Quality and a Timestamp.
[Return Value]	The results of the write operation for each item.

	The number of item results returned must equal the number of item values passed to the method. The client uses the index in the arrays to match a item result with the item value. The server indicates errors on individual items by returning the appropriate result code as part of the item value. The server may support writing to the quality and/or timestamp. In these cases, the server does not write the value and returns 'E_NO_WRITEQT' for the item.

5.1.3.2 ITsCDaSubscription Interface

This interface allows clients to control subscriptions to data items exposed by a Data Access server.



The OPC DA/AE/HDA Client SDK .NET subscription mechanism unifies the group/subscription mechanisms from COM-DA. The OPC DA/AE/HDA Client SDK .NET API implements a fully asynchronous callback.

The ITsCDaSubscription interface is the base for the following sub-classes:

Name	Description
TsCDaSubscription	This class is an in-process object used to access subscriptions on OPC Data Access servers.

5.1.3.2.1 Methods

The ITsDaSubscription interface has the following methods:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>AddItems</td><td>This method adds items to the subscription.
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>items</td><td>The set of items to add to the subscription.
The <i>TsCDaItem</i> object is described in Section 5.1.4.1.
Each item must have an ItemName
Each item may have the ItemPath, ClientHandle, ReqType, Active, SamplingRate or EnableBuffering properties specified.</td></tr>
<tr><td>[Return Value]</td><td>The results of the add item operation for each item.
In some cases, the server will not be able to satisfy the client request for some of the item state parameters (e.g. individual item sampling rates might not be supported). The <i>ItemResult</i> object contains the actual item.
The <i>TtemResult</i> object also contains a ServerHandle which the client must use to reference the item in other methods on the subscription.</td></tr>
</table>
The index in the array is used to associate a result with a specific item.</td></tr>
</table>

Cancel

This method cancels an asynchronous read or writes operation.

This method takes the following parameters:

Name	Description
request	The object returned from the Read or Write request.
callback	The function to invoke when the cancel completes.

GetEnabled

This method checks whether data change notifications from the server are enabled.

This method takes the following parameters:

Name	Description
[Return Value]	Whether data changed notifications should be sent.

GetResultFilters

This method returns the filters applied by the server to any item results returned to the client.

This method is the same as the method described in Section 5.1.3.1.

The filters specified at the subscription level override filters specified at the server level. This method takes the following parameters:

Name	Description
[Return Value]	A bit mask indicating which fields should be returned in any item results.

GetState

This method returns the current state of the subscription.

This method has the following parameters:

Name	Description
[Return Value]	Returns the current state of the subscription as TsCDaSubscriptionState object.

ModifyItems

This method modifies the state of items in the subscription.

This method has the following parameters:

Name	Description
masks	A bit mask indicating which item state parameters are being modified.
items	The new state for the items being modified. The TsCDaItem object is described in Section 5.1.4.1. Each item must have the ServerHandle specified. The TsCDaStateMask enumeration contains the bit masks used to indicate which properties of the TsCDaItem object contain valid information.
[Return Value]	The results of the modify item operation for each item. In some cases, the server will not be able to satisfy the client request for some of the item state parameters (e.g. individual item sampling rates might not be supported). The TsCDaItemResult object contains the actual item state.

ModifyState	This method changes the state of a subscription. This method has the following parameters: **Name** / **Description** masks: A bit mask that indicates which elements of the subscription state are changing. state: The new subscription state. The TsCDaSubscriptionState object is described in Section 5.1.4.7. The TsCDaStateMask enumeration contains the bit masks used to indicate which properties of the TsCDaSubscriptionState object contain valid information. [Return Value]: The actual subscription state after applying the changes. In some cases, the server will not be able to satisfy the client request (e.g the requested update rate may not be supported). The client must check the return value to determine the actual state of the subscription.
Read	**Read(TsCDaItem[] items)** This method reads the values for a set of items in the subscription. This method has the following parameters: **Name** / **Description** items: The set of items to read. The TsCDaItem object is described in Section 5.1.4.1. Each item must have its ServerHandle specified. Each item may have the ReqType and/or MaxAge specified. [Return Value]: The results of the read operation for each item as OpcItemResult Object.

Read

Read(TsCDaItem[] items, object requestHandle, TsCDaReadCompleteHandler callback, IOpcRequest request)

This method begins an asynchronous read operation for a set of items.

The .NET framework allows clients to invoke any method on any object as an asynchronous call, however, this mechanism just causes the .NET framework invoke synchronous call on the server on behalf of the client. However, the COM-DA specification allows clients to ask the server to handle the asynchronous processing instead. This can result in more efficient I/O for some OPC servers. For this reason, the .NET API makes this server-side asynchronous I/O available to clients.

This method takes the following parameters:

Name	Description
items	The set of items to read. The TsCDaItem object is described in Section 5.1.4.1. Each item must have its ServerHandle specified. Each item may have the ReqType and/or MaxAge specified.
requestHandle	A client assigned identifier for the request.
callback	The function to invoke when the request completes. The TsCDaReadCompleteHandler delegate is described below.
request	An identifier for the request (may be used to cancel the request).
[Return Value]	An array of OpcItemResult containing any errors encountered when the server validated the items.

The TsCDaReadCompleteHandler delegate has the following parameters:

Name	Description
requestHandle	A client assigned identifier for the request.
results	The value of each item as TsCDaItemValueResult array. The item results always contain the ClientHandle.

Refresh

Refresh( )

This method causes the server to send a data changed notification for all active items.

The results of this method sent to subscribers for the DataChanged event.

This method has no parameters.

Refresh

Refresh(object requestHandle, IOpcRequest request)

This method causes the server to send a data changed notification for all active items.

The results of this method sent to subscribers for the DataChanged event.

This method has the following parameters:

Name	Description
requestHandle	A client assigned identifier for the request.
request	An identifier for the request (may be used to cancel the request).

RemoveItems	This method modifies the state of items in the subscription. This method has the following parameters: **Name** — **Description** items — The identifiers (i.e. server handles) for the items being removed. The OpcItem object is described in Section 4.7.3.1. Each item must have the ServerHandle specified. [Return Value] — An array of OpcItemResult containing the results of the remove item operation for each item.
SetEnabled	This method enables or disables data change notifications from the server. This method takes the following parameters: **Name** — **Description** enabled — Whether data changed notifications should be sent.
SetResultFilters	This method sets the filters applied by the server to any item results returned to the client. This method is the same as the method described in Section 5.1.3.1. The filters specified at the subscription level override filters specified at the server level.
Write	**Write(TsCDaItemValue[] items)** This method writes the value, quality and timestamp for a set of items in the subscription. This method has the following parameters: **Name** — **Description** items — The set of item values to write. The TsCDaItemValue object is described in Section 5.1.4.2. Each item must have its ServerHandle and Value specified. Each item may have a Quality and/or a Timestamp specified. [Return Value] — The results of the write operation for each item as OpcItemResult Object.

Write	**Write(TsCDaItemValue[] items, object requestHandle, TsCDaWriteCompleteHandler callback, IOpcRequest request)** This method begins an asynchronous write operation for a set of items. This method is provided in addition to the .NET framework support for asynchronous I/O for the reasons discussed above. This method takes the following parameters:

Name	Description
items	The set of items to write. The TsCDaItemValue object is described in Section 5.1.4.2. Each item must have its ServerHandle specified. Each item may have the ReqType and/or MaxAge specified.
requestHandle	A client assigned identifier for the request.
callback	The function to invoke when the request completes. The TsCDaWriteCompleteHandler delegate is described below.
request	An identifier for the request (may be used to cancel the request).
[Return Value]	An array of OpcItemResult containing any errors encountered when the server validated the items.

The TsCDaWriteCompleteHandler delegate has the following parameters:

Name	Description
requestHandle	A client assigned identifier for the request.
results	The results of the write operation for each item as OpcItemResult array. The item results always contain the ClientHandle.

5.1.3.2.2 *Events*

The ITsDaSubscription interface has the following events:

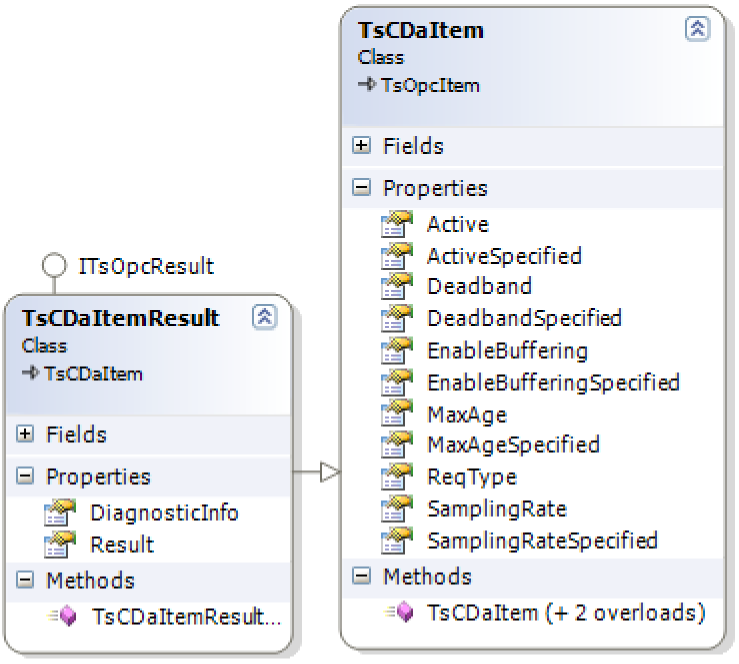
Name	Description
DataChanged	An event to receive data change updates.

5.1.4 Classes

5.1.4.1 TsCDaItem class

This class describes how an item in the server address space should be accessed.

Its properties are used to control how the server should access the item and what should be returned to the client. A T*sCDaItem* object is used in many contexts, only some of the properties will be meaningful in each context. The description of any method that has a *TsCDaItem* object as a parameter must specify which properties are relevant for that method.



5.1.4.1.1 *Properties*

The TsCDaItem class has the following properties:

Name	Description
Active	Whether the server should send data change updates.
ActiveSpecified	Whether the Active state is specified.
Deadband	The minimum percentage change required to trigger a data update for an item. The range of the Deadband is from 0.0 to 100.0 Percent. Deadband will only apply to items in the group that have a dwEUType of Analog available. If the dwEUType is Analog, then the EU Low and EU High values for the item can be used to calculate the range for the item. This range will be multiplied with the Deadband to generate an exception limit. An exception is determined as follows: Exception if (absolute value of (last cached value - current value) > (pPercentDeadband/100.0) * (EU High - EU Low)) The PercentDeadband can be set when AddGroup is called, allowing the same PercentDeadband to be used for all items within that particular group. However, with OPC DA 3.0, it is allowable to set the PercentDeadband on a per item basis. This means that each item can potentially override the PercentDeadband set for the group it resides within. If the exception limit is exceeded, then the last cached value is updated with the new value and a notification will be sent to the client's callback (if any). The pPercentDeadband is an optional behavior for the server. If the client does not specify this value on a server that does support the behavior, the default value of 0 (zero) will be assumed, and all value changes will update the CACHE. Note that the timestamp will be updated regardless of whether the cached value is updated. A server which does not support deadband should return an error (OPC_E_DEADBANDNOTSUPPORTED) if the client requests a deadband other than 0.0. The UpdateRate for a group or the sampling rate of the item, if set, determines time between when a value is checked to see if the exception limit has been exceeded. The PercentDeadband is used to keep noisy signals from updating the client unnecessarily.
DeadbandSpecified	Whether the Deadband is specified.
EnableBuffering	Whether the server should buffer multiple data changes between data updates. **Only supported for OPC DA 3.0 servers!**
EnableBufferingSpecified	Whether the Enable Buffering is specified.

	Only supported for OPC DA 3.0 servers!
MaxAge	The oldest (in milliseconds) acceptable cached value when reading an item. The server will calculate the number of milliseconds between "now" and the timestamp on the item. If the has not been updated within the last MaxAge milliseconds, the item must be obtained from the underlying device. Or if the item is not available from the cache, it will also need to be obtained from the underlying device. A max age of <=0 is equivalent to OPC_DS_DEVICE and a max age of *Int32.MaxValue* is equivalent to OPC_DS_CACHE. Without existence of a cache the server will always read from device. In this case MaxAge is not relevant. Clients should not expect that a cache exists, if they have not activated both the item and the containing group. Some servers maintain a global cache for all clients. If the needed item is in this global cache, it is expected that the server makes use of it to check the MaxAge value. Servers should not automatically create or change the caching of an item based on a Read call with MaxAge. (Note: Since this is an Int32 of milliseconds, the largest MaxAge value would be approximately is 24 days). **OPC DA 2.0:** If MaxAgeSpecified is false the OPC DA/AE/HDA Client SDK .NET will issue a read from cache. If MaxAgeSpecified is true the OPC DA/AE/HDA Client SDK .NET will issue a read depending on MaxAge; a MaxAge of <=0 issue a read from device otherwise a read from cache is issued. **OPC DA 3.0:** MaxAge values <=0 will be given to the server with value 0.
MaxAgeSpecified	Whether the Max Age is specified.
ReqType	The data type to use when returning the item value.
SamplingRate	How frequently the server should sample the item value. This overrides the update rate of the subscription as far as collection from the underlying device is concerned. The update rate associated with individual items does not effect the callback period. If the item sampling rate is less than the subscription/group update rate, the server must buffer multiple values (if supported) for the item to be included in a single callback performed at the subscription update rate. Multiple values for the same item must be in chronological order within the callback array. In other words, if the subscription has an update rate of 10 seconds and there is an item within the subscription that has a sampling rate of 2 seconds, then the callback will continue to occur no faster than every 10 seconds as defined by the subscription. In the case where an item has a different update rate than the subscription, this will indicate to the server how often this particular item should be sampled from the underlying device as well as how 'fresh' the cache will be for this particular item. If the item has a faster sampling rate than the subscription update rate and the value and/or quality change more often than the subscription update rate, then the server will buffer (if supported) each occurrence and then pass this information onto the client in the scheduled callback. The amount of data buffered is server dependent. In the case where a server does not support buffering, then the timestamp of the collected item will reflect the update rate of the item as opposed to the update rate of the subscription. A requested sampling rate of zero indicates that the client wants the item sampled at the fastest rate supported by the server. The returned revised sampling rate will indicate the actual sampling rate being used by the server. If the sampling rate is slower than the subscription update rate, then the item will only be collected from the underlying device at the sampling rate, as opposed to the subscription update rate. Because the server may need to buffer an unknown amount of data, the server is allowed to determine the maximum amount of data buffered. The server should maintain the same size buffer for each item. If the server determines that its maximum buffer capacity has been reached, then it will begin to push out the older data, keeping the newest data in the buffer for each item. In the event of a buffer overflow for a particular item, the error for this item will be OPC_S_DATAQUEUEOVERFLOW and the HRESULT will be set to S_FALSE for the callback.

	If an item has more than one value/quality/timestamp to be returned with a particular OnDataChange callback, then there will be multiple duplicate ClientHandles, depending on the size of the collection, returned with their corresponding value/quality/timestamp trio. **Only supported for OPC DA 3.0 servers!**
SamplingRateSpecified	Whether the Sampling Rate is specified. **Only supported for OPC DA 3.0 servers!**

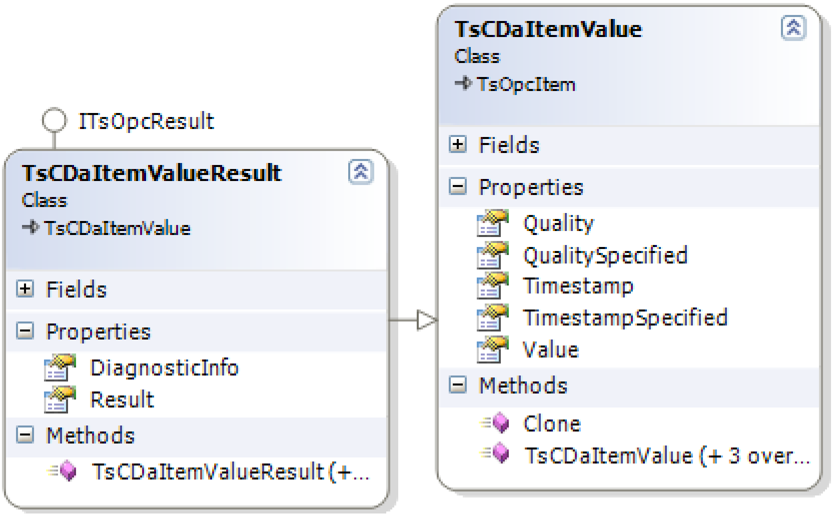
The TsCDaItemResult class extends the TsCDaItem class by adding the following properties:

Name	**Description**
DiagnosticInfo	Vendor specific diagnostic information (not the localized error text).
Result	The error id for the result of an operation on a property.

5.1.4.2 TsCDaItemValue class

This class contains the value, quality and timestamp of an item.

This class is used to return item values read from the server or to specify item value to write.



5.1.4.2.1 Properties

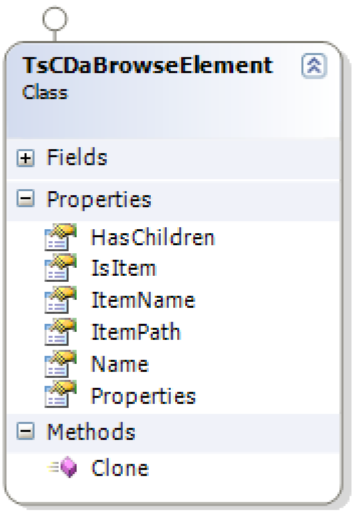
The TsCDaItemValue class has the following properties:

Name	**Description**
Quality	The quality of the item value.
QualitySpecified	Whether the quality is specified.
Timestamp	The timestamp for the item value.
TimestampSpecified	Whether the timestamp is specified.
Value	The item value.

The TsCDaItemValueResult class extends the TsCDaItemValue class by adding the following properties:

Name	**Description**
DiagnosticInfo	Vendor specific diagnostic information (not the localized error text).
Result	The result code identifier which describes the result of a operation on an item.

2. **TsCDaBrowseElement class**
This class describes an element in the server address space.



A browse element is an identifiable node in the server address space. Each browse element may contain other browse elements and may be an item (i.e. a value that can read from or written to). A browse element that is also an item may have one or more TsCDaItemProperty.

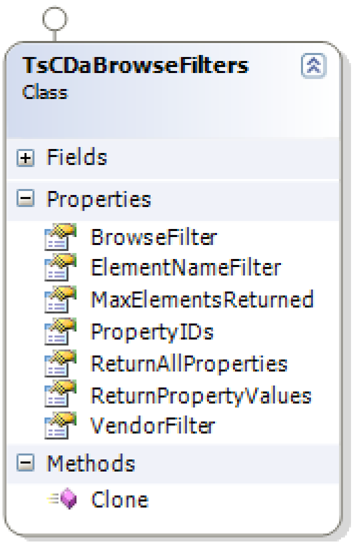
5.1.4.2.2 *Properties*

The TsCDaBrowseElement class has the following properties:

Name	Description
HasChildren	Whether the element has children.
IsItem	Whether the element refers to an item with data that can be accessed.
ItemName	The primary identifier for the element within the server namespace.
ItemPath	A secondary identifier for the element within the server namespace.
Name	A descriptive name for element that is unique within a branch.
Properties	The set of properties for the element. The TsCDaItemProperty object is described in the next section.

5.1.4.3 TsCDaBrowseFilters class

This class defines a set of filters to apply when browsing



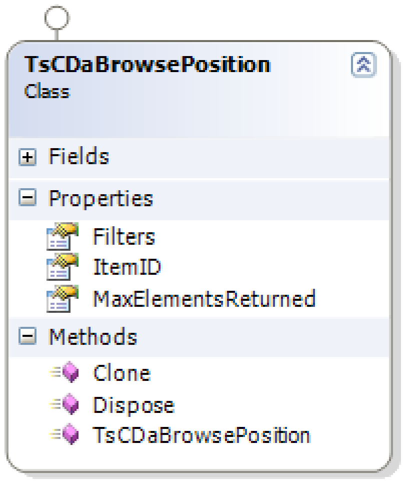
5.1.4.3.1 *Properties*

The TsCDaBrowseFilters object has the following properties:

Name	Description
MaxElementsReturned	The maximum number of elements to return. Zero means no limit.
BrowseFilter	The type of elements to return (Branch, Item or All).
ElementNameFilter	An expression used to match the name of the element.
VendorFilter	A filter which has semantics that defined by the server.
ReturnAllProperties	Whether all supported properties to return with each element.
PropertyIDs	A list of names of the properties to return with each element.
ReturnPropertyValues	Whether property values should be returned with the properties.

5.1.4.4 TsCDaBrowsePosition class

This class stores the state of a browse operation.

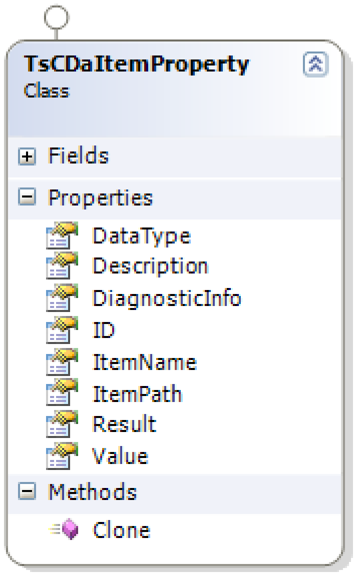


5.1.4.4.1 Properties

The TsCDaBrowsePosition object has the following properties:

Name	Description
Filters	The filters applied during the browse operation.
ItemID	The item identifier of the branch being browsed.
MaxElementsReturned	The maximum number of elements that may be returned in a single browse.

5.1.4.5 TsCDaItemProperty class



This class describes a property of an item.

Item Properties can be accessed by the TsCDaBrowseElement . This interfaces can be used by clients to browse the available properties (also referred to as attributes or parameters) associated with an ItemID and to read the current values of these properties. In some respects the functionality is similar to that provided by the SyncIO Read function. It differs from this interface in two important respects;

(a) it is intended to be much easier to use and

(b) it is not optimized for efficient access to large amounts of data.

Rather it is intended to allow an application to easily browse and read small amounts of additional information specific to a particular ItemID.

The design of this interface is based upon the assumption that many ItemIDs are associated with other ItemIDs which represent related values such as Engineering Units range or Description or perhaps Alarm Status. For example the system might be built internally of ‘records’ that represent complex objects (like PID Controllers, Timers, Counters, Analog Inputs, etc). These record items would have properties (like current value, setpoint, hi alarm limit, low alarm limit, description, etc).

As a result, these methods allows a flexible and convenient way to browse, locate and read this related information without imposing any particular design structure on the underlying system.

It also allows such information to be read without the need to create and manage subscriptions.

How ‘Properties’ relate to ItemIDs.

In most cases it is expected (but not required) that such properties can also be accessed via ItemIDs such as FIC101.HI_EU, FIC101.DESC, FIC101.ALMSTAT, etc. These related ItemIDs could be used in a subscription. These methods provide a way to easily determine if such an alternate method of efficiently.

A system like the one above (i.e. one composed internally of 'records') may also expose a hierarchical address space to OPC in the form of A100 as a 'branch' and A100.CV, A100.SP, A100.OUT, A100.DESC as 'children'. In other words, the properties of an item that happens to be a record, will generally map into lower level ITEMIDS. Another way to look at this is that things, like A100, that have properties are going to show up as 'Branch' Nodes in the OPC Browser and things that are properties are going to show up as 'children' nodes in the OPC Browser.

Note that the A100 item could in fact be embedded in a higher level "Plant.Building.Line" hierarchy however for the moment we will ignore this as it is not relevant to this discussion.

The general intent of these methods is to provide the following functions:

1. Given an ItemID of any one of a number of related properties (like A100.CV or A100.DESC or even A100), return a list of the other related properties.
2. Given a list of Item Property IDs, return a list of current data values.
3. Given a list of Item Property IDs, return a list of valid ItemIDs that can be utilized in a call to CreateSubscripton.

It should be noted that the first 8 properties (the OPC Specific Property Set 1) are 'special cases' in that they represent data that would exist within the OPC Server if this item were added to a subscription and may not represent ‘named’ properties of the 'real' tag record in the underlying system. These include, CanonicalDataType, AccessRights, EUType and EUInfo. These properties, as well as those representing the properties of the Value, Timestamp and Quality, apply to an individual and valid ItemID (one that is represented by a property ID greater than 99). Therefore, the OPC Specific Property Set 1 behaves differently in the methods on this interface. Refer to each method definition for the behavior description.

5.1.4.5.1 Properties

The TsCDaItemProperty class has the following properties:

Name	**Description**
DataType	The data type of the property.
Description	A short description of the property.
DiagnosticInfo	Vendor specific diagnostic information (not the localized error text).
ID	The property identifier.
ItemName	The primary identifier for the property if it is directly accessible as an item.
ItemPath	The secondary identifier for the property if it is directly accessible as an item.
Result	The OpcResult object with the result of an operation on an property.
Value	The value of the property.

5.1.4.6 TsDaProperty class

This class defines identifiers for well-known properties.

The server will need to assign DWORD ID codes to the properties. This allows the client to more easily manage the list of properties it wants to access. These properties are divided (somewhat arbitrarily) into 3 'sets'. The OPC 'Fixed' set contains properties that are identical to some of those returned by OPCITEMATTRIBUTES, the 'recommended' set is expected to be common to many servers, the 'vendor specific' set contains additional properties as appropriate. The assigned IDs for the first two sets are fixed. The vendor specific properties should use ID codes above 5000.

The OPC Property Sets

This is a set of property IDs that are common to many servers. Servers that provide the corresponding properties must do so using the ID codes from this list.

Note again that this interface is NOT intended to allow efficient access to large amounts of data.

The LocaleID of the server (as set by IOPCCommon::SetLocaleID) will be used by the server to localize any data items returned as strings. The item descriptions are not localized.

5.1.4.6.1 Fields

The TsDaProperty class has the following fields:

ID Set 1 - OPC Specific Properties - This includes information directly related to the OPC Server for the system.

Name	ID	Data Type	Description
DATATYPE	1	VT_I2	Item Canonical DataType
VALUE	2	<varies>	Item Value Note the type of value returned is as indicated by the "Item Canonical DataType" above and depends on the item. This will behave like a read from DEVICE.
QUALITY	3	VT_I2	Item Quality (OPCQUALITY stored in an I2). This will behave like a read from DEVICE.)
TIMESTAMP	4	VT_DATE	Item Timestamp (will be converted from FILETIME). This will behave like a read from DEVICE.)
ACCESSRIGHTS	5	VT_I4	Item Access Rights (OPCACCESSRIGHTS stored in an I4)
SCANRATE	6	VT_R4	Server Scan rate In Milliseconds. This represents the fastest rate at which the server could obtain data from the underlying data source. The nature of this source is not defined but is typically a DCS system, a SCADA system, a PLC via a COMM port or network, a Device Network, etc. This value generally represents the 'best case' fastest RequestedUpdateRate which could be used if this item were added to an OPCGroup. The accuracy of this value (the ability of the server to attain 'best case' performance) can be greatly affected by system load and other factors.
EUTYPE	7	VT_I4	Item EU Type (OPCEUTYPE stored in an I4) See IOPCItemAttributes for additional information)

EUINFO	8	VT_BSTR | VT_ARRAY	Item EU Info If item 7 "Item EU Type" is "Enumerated" - EUInfo will contain a SAFEARRAY of strings (VT_ARRAY | VT_BSTR) which contains a list of strings (Example: "OPEN", "CLOSE", "IN TRANSIT",
	9-99		Reserved for future OPC use

ID Set 2 - Recommended Properties - This is additional information which is commonly associated with ITEMs. This includes additional ranges of values that are reserved for use by other future OPC specifications. For information about the newest field ID assignments, consult the other OPC Foundation specifications.

The position of the OPC Foundation is that if you have properties associated with an item which seem to fit the descriptions below then it is recommended that you use these specific descriptions and ID codes to expose those properties via this interface.

A server can provide any subset of these values (or none of them).

Name	**ID**	**Data Type**	**Description**
			Properties related to the Item Value.
ENGINEERINGUINTS	100	VT_BSTR	EU Units e.g. "DEGC" or "GALLONS"
DESCRIPTION	101	VT_BSTR	Item Description e.g. "Evaporator 6 Coolant Temp"
HIGHEU	102	VT_R8	High EU Present only for 'analog' data. This represents the highest value likely to be obtained in normal operation and is intended for such use as automatically scaling a bargraph display. e.g. 1400.0
LOWEU	103	VT_R8	Low EU Present only for 'analog' data. This represents the lowest value likely to be obtained in normal operation and is intended for such use as automatically scaling a bargraph display. e.g. -200.0
HIGHIR	104	VT_R8	High Instrument Range Present only for 'analog' data. This represents the highest value that can be returned by the instrument. e.g. 9999.9
LOWIR	105	VT_R8	Low Instrument Range Present only for 'analog' data. This represents the lowest value that can be returned by the instrument. e.g. -9999.9
CLOSELABEL	106	VT_BSTR	Contact Close Label Present only for 'discrete' data. This represents a string to be associated with this contact when it is in the closed (non-zero) state e.g. "RUN", "CLOSE", "ENABLE", "SAFE" ,etc.
OPENLABEL	107	VT_BSTR	Contact Open Label Present only for 'discrete' data. This represents a string to be associated with this contact when it is in the open (zero) state e.g. "STOP", "OPEN", "DISABLE", "UNSAFE" ,etc.
TIMEZONE	108	VT_I4	Item Timezone The difference in minutes between the items UTC Timestamp and the local time in which the item value was obtained. See the OPCGroup TimeBias property. Also see the WIN32 TIME_ZONE_INFORMATION structure.
	109 - 199		Reserved for future OPC use. Additional IDs may be added without revising the interface ID.

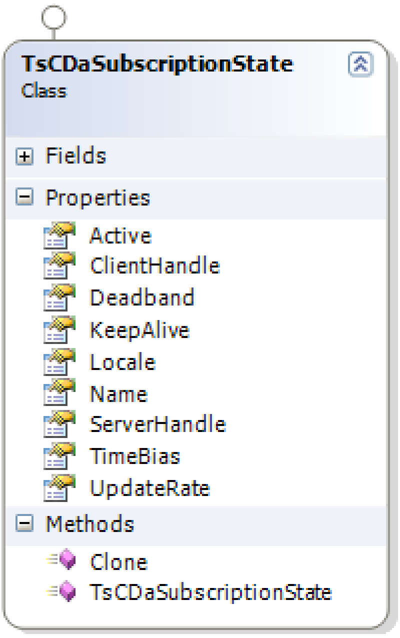
Name	ID	Data Type	Description
			Properties Related to Alarm and Condition Values (preliminary)... **IDs 300 to 399 are reserved for use by OPC Alarms and Events.** **See the OPC Alarm and Events specification for additional information.**
CONDITION_STATUS	300	VT_BSTR	Condition Status The current alarm or condition status associated with the Item e.g. "NORMAL", "ACTIVE", "HI ALARM", etc
ALARM_QUICK_HELP	301	VT_BSTR	Alarm Quick Help A short text string providing a brief set of instructions for the operator to follow when this alarm occurs.
ALARM_AREA_LIST	302	VT_BSTR | VT_ARRAY	Alarm Area List An array of stings indicating the plant or alarm areas which include this ItemID.
PRIMARY_ALARM_AREA	303	VT_BSTR	Primary Alarm Area A string indicating the primary plant or alarm area including this ItemID
CONDITION_LOGIC	304	VT_BSTR	Condition Logic An arbitrary string describing the test being performed. e.g. "High Limit Exceeded" or "TAG.PV >= TAG.HILIM"
LIMIT_EXCEEDED	305	VT_BSTR	Limit Exceeded For multistate alarms, the condition exceeded e.g. HIHI, HI, LO, LOLO
DEADBAND	306	VT_R8	Deadband
HIHI_LIMIT	307	VT_R8	HiHi Limit
HI_LIMIT	308	VT_R8	Hi Limit
LO_LIMIT	309	VT_R8	Lo Limit
LOLO_LIMIT	310	VT_R8	LoLo Limit
RATE_CHANGE_LIMIT	311	VT_R8	Rate of Change Limit
DEVIATION_LIMIT	312	VT_R8	Deviation Limit
SOUNDFILE	313	VT_BSTR	Sound File e.g. C:\MEDIA\FIC101.WAV, or .MID
	314 - 399		Reserved for future OPC Alarms and Events use. Additional IDs may be added without revising the interface ID.
	400 - 4999		Reserved for future OPC use. Additional IDs may be added without revising the interface ID.

ID Set 3 - Vendor specific Properties

Name	ID	Data Type	Description
	5000...	VT_xxx	Vendor Specific Properties. ID codes for these properties must have values of 5000 or greater. They do not need to be sequential. The datatypes must be compatable with the VARIANT.

5.1.4.7 TsCDaSubscriptionState class

This class describes the state of a subscription.



5.1.4.7.1 *Properties*

The TsCDaSubscriptionState object has the following properties:

Name	Description
Active	Whether the subscription is scanning for updates to send to the client.
ClientHandle	A unique identifier for the subscription assigned by the client.
Deadband	The minimum percentage change required to trigger a data update for an item.
KeepAlive	The maximum period between updates sent to the client.
Locale	The locale used for any error messages or results returned to the client.
Name	A unique name for the subscription controlled by the client.
ServerHandle	A unique identifier for the subscription assigned by the server.
TimeBias	The Time Zone Bias of the subscription (in minutes).
UpdateRate	The rate at which the server checks of updates to send to the client.

Active

Subscriptions and Items within Subscriptions have an Active Flag. The active state of the subscription is maintained separately from the active state of the items. Changing the state of the subscription does not change the state of the items.

For the most part the Active flag is treated as 'abstract' within this specification. The state of these flags affects the described behavior of various interfaces in a well defined way. The implementation details of these capabilities are not dictated by this specification.

In practice it is expected that most servers will make use of this flag to optimize their use of communications and CPU resources. Items and Subscriptions which are not active do not need to be maintained in the CACHE.

It is also expected that clients will simply set and clear active flags of subscriptions and items as a more efficient alternative to adding and removing entire subscriptions and items. For example if an operator display is minimized, its items might be set to inactive.

Refer to the Data Acquisition and Active State Behavior summary in the OPC Data Access Specification for a quick overview of the behavior of a client and server with respect to the active state of a subscriptions and items.

OnDataChange within the client's address space can be called whenever any active data item in a active subscription changes, where "change" is defined as a change in value (from the last value sent to this client), or a change in the Quality of the value. The server can return values and quality flags for those items within the subscription that changed (this will be discussed more in the OPC Data Access specification).

ClientHandle

This handle will be returned in any callback. This allows the client to identify the subscription to which the data belongs.

It is expected that a client will assign unique value to the client handle if it intends to use any of the asynchronous functions of the OPC interfaces, such as IOPCAsyncIO2, and IConnectionPoint/IOPCDataCallback interfaces.

Deadband

The range of the Deadband is from 0.0 to 100.0 Percent. Deadband will only apply to items in the subscription that have a dwEUType of Analog available. If the dwEUType is Analog, then the EU Low and EU High values for the item can be used to calculate the range for the item. This range will be multiplied with the Deadband to generate an exception limit. An exception is determined as follows:

Exception if (absolute value of (last cached value - current value) > (pPercentDeadband/100.0) * (EU High - EU Low))

The PercentDeadband can be set when AddGroup is called, allowing the same PercentDeadband to be used for all items within that particular subscription. However, with OPC DA 3.0, it is allowable to set the PercentDeadband on a per item basis. This means that each item can potentially override the PercentDeadband set for the subscription it resides within.

If the exception limit is exceeded, then the last cached value is updated with the new value and a notification will be sent to the client's callback (if any). The pPercentDeadband is an optional behavior for the server. If the client does not specify this value on a server that does support the behavior, the default value of 0 (zero) will be assumed, and all value changes will update the CACHE. Note that the timestamp will be updated regardless of whether the cached value is updated. A server which does not support deadband should return an error (OPC_E_DEADBANDNOTSUPPORTED) if the client requests a deadband other than 0.0.

The UpdateRate for a subscription or the sampling rate of the item, if set, determines time between when a value is checked to see if the exception limit has been exceeded. The PercentDeadband is used to keep noisy signals from updating the client unnecessarily.

UpdateRate

The client can specify an 'update rate' for each subscription. This determines the time between when the exception limit is checked. If the exception limit is exceeded, the CACHE is updated. The server should make a 'best effort' to keep the data fresh. This also affects the maximum rate at which notifications will be sent to the client's callback. The server should never send data to a client at a rate faster than the client requests.

IMPORTANT:

Note that this is NOT necessarily related to the server's underlying processing rate. For example if a device is performing PID control at 0.05 second rate the an MMI requests updates at a 5 second rate via OPC, the device would of course continue to control at a 0.05 second rate.

In addition, the server implementation would also be allowed to update the cached data available to sync or async read at a higher rate than 5 seconds if it wished to do so. All the update rate indicates is that (a) callbacks should happen no faster than this and (b) the cache should be updated at at least this rate.

The update rate is a 'request' from the client. The server should respond with an update rate that is as close as possible to that requested.

Optionally, each individual item contained within a subscription may have a different sampling rate. The sampling rate associated with individual items does not effect the callback period. In other words, if the subscription has an update rate of 10 seconds and there is an item within the subscription that has a sampling rate of 2 seconds, then the callback will continue to occur no faster than every 10 seconds as defined by the subscription. In the case where an item has a different sampling rate than the update rate of the subscription, this will indicate to the server how often this particular item should be read from the underlying device as well as how 'fresh' the cache will be for this particular item.

If the item has a faster sampling rate than the subscription and the value and/or quality change more often than the subscription update rate, then the server will buffer each occurrence and then pass this information onto the client in the scheduled callback. The amount of data buffered is server dependent. See IOPCItemSamplingMgt in the OPC Data Access Specification for more detail.

T

Time Zone (TimeBias)

In some cases the data may have been collected by a device operating in a time zone other than that of the client. Then it will be useful to know what the time of the device was at the time the data was collected (e.g. to determine what 'shift' was on duty at the time).

This time zone information may rarely be used and the device providing the data may not know its local time zone, therefore it was not prudent to add this overhead to all data transactions. Instead, the subscription provides a place to store a time zone that can be set and read by the client. The default value for this is the time zone of the host computer. The OPC Server will not make use of this value. It is there only for the convenience of the client.

The purpose of the TimeBias is to indicate the time zone in which the data was collected (which may occasionally be different from the time zone in which either the client or server is running). The default TimeBias for the subscription (if a NULL pointer is passed to AddGroup) will be that of the system in which the subscription is created (i.e. the server). This bias behaves like the Bias field in the Win32 TIME_ZONE_INFORMATION structure which is to say it does NOT account for daylight savings time (DST). The TimeBias is never changed 'behind the scenes' by the server. It is set ONLY when the subscription is created or when SetState is called. In general a Client computes the data's 'local' time by TimeStamp + TimeBias + DSTBias (if any). There is an implicit assumption in this design that the DST characteristics at the data site are the same as at the client site. If this is not the case, the client will need to use some other means to compute the data's local time.

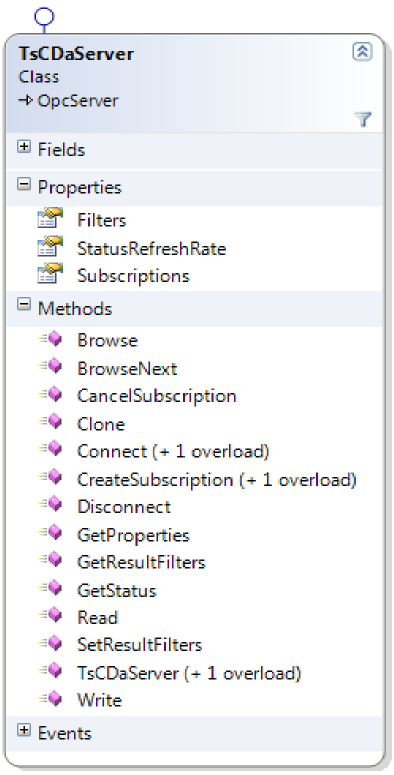
5.2 Client API

5.2.1 Classes

5.2.1.1 TsCDaServer class

This class is a base class for an in-process object used to access OPC Data Access servers and is an in-process wrapper for a remote server (i.e. a server that implements the IOpcServer and ITsDaServer interface). This class provides a mechanism to cache properties of the remote server locally for fast access and supports serialization (which simplifies the task of saving client configuration information).

This object contains references to unmanaged resources (e.g. COM servers), as a result, this object must be explicitly released by calling the *Dispose* method. A call to the *Dispose* method is automatically done by calling the *Disconnect* method.



5.2.1.1.1 *Properties*

The TsCDaServer class has the following properties:

Name	Description
Filters	The current result filters applied by the server.
StatusRefreshRate	The update rate in ms for the Status Update Thread.
Subscription	Returns an array of all subscriptions for the server.

5.2.1.1.2 *Methods*

The TsCDaServer class has the following methods:

Browse	This method fetches the children of a branch that meet the filter criteria. This method has the following parameters:	
	Name	**Description**
	itemID	The identifier of branch which is the target of the search. The ClientHandle and ServerHandle have no meaning in this context. Passing a null value searches for elements with no parent (e.g. the top of tree).
	filters	The filters to use to limit the set of child elements returned. The TsCDaBrowseFilters object is described in section 5.1.4.3.
	position	An object used to continue a browse operation A browse operation may not complete if the number of elements exceeds the value of the MaxElementsReturned filter. The client may continue the browse by calling *BrowseNext*, otherwise the client must call *Dispose* on the TsCDaBrowsePosition object to ensure that all resources allocated for the browse are released. A server will typically create sub-classes of the TsCDaBrowsePosition object that contain information used to optimize *BrowseNext* operations. This object has no properties that are visible to clients.
	[Return Value]	The set of elements found.
BrowseNext	This method continues a browse operation with previously specified search criteria. This method has the following parameters:	
	Name	**Description**
	position	An object containing the browse operation state information. This object must be returned from a call to Browse. If the position is invalid for any reason this method throws a OpcResultException exception. If there are no more elements to fetch this method will set the TsCDaBrowsePosition to null. Otherwise, this method will return a new TsCDaBrowsePosition object.
	[Return Value]	The set of elements found.
CancelSubscription	This method cancels a subscription and releases all resources allocated for it. Clients must always explicitly cancel all subscriptions that it creates. This method has the following parameters:	
	Name	**Description**
	subscription	The subscription to cancel.

Clone	Returns an unconnected copy of the server with the same URL. This method calls the base class *Clone* method and then automatically creates all necessary subscriptions. Note that an unconnected server object will only contain subscriptions if they were part of the object when it was serialized. This allows the client to restore the remote server state by calling this method after deserializing the object.
Connect	**Connect(string url)** Connects the object to an OPC DA Server. This method has the following parameters:

Name	Description
url	Name of the OPC DA server. The usual form is http:://xxx/yyy, e.g. opcda://localhost//Technosoftware.DaSample.

Connect	**Connect(string url, OpcConnectData connectData)** Establishes a physical connection to the remote server. This method has the following parameters:

Name	Description
url	The network address for the remote server. It replaces the default URL for the server if the method succeeds.
connectData	Any protocol configuration or user authentication information.

CreateSubscription	This method creates a new subscription. A subscription allows a client to receive asynchronous notifications from the server whenever an item value changes. All subscriptions that a client creates must be destroyed with the *CancelSubscription* method. The SOAP/XML protocol introduces some complexity with regards to subscriptions because no other method requires that the server maintain state information across method calls. The SOAP/XML stub resolves this issue by managing the references to the ITsDaServer and ITsCDaSubscription objects on behalf of the remote client. This method has the following parameters:

Name	Description
state	The initial state of the subscription. The TsCDaSubscriptionState object is described below.
[Return Value]	The new subscription object.

Disconnect	Disconnects from the server and releases all network resources. This method removes all existing subscriptions before it calls the base class *Disconnect* method. Also Dispose() is called.

GetProperties	This method returns the item properties for a set of items. This method has the following parameters:

Name	Description
itemIDs	A list of item identifiers.
propertyIDs	A list of properties to fetch for each item. If this parameter is null then all available properties are returned.
returnValues	Whether the property values should be returned with the properties.
[Return Value]	A list of properties for each item. The TsCDaItemPropertyCollection object is described below.

The TsCDaItemPropertyCollection object extends *ArrayList* and has the following properties/methods:

Name	Description
ItemName	The primary identifier for the item within the server namespace.
ItemPath	The secondary identifier for the item within the server namespace.
Result	A result code that indicates any item-level errors.
operator[]	Returns the TsCDaItemProperty object at the specified index.

GetResultFilters	This method returns the filters applied by the server to any item results returned to the client. This method has the following parameters:

Name	Description
[Return Value]	A bit mask indicating which fields should be returned in any item results.

The set of masks is has the following values:

Name	Value	Description
ItemName	0x01	Include the ItemName in the OpcItem if bit is set.
ItemPath	0x02	Include the ItemPath in the OpcItem if bit is set.
ClientHandle	0x04	Include the ClientHandle in the OpcItem if bit is set.
ItemTime	0x08	Include the Timestamp in the *ItemValue* if bit is set.
ErrorText	0x10	Include verbose, localized error text with result if bit is set.
DiagnosticInfo	0x20	Include additional diagnostic information with result if bit is set.
Minimal	0x09	Include the ItemName and Timestamp if bit is set.
All	0xFF	Include all information in the results if bit is set.

Note that the ClientHandle property of and OpcItem has no meaning when used at the server level.

The filters only affect results returned from the Read and Write methods. They are also used as the default for new Subscriptions.

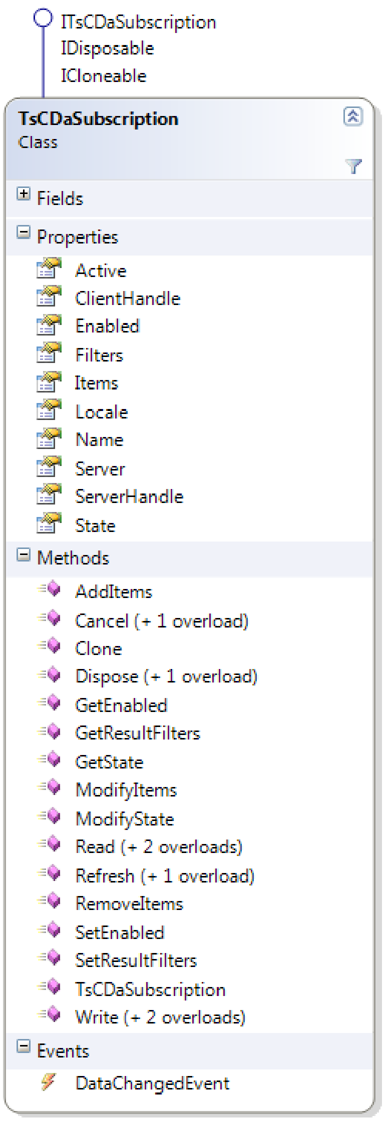
GetServerStatus	This method returns the current status of the server. This method has the following parameters: **Name** | **Description** [Return Value] | The current server state as OpcServerStatus object. See chapter 4.8.2.2 for more details.
Read	The method reads the current values for a set of items. This method has the following parameters: **Name** | **Description** items | The set of items to read. Each item must have an ItemName Each item may have an ItemPath, a ReqType or MaxAge. [Return Value] | The results of the read operation for each item. The number of item values returned must equal the number of items passed to the method. The client uses the index in the arrays to match a item value with the item. The server indicates errors on individual items by returning the appropriate result code as part of the item value. It is up to the server to decide whether a cache read is appropriate for a given item, as a result, a server may choose to read directly from the device even if the client requests a MaxAge of *Int32.MaxValue*.
SetResultFilters	This method sets the filters applied by the server to any item results returned to the client. This method has the following parameters: **Name** | **Description** filters | A bit mask indicating which fields should be returned in any item results.
Write	This method writes the value, quality and timestamp for a set of items. This method has the following parameters: **Name** | **Description** itemValues | The set of item values to write. Each item must have an ItemName and a Value. Each item may have an ItemPath, a Quality and a Timestamp. [Return Value] | The results of the write operation for each item. The number of item results returned must equal the number of item values passed to the method. The client uses the index in the arrays to match a item result with the item value. The server indicates errors on individual items by returning the appropriate result code as part of the item value. The server may support writing to the quality and/or timestamp. In these cases, the server does not write the value and returns 'E_NO_WRITEQT' for the item.

5.2.1.2 TsCDaSubscription Class

This class is an in-process object used to access subscriptions on OPC Data Access servers.

This class may be used to access subscriptions (also called groups) on OPC DA servers. Clients may create sub-classes which add additional functionality.

This object contains references to unmanaged resources (e.g. COM servers), as a result, this object must be explicitly released by calling the Dispose method.



5.2.1.2.1 Properties

The TsCDaSubscription class has the following properties:

Name	Description
Active	Whether the subscription is active.
ClientHandle	The handle assigned to the group by the client.
Enabled	Whether data callbacks are enabled.
Filters	The current result filters applied by the subscription.
Items	The items belonging to the subscription.
Locale	The current locale used by the subscription.
Name	The name assigned to the subscription by the client.
Server	The server that the subscription is attached to.
ServerHandle	The handle assigned to the subscription by the server.
State	Returns a copy of the current subscription state.

The properties of this object are all read only. They are intended to provide fast access to these values without making any network calls. The client must use the methods of the ITsCDaSubscription interface to actually change any of these properties.

5.2.1.2.2 *Methods*

The TsCDaSubscription interface has the following methods:

Name	Description
AddItems	This method adds items to the subscription. This method has the following parameters:

Name	Description
items	The set of items to add to the subscription. The *TsCDaItem* object is described in Section 5.1.4.1. Each item must have an ItemName Each item may have the ItemPath, ClientHandle, ReqType, Active, SamplingRate or EnableBuffering properties specified.
[Return Value]	The results of the add item operation for each item. In some cases, the server will not be able to satisfy the client request for some of the item state parameters (e.g. individual item sampling rates might not be supported). The *ItemResult* object contains the actual item. The *TtemResult* object also contains a ServerHandle which the client must use to reference the item in other methods on the subscription.

The index in the array is used to associate a result with a specific item.

Name	Description
Cancel	This method cancels an asynchronous read or writes operation. This method takes the following parameters:

Name	Description
request	The object returned from the Read or Write request.
callback	The function to invoke when the cancel completes.

Name	Description
Clone	Returns an unconnected copy of the subscription with the same items.
GetEnabled	This method checks whether data change notifications from the server are enabled. This method takes the following parameters:

Name	Description
[Return Value]	Whether data changed notifications should be sent.

GetResultFilters	This method returns the filters applied by the server to any item results returned to the client. This method is the same as the method described in Section 5.1.3.1. The filters specified at the subscription level override filters specified at the server level. This method takes the following parameters: **Name** / **Description** [Return Value] / A bit mask indicating which fields should be returned in any item results.
GetState	This method returns the current state of the subscription. This method has the following parameters: **Name** / **Description** [Return Value] / Returns the current state of the subscription as TsCDaSubscriptionState object.
ModifyItems	This method modifies the state of items in the subscription. This method has the following parameters: **Name** / **Description** masks / A bit mask indicating which item state parameters are being modified. items / The new state for the items being modified. The TsCDaItem object is described in Section 5.1.4.1. Each item must have the ServerHandle specified. The TsCDaStateMask enumeration contains the bit masks used to indicate which properties of the TsCDaItem object contain valid information. [Return Value] / The results of the modify item operation for each item. In some cases, the server will not be able to satisfy the client request for some of the item state parameters (e.g. individual item sampling rates might not be supported). The TsCDaItemResult object contains the actual item state.

ModifyState	This method changes the state of a subscription. This method has the following parameters:

Name	**Description**
masks	A bit mask that indicates which elements of the subscription state are changing.
state	The new subscription state. The TsCDaSubscriptionState object is described in Section 5.1.4.7. The TsCDaStateMask enumeration contains the bit masks used to indicate which properties of the TsCDaSubscriptionState object contain valid information.
[Return Value]	The actual subscription state after applying the changes. In some cases, the server will not be able to satisfy the client request (e.g the requested update rate may not be supported). The client must check the return value to determine the actual state of the subscription.

Read	**Read(TsCDaItem[] items)** This method reads the values for a set of items in the subscription. This method has the following parameters:

Name	**Description**
items	The set of items to read. The TsCDaItem object is described in Section 5.1.4.1. Each item must have its ServerHandle specified. Each item may have the ReqType and/or MaxAge specified.
[Return Value]	The results of the read operation for each item as OpcItemResult Object.

Read	**Read(TsCDaItem[] items, object requestHandle, TsCDaReadCompleteEventHandler callback, IOpcRequest request)** This method begins an asynchronous read operation for a set of items. The .NET framework allows clients to invoke any method on any object as an asynchronous call, however, this mechanism just causes the .NET framework invoke synchronous call on the server on behalf of the client. However, the COM-DA specification allows clients to ask the server to handle the asynchronous processing instead. This can result in more efficient I/O for some OPC servers. For this reason, the .NET API makes this server-side asynchronous I/O available to clients. This method takes the following parameters:

Name	Description
items	The set of items to read. The TsCDaItem object is described in Section 5.1.4.1. Each item must have its ServerHandle specified. Each item may have the ReqType and/or MaxAge specified.
requestHandle	A client assigned identifier for the request.
callback	The function to invoke when the request completes. The TsCDaReadCompleteEventHandler delegate is described below.
request	An identifier for the request (may be used to cancel the request).
[Return Value]	An array of OpcItemResult containing any errors encountered when the server validated the items.

The TsCDaReadCompleteEventHandler delegate has the following parameters:

Name	Description
requestHandle	A client assigned identifier for the request.
results	The value of each item as TsCDaItemValueResult array. The item results always contain the ClientHandle.

Refresh	**Refresh( )** This method causes the server to send a data changed notification for all active items. The results of this method sent to subscribers for the DataChanged event. This method has no parameters.
Refresh	**Refresh(object requestHandle, IOpcRequest request)** This method causes the server to send a data changed notification for all active items. The results of this method sent to subscribers for the DataChanged event. This method has the following parameters:

Name	Description
requestHandle	A client assigned identifier for the request.
request	An identifier for the request (may be used to cancel the request).

RemoveItems	This method modifies the state of items in the subscription.

	This method has the following parameters:

Name	Description
items	The identifiers (i.e. server handles) for the items being removed. The OpcItem object is described in Section 4.7.3.1. Each item must have the ServerHandle specified.
[Return Value]	An array of OpcItemResult containing the results of the remove item operation for each item.

SetEnabled	This method enables or disables data change notifications from the server. This method takes the following parameters:

Name	Description
enabled	Whether data changed notifications should be sent.

SetResultFilters	This method sets the filters applied by the server to any item results returned to the client. This method is the same as the method described in Section 5.1.3.1. The filters specified at the subscription level override filters specified at the server level.
Write	**Write(TsCDaItemValue[] items)** This method writes the value, quality and timestamp for a set of items in the subscription. This method has the following parameters:

Name	Description
items	The set of item values to write. The TsCDaItemValue object is described in Section 5.1.4.2. Each item must have its ServerHandle and Value specified. Each item may have a Quality and/or a Timestamp specified.
[Return Value]	The results of the write operation for each item as OpcItemResult Object.

Write	**Write(TsCDaItemValue[] items, object requestHandle, TsCDaWriteCompleteEventHandler callback, IOpcRequest request)** This method begins an asynchronous write operation for a set of items. This method is provided in addition to the .NET framework support for asynchronous I/O for the reasons discussed above. This method takes the following parameters: **Name** | **Description** items | The set of items to write. The TsCDaItemValue object is described in Section 5.1.4.2. Each item must have its ServerHandle specified. Each item may have the ReqType and/or MaxAge specified. requestHandle | A client assigned identifier for the request. callback | The function to invoke when the request completes. The TsCDaWriteCompleteEventHandler delegate is described below. request | An identifier for the request (may be used to cancel the request). [Return Value] | An array of OpcItemResult containing any errors encountered when the server validated the items. The TsCDaWriteCompleteEventHandler delegate has the following parameters: **Name** | **Description** requestHandle | A client assigned identifier for the request. results | The results of the write operation for each item as OpcItemResult array. The item results always contain the ClientHandle.

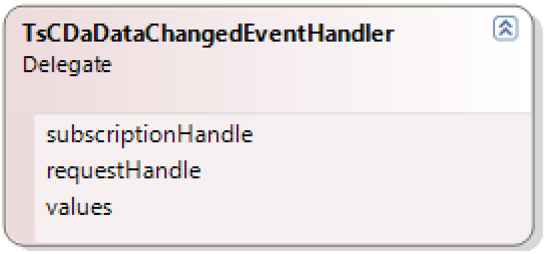
5.2.1.2.3 *Events*

The TsCDaSubscription class has the following events:

Name	Description
DataChangedEvent	An event to receive data change updates.

5.2.2 Delegates

5.2.2.1 TsCDaDataChangedEventHandler delegate



This delegate is provided by the client to handle notifications from the subscription (OPC Group) for exception based data changes and Refreshes.

This section will discuss the reasons why the client may receive callbacks.

Callbacks can occur for the following reasons;

+ One or more 'data change' events. These will happen for active items within an active group where the value or quality of the item has changed. They will happen no faster than the TsCDaSubscriptionState.UpdateRate of the subscription. TsCDaSubscriptionState.DeadBand is used to determine what items have changed. The requestHandle will be 0 in this case. In general, additional updates are not sent unless there is a change in value or quality.
+ Refresh Request made through the TsCDaSubscription interface. These will happen for all active items in an active group. They will happen as soon as possible after the refresh request is made. The handle list will contain the handles for all of the active items in the group. The requestHandle will be non-zero if Refresh(requestHandle, request) was called and will be 0 if Refresh() was called.

The values array can return additional information in the case where the server is having problems obtaining data for an Item. These vendor specific errors could contain helpful information about communications errors or device status. E_FAIL, while allowed, is generally not a very helpful error to return.

See TsCDaItem.SamplingRate for more details on server behavior for buffered values.

Note: although it is not recommended, the client could change the active status of the subscription (group) or items while an asynchronous call is outstanding. The server should be able to deal with this in a reasonable fashion (i.e. not crash) although the exact behavior is undefined.

5.2.2.1.1 Properties

The TsCDaDataChangedEventHandler delegate has the following properties:

Name	Description
subscriptionHandle	A unique identifier for the subscription assigned by the client. If the parameter TsCDaSubscriptionState.ClientHandle is not defined this parameter is empty.
requestHandle	An identifier for the request assigned by the caller. This parameter is empty if the corresponding parameter in the calls Read(), Write() or Refresh() is not defined. Can be used to Cancel an outstanding operation.
values	The set of changed values Each value will always have the item's ClientHandle field specified.

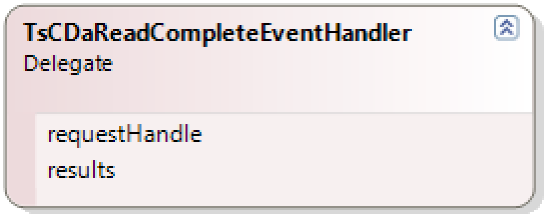
Samples
An example of a TsCDaDataChangedEventHandler delegate:

```
public void OnDataChangeEvent(object subscriptionHandle, object requestHandle,
                              TsCDaItemValueResult[] values)
{
        if (requestHandle != null)
        {
                Console.Write("DataChange() for requestHandle :");
                Console.WriteLine(requestHandle.GetHashCode().ToString());
        }
        else
        {
                Console.WriteLine("DataChange():");
        }
        for (int i = 0; i < values.GetLength(0); i++)
        {
                Console.Write("Client Handle : ");
                Console.WriteLine(values[i].ClientHandle);
                if (values[i].Result.IsSuccess())
                {
                        Console.Write("Value         : ");
                        Console.WriteLine(values[i].Value);
                }
                Console.Write("Time Stamp    : ");
                Console.WriteLine(values[i].Timestamp.ToString());
                Console.Write("Quality       : ");
                Console.WriteLine(values[i].Quality);
                Console.Write("Result        : ");
                Console.WriteLine(values[i].Result.Description());
        }
}
```

Installation of the TsCDaDataChangedEventHandler delegate:

```
group.DataChangedEvent += new TsCDaDataChangedEventHandler(OnDataChangeEvent);
```

5.2.2.2 TsCDaReadCompleteEventHandler delegate



This method is provided by the client to handle notifications from the OPC Group on completion of asynchronous reads.

For any S_xxx result code the client should assume the corresponding Value, Quality and Timestamp are well defined although the Quality may be UNCERTAIN or BAD. It is recommended (but not required) that server vendors provide additional information here regarding UNCERTAIN or BAD items.

For any FAILED result code the client should assume the corresponding Value, Quality and Timestamp are undefined. In fact the Server must set the corresponding Value VARIANT to VT_EMPTY so that it can be marshaled properly.

Items for which an error (E_xxx) was returned in the initial asynchronous read request will NOT be returned here. I.e. the returned list may be ‘sparse’. Also the order of the returned list is not specified (it may not match the order of the list passed to read).

This Callback occurs only after an asynchronous Read made with the call

TsCDaSubscription.Read(TsCDaItem[] items, object requestHandle,
TsCDaReadCompleteEventHandler callback, IOpcRequest request)

The results array can return additional information in the case where the server is having problems obtaining data for an Item. These vendor specific errors could contain helpful information about communications errors or device status. E_FAIL, while allowed, is generally not a very helpful error to return.

5.2.2.2.1 Properties

The TsCDaReadCompleteEventHandler delegate has the following properties:

Name	Description
requestHandle	An identifier for the request assigned by the caller. This parameter is empty if the corresponding parameter in the calls Read(), Write() or Refresh() is not defined. Can be used to Cancel an outstanding operation.
Results	The results of the last asynchronous read operation.

Sample

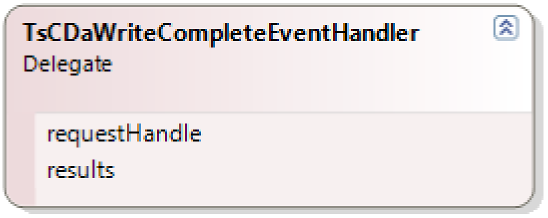
An example of a TsCDaReadCompleteEventHandler delegate:

```
public void OnReadCompleteEvent(object clientHandle, TsCDaItemValueResult[] results)
{
        Console.WriteLine("Read completed");
        foreach (TsCDaItemValueResult readResult in results)
        {
                Console.Write("Item Name     : ");
                Console.WriteLine(readResult.ItemName);
                Console.Write("Value         : ");
                Console.WriteLine(readResult.Value);
                Console.Write("Time Stamp    : ");
                Console.WriteLine(readResult.Timestamp.ToString());
                Console.Write("Quality       : ");
                Console.WriteLine(readResult.Quality);
        }
        Console.WriteLine();
}
```

Use of the TsCDaReadCompleteEventHandler delegate:

```
res = opcGroup.Read(addedItems,100,
       new TsCDaReadCompleteEventHandler(OnReadCompleteEvent),out request);
```

5.2.2.3 TsCDaWriteCompleteEventHandler delegate



This method is provided by the client to handle notifications from the OPC Group on completion of asynchronous Writes.

Items for which an error (E_xxx) was returned in the initial asynchronous Write request will NOT be returned here. I.e. the returned list may be 'sparse'. Also the order of the returned list is not specified (it may not match the order of the list passed to write).

This Callback occurs only after an asynchronous Write made with the call

TsCDaSubscription.Write(TsCDaItemValue[] items, object requestHandle, TsCDaWriteCompleteHandler callback, IOpcRequest request))

The 'errors' array can return additional information in the case where the server is having problems accessing data for an Item. These vendor specific errors could contain helpful information about communications errors or device status. E_FAIL, while allowed, is generally not a very helpful error to return.

5.2.2.3.1 *Properties*

The TsCDaWriteCompleteEventHandler delegate has the following properties:

Name	Description
requestHandle	An identifier for the request assigned by the caller. This parameter is empty if the corresponding parameter in the calls Read(), Write() or Refresh() is not defined. Can be used to Cancel an outstanding operation.
results	The results of the last asynchronous write operation.

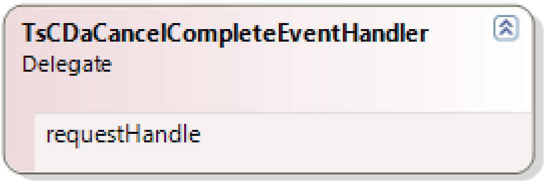
Sample
An example of a TsCDaWriteCompleteEventHandler delegate:

```
public void OnWriteCompleteEvent(object clientHandle, OpcItemResult[] results)
{
        foreach (OpcItemResult res in results)
        {
        }
}
```

Use of the TsCDaWriteCompleteEventHandler delegate:

```
res = opcGroup.Write(writeValues, 321,
        new TsCDaWriteCompleteEventHandler(OnWriteCompleteEvent), out request);
```

5.2.2.4 TsCDaCancelCompleteEventHandler delegate



This method is provided by the client to handle notifications from the OPC Group on completion of asynchronous Cancel.

This Callback occurs only after an asynchronous Cancel. Note that if the Cancel Request returned S_OK then the client can expect to receive this callback. If the Cancel request failed then the client should NOT receive this callback

5.2.2.4.1 *Properties*

The TsCDaCancelCompleteEventHandler delegate has the following properties:

Name	Description
requestHandle	An identifier for the request assigned by the caller.

Sample

An example of a TsCDaCancelCompleteEventHandler delegate:

```
public void OnCancelCompleteEvent(object requestHandle)
{
    Console.WriteLine("Transaction successfully cancelled");
}
```

Use of the TsCDaReadCompleteEventHandler delegate:

```
opcGroup.Cancel(tequest, new TsCDaCancelCompleteEventHandler(OnCancelCompleteEvent));
```

5.2.2.5 OpcServeShutdownEventHandler delegate

OpcServerShutdownEventHandler
Delegate
reason

This method is provided by the client so that the server can request that the client disconnect from the server. The client should unadvise all connections, Remove all groups and release all interfaces.

A client who is connected to multiple OPC Servers (for example Data access and/or other servers such as Alarms and events servers from one or more vendors) should maintain separate shutdown callbacks for each object since any server can shut down independently of the others.

5.2.2.5.1 *Properties*

The OpcServerShutdownEventHandler delegate has the following properties:

Name	Description
reason	The reason why the server requested a shutdown of the client.

6 OPC Complex Data

This chapter describes the OPC Complex Data Addition available for OPC Data Access client development**.**

The OPC Complex Data specification specifies new OPC DA item properties that define the structure of complex data items. Complex data is a term that describes OPC Data Access (DA) items whose values are constructed. OPC Complex Data provides a mechanism for OPC DA clients to discover the structure of the data values. This section introduces how OPC Complex Data relates to it. These properties can be used by any version of OPC DA.

Knowledge of the corresponding specifications is required for the understanding of this chapter.

6.1 Where OPC Complex Data Fits

The OPC DA specifications require DA items to be simple types or arrays of simple types. Therefore, prior to the OPC Complex Data Specification, OPC DA servers represented structured data simply as a sequence of bytes. The OPC DA specifications do not provide a mechanism for servers to describe the structure of these bytes, and as a result, clients that did not have apriori knowledge about the structure were unable to interpret them.

This specification defines the information, represented as OPC DA item properties, that OPC DA servers may make available to OPC DA clients to describe the structure of the data. Constructed data items whose structure is defined using these properties are known as complex data items.

An example of a complex data item is a data structure that represents a 'Connection' to some physical I/O device. It contains read-only configuration information as well as read-only runtime status information and writable control points. The elements of the structure are described in the following table:

Data Point	**Description**
Device Name	A unique identifier for an instance (i.e. the name of the device).
Device Settings	A list of command strings used to initialize the device on connection.
Wait Time	The time the server waits after a connect failure before re-connecting.
Connect State	Whether the server should attempt to connect to the device.
Last Connect Time	The last time the server connected to the device.
Connect Fail Count	The number of times the server attempted and failed to connect to the device.
Is Connected	Whether the device is currently connected.

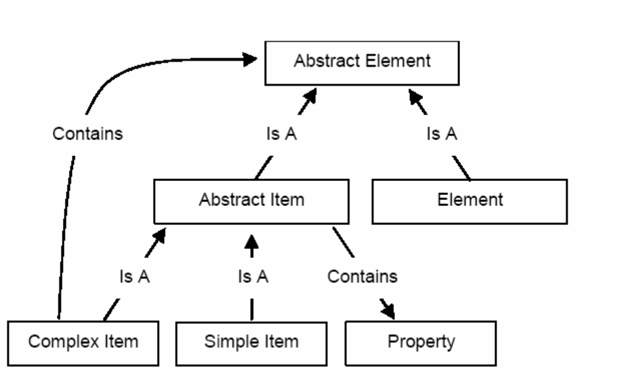
This structure could be represented in a DA server namespace as multiple DA items within a singlebranch; however, the DA client would not be able to read a coherent snapshot of the connection status since the status could change while each individual item is being read. Moreover, representing the structure a set of independent items means the all information about how the items relate to each other is lost.

OPC Complex Data solves these problems by:

- Providing a mechanism to describe complex data structures contained within the DA server namespace with either the type system defined in this specification or with any existing type system such as XML Schema.
- Providing a mechanism to represent complex structures as single DA items that clients may access as atomic blocks of data and still be able to access the contents of these atomics blocks by using the type descriptions.

6.2 Complex Data Model

Complex data is an OPC DA item whose value has a defined structure. A Complex Data item may be composed of a combination of structured data, simple items, and complex items. Structured data, in this context, means data that contains elements, none of which are themselves simple or complex items. When a complex data item is composed of other items, the client may be able to Browse the OPC DA Server to discover this relationship. The figure on the right illustrates this definition of complex data.

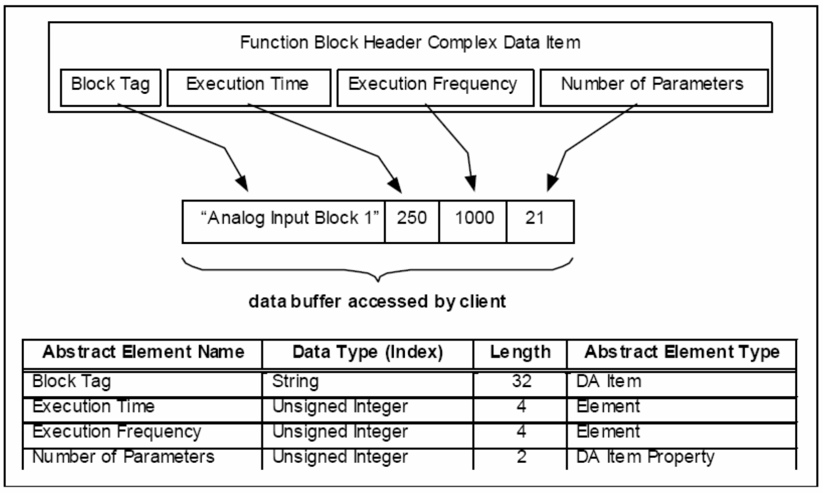


The figure above uses Abstract Elements, Abstract Items, Elements, Complex Items, Simple Items, and Item Properties to illustrate how complex data items can be constructed. The definition of each of these terms is defined below.

Term	Meaning
Abstract Element	Represents any component of a complex data item.
Abstract Item	Represents a simple or complex DA Item or DA Item Property
Element	A component of a complex data item that is not represented as a DA Item or DA Item Property. Elements may be simple or structured.
Complex Item	A DA Item that is structured. Complex data items can contain complex data items.
Simple Item	A DA Item with an unstructured data type.
Item Property	A DA Item Property

Complex data items can be composed of elements, complex data items, simple data items, or data item properties. Servers may expose abstract elements separately by defining them as Complex Items, Simple Items, or as Item Properties. Those defined as Elements are not individually accessible.

The figure on the right provides an example of a complex data item that exposes some abstract elements as "DA Items", "Item Properties", and "Elements". In this example, the Block Tag is exposed by the server as a DA Item. Therefore, it is accessible through the Function Block Header complex item, but also as a DA Item of its own. The Number of Parameters can also be accessed both as part of the Function Block Header complex item, and separately as a property of that complex item. Finally the Execution Time and Execution Frequency can only be accessed as part of the Function Block Header complex item.



Abstract Element Name	Data Type (Index)	Length	Abstract Element Type
Block Tag	String	32	DA Item
Execution Time	Unsigned Integer	4	Element
Execution Frequency	Unsigned Integer	4	Element
Number of Parameters	Unsigned Integer	2	DA Item Property

6.2.1 Complex Data Type Descriptions

Complex Data Type Descriptions define the structure of complex data items. Their descriptions are independent of whether or not some of the components are exposed by the OPC DA Server as other OPC Items. That is, the type description does not indicate whether or not any of the components are identified by ItemIDs.

The structure of a complex data item is normally defined by the organization responsible for defining the data. The mechanism used by that organization for describing the data to clients is referred to as a *Type System*.

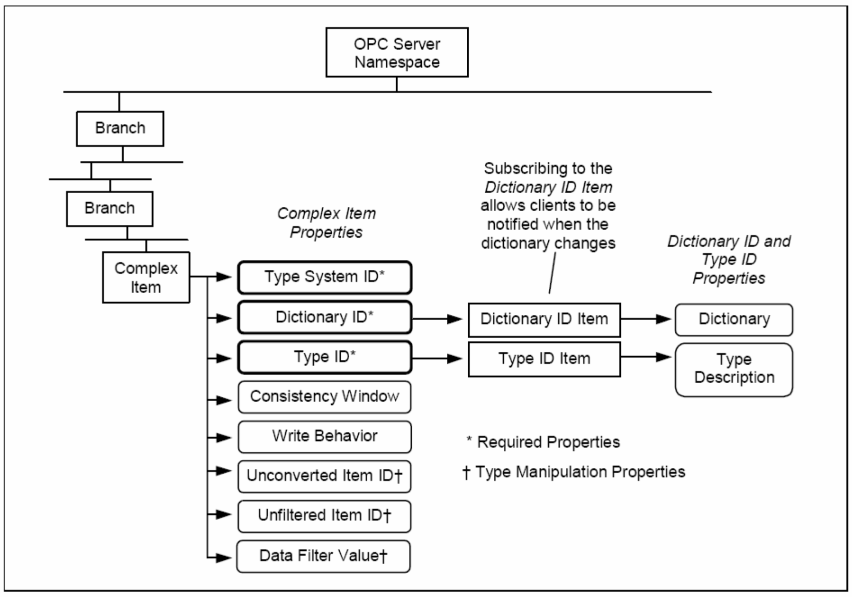
Type Systems normally produce Type Descriptions that enable clients to interpret the syntax of complex data, but not the semantics of the data. OPC defines two Type Systems that provide this level of capability, XML Schema and OPC Binary.

OPC defines the use of XML Schemas to describe complex data values represented in XML. OPC uses the OPC Binary Type System to describe complex binary data values. This system is described later in this section.

OPC DA Servers may use one or more Type Systems to describe complex data to clients. Therefore, clients may need to be able to understand more than one Type System. Allowing the use of multiple Type Systems ensures that the full capabilities of native Type Systems can be used to describe complex data.

6.2.2 Complex Data Item properties

Complex Data Type Descriptions are provided through DA Item Properties, as shown in the figure on the right.



A Dictionary is an entity that describes one or more complex types using a syntax defined by a Type System. A Type Description is a portion of a Dictionary that describes a single complex type. A Type Description may contain references to other complex types within the same Dictionary, as a result, a Type Description may not contain all information required to understand the complex type. A Dictionary, on the other hand, should contain all information that a DA client needs to understand the complex types it contains.

An example of a Dictionary is an XML document containing an XML Schema. In this case the Type System is 'XML Schema' and the top level element declarations are the 'Type Descriptions'.

The Type System ID, the Dictionary ID, and the Type ID properties form a hierarchy that identifies complex data type within a dictionary that conforms to the type system. It is expected that dictionaries may change during the lifetime of the OPC server, so the Dictionary ID provides for version control of the dictionary. Changes may be a result of a change to a complex data type but it is more likely that dictionary changes are a result of the addition or deletion of type descriptions.

DA clients should assume that Dictionaries and Type Descriptions are relatively large and that they will encounter performance problems if they automatically fetch the entire entity each time they encounter an instance of a specific complex type. The DA client should use the Type System ID, the Dictionary ID, and the Type ID properties to determine whether a locally cached copy of a Dictionary or Type Description is still valid.

When a DA clients wishes to read a Dictionary or Type Description for a complex data item, it must fetch the Item ID associated with the Dictionary ID or Type ID properties for the complex data item from the DA server. This Item ID references an item which exposes the either the Dictionary or Type Description properties. These items are refered to as the Dictionary ID Item and the Type ID Item in the diagram above.

The Item ID for the Dictionary ID Item or the Type ID Item should never be the same as the Dictionary ID or the Type ID. In addition, a Dictionary ID Item or the Type ID Item may be part of the Complex Data Namespace (see Section 6.2.4). The value of the Dictionary ID Item is the same as the value of the Dictionary ID property for a complex data item. This allows clients to subscribe to it to detect changes in the dictionary. Similarily, the value of the Type ID Item is the Type ID, however, it is not allowed to change, and therefore, DA clients have no need to subscribe to it.

Servers associate a Dictionary ID Item, as shown in Figure 5, with a given ictionary. The value of the Dictionary ID Item is the Dictionary ID, allowing clients to subscribe to it to detect changes in the dictionary. This item also provides client access to the Dictionary property. This property contains the dictionary. Standard OPC DA interfaces can be used to obtain the Item ID of the Dictionary ID Item from the Dictionary ID property.

When a dictionary contains descriptions of multiple complex items, the Item ID obtained throughassociated with the Dictionary ID property must be the same for each complex item. That is, when two complex items are described by the same version of the same Dictionary, their Dictionary ID properties must have equal values, they must all reference the same Dictionary ID Item.

These concepts also apply to the Type ID property, its associated Type ID item, and its associated Type Description property, except that the Type ID value is not allowed to change, and clients, therefore, have no need to subscribe to it.

The remaining complex item properties are optional. They provide additional information to clients to allow them to consistently use complex items.

The following table summarizes the Item Properties defined to support complex data. These properties are available with the TsDaProperty class.

Name	ID	Data Type	Description
TYPE_SYSTEM_ID	600	string	Type System ID Mandatory
DICTIONARY_ID	601	string	Dictionary ID Mandatory
TYPE_ID	602	string	Type ID Mandatory
DICTIONARY	603	BLOB	Dictionary The term BLOB is used here, and below, to represent a sequence of bytes whose data type depends on the Type System.
TYPE_DESCRIPTION	604	BLOB	Type Description Some servers may not have access to the type descriptions. In these cases, the client needs to be able to determine the structure of the data based on the Type ID. The term BLOB is used here, and below, to represent a sequence of bytes whose data type depends on the Type System.
CONSISTENCY_WINDOW	605	string	Consistency Window
WRITE_BEHAVIOR	606	string	Write Behaviour Defaults to "All or Nothing" if the complex data item is writable. Not used for Read-Only items.
UNCONVERTED_ITEM_ID	607	string	Unconverted Item ID The ID of the item that exposes the same complex data value in its native format. This property is mandatory for items that implement complex data type conversions.

UNFILTERED_ITEM_ID	608	string	Unfiltered Item ID The ID the item that exposes the same complex data value without any data filter or with the default query applied to it. It is mandatory for items that implement complex data filters or queries.
DATA_FILTER_VALUE	609	string	Data Filter Value The value of the filter that is currently applied to the item. It is mandatory for items that implement complex data filters or queries.

6.2.2.1 Type System ID Property

This property is identifies the Type System. If the server supports the CPX namespace (see Section 6.2.4), then this identifier is used as the name of the Type System branch in the namespace. A DA client must recognize the type system in order to make use of any of the type description information. The Type System ID is assigned by the DA server vendor, the entity responsible for the Type System, or by the OPC Foundation. The following Type Systems are defined by this specification.

Type System	Property Value	Description
XML Schema	XML Schema	The complex type is described by an XML Schema that conforms to the W3C XML Schema Specification.
OPC Binary	OPC Binary	The binary format of the complex data is described by an XML document that conforms to OPC Binary Dictionary schema defined in Section 6 of the OPC CPX specification.

The following restrictions apply to all items described with XML Schema type system:

- All item values are complete XML documents transported as strings;
- The Dictionary and Type Description properties are XML documents transported as strings;
- • The TypeID is the value of the 'name' attribute of the 'element' or 'complexType' element in the XML Schema. Nested types have a TypeID constructed by prepending the names of all ancestor elements followed by a forward slash ('/').

The following restrictions apply to all items described with OPC Binary type system:

- All item values are transported as arrays of unsigned bytes;
- The Dictionary and Type Description properties are XML documents transported as strings;
- The TypeID is the value of the 'TypeID' attribute for the 'TypeDescription' element.

6.2.2.2 Dictionary ID Property

Most type systems provide a set of standard types that may be used to define data. Many also allow extending these types with user-defined types. The resulting set of type descriptions is commonly referred to as a "dictionary", while the format of the dictionary is referred to as the dictionary schema.

This item property contains that string that identifes the dictionary that contains the definition of the complex data item (the one described by this property). The server may, but is not required to, make the dictionary available to clients. The Dictionary item property, described below, is used for this purpose. The value of this property depends on the DA server and the type system. The DA server is free to concatenate several pieces of information together to construct this identifier, but it must be unique within the type system. Different pieces of information should be separated by a CRLF. Examples of information a DA server could use include, but are not limited to, XML schema URI; device manufacturer; device, device firmware version; disk file name and modification time.

This identifier qualifies (scopes) the Type IDs used to identify individual Type Descriptions. That is, Type IDs are unique within the context of its dictionary. See the description of the Type ID item property below.

A change to any single Type Description scoped by this property results in a change to this identifier. This includes changes made while the DA server is offline so that the new Dictionary ID is available when the DA server restarts.

DA servers that support dynamic changes to this identifier must expose this property as a DA Item, and DA clients may detect dictionary changes by subscribing to this DA item. DA clients obtain the Item ID for this item via the appropriate DA interfaces. Refer to the appropriate DA specification for details on how Item IDs are associated with individual properties.

DA servers that support dynamic changes to Dictionaries must cache the value of this identifier whenever a DA client adds the item to a group. If the current value of this identifier differs from the original identifier then the DA Server must send an error of 'E_TYPE_CHANGED' instead of the value to the client for any read or data update operation. The DA server must also reject any writes from the client with the 'E_TYPE_CHANGED' error.

6.2.3 Type ID Property

This property is an identifier for the Type Description of a complex data item. The identifier is unique within the context of the Dictionary ID. The syntax for the identifier is defined by the specified type system. Usually, this property contains an identifier that can be used to locate a type within the dictionary. Values for this identifier are defined by the type system or by the source of the data that the type description describes (i.e. the device that is the source of the complex data).

6.2.3.1 Dictonary Property

The dictionary associated with the Dictionary ID Item may be represented by a single, consolidated schema, dictionary, or similar entity.

This property is a BLOB that contains the complete dictionary. The formatting of the BLOB is defined by the type system. For the "XML Schema" type system, the BLOB contains a valid XML Schema document. For the "OPC Binary" type system, the BLOB contains a string that is a valid XML document whose schema is defined in Section 6 of the OPC CPX specification.

The DA client may assume that it only needs to read this BLOB once for the first complex data item it encounters with the type system and Dictionary ID for this BLOB. The DA server must use the same Dictionary ID for the same BLOB across all clients and all sessions.

The DA server must ensure that any change to the contents of the BLOB is matched with a corresponding change to the Dictionary ID. This includes changes made while the DA server is offline so that the new Dictionary ID is available when the DA server restarts. In other words, the DA client may safely cache the dictionary between sessions with a DA server.

6.2.3.2 Type Description Property

The Complex Data Type Item Description Property is a BLOB that contains the information necessary for clients to interpret the value of a complex data item. It is not used with data item values that are not complex.

This property describes the syntax of the complex data value, allowing clients to parse the complex data value into its component parts. This property may, or may not be present if the Dictionary property is present.

When this property is present, the corresponding Complex Data "Type ID" Property must also be present. However, the converse is not true; for some complex data items the Complex Data Type ID Item Property will be present, but this property will not.

6.2.3.3 Consistency Window Property

This property is a string that indicates support for time consistency between and among elements of the complex data item. The value is normally the string form of an integer that specifies in milliseconds the range between the oldest and newest element values. For example, the value "100" indicates that all element values were accessed and updated by the server within 100 milliseconds of each other. The value "0" indicates the values are consistent.

For the case where the DA server cannot quantify the window with a specific time period, but knows the values to be consistent, the value "Unknown" is used. The value "Not Consistent" indicates that the data is not consistent. If this property is not present, then the server does not support time consistent access for the related complex data item, and the client can make no assumptions about its time consistency.

Servers may expose this property as an OPC Item to allow clients to request a different consistency window. If the server exposes this property in this manner, and the client requests an unsupported value, then the server returns an error.

6.2.3.4 Write Behaviour Property

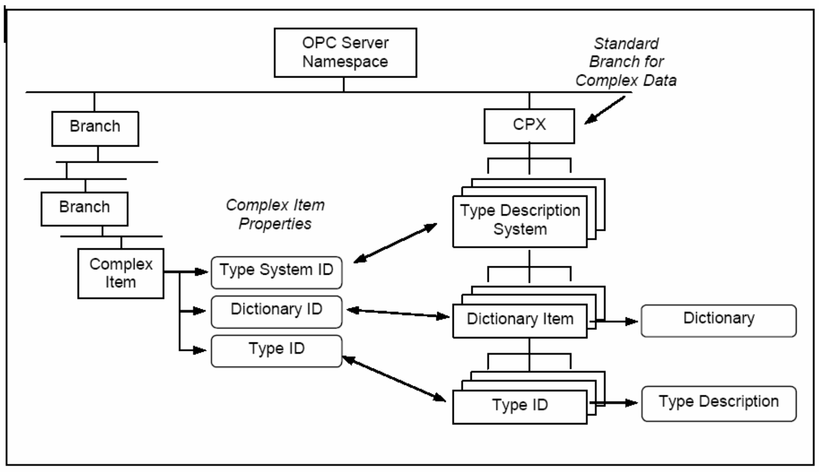
This property is a string that indicates whether the server supports "All or Nothing" or "Best Effort" writes to the complex data value.

"All or Nothing" is the default value. It means that writes to all writable elements need to succeed or the write operation to the complex data value fails. If the client does not know which elements are read-only, then the client should perform a read operation after a successful write to determine which elements were updated, and are, consequently, read-only.

"Best Effort" means that a write succeeds if any of the elements can be updated. If "Best Effort" is supported, the client is advised to perform a read operation after a successful write to determine which elements were updated.

6.2.4 Complex Data Namespace

In addition to providing support for the Complex Data Item Properties just described, OPC Servers may also extend their namespace to allow clients to browse for supported complex data descriptions. The "CPX" branch is reserved for this purpose. Its relationship to Complex Data Item properties is shown in the figure on the right.



The following table describes each of the components of the CPX subtree.

Item/Branch	Description
Type System ID	Used to identify the type systems supported by the OPC Server. The name of the type system is used as the name of this branch and also as the value of the corresponding type system property (sees Section 6.2.2.1).
Dictionary ID	Used to identify the dictionaries supported for each type system. The name of the branch identifies the dictionary independent of its version. The value of this item identifies the dictionary and its current version. This value is used as the value of the Dictionary ID Item Property (see Section 6.2.2.2). Clients may subscribe to this item to detect changes to the dictionary. The associated Dictionary property (see Section 6.2.3) contains the latest copy of the dictionary.
Type ID	Used to identify the type descriptions within each dictionary. The Type ID is used as the name of this branch and also as the value of the corresponding Type ID Item Property (see Section 6.2.3). If the type system provides both a Type Name and a Type ID, the Type Name is used for the value of this item. Otherwise, the Type ID is used.

6.2.5 Complex Data Behaviour

Type Conversion:

Clients that access OPC data items may request the server to perform certain data type conversions on the item value when sending the value to the client. This capability is not supported for complex data. In other words, DA clients may only use VT_EMPTY as a requested data type for reads and updates. However, this specification does provide a mechanism to request alternate representations of a complex data item. This mechanism in described in Section 7 of the OPC CPX specification.

OnDataChange:

OPC data items may be subscribed to by clients, causing callbacks to be triggered when changes in the value of the item exceeds the deadband specified for the subscription. For complex data items, the callback is triggered when any of the elements of the complex data item exceeds the specified deadband. It is up to the DA server to determine how the deadband should be applied to any given complex data item. DA clients may control how the DA server checks for data changes by using the Data Filter items described in Section 8 of the OPC CPX specification.

Quality:

OPC data items have a quality code that describes the quality of the data value in read responses and subscription callbacks. The quality code for complex data items applies to the entire complex data item, and reflects the poorest quality of its elements. That is, the quality code for the complex data item is taken from the element that has the poorest quality.

Timestamp:

OPC data items have timestamps that accompany the data value in read responses and subscription callbacks. The timestamp for complex data items applies to the entire complex data item, and reflects that latest update to any of its elements. That is, the timestamp is updated each time an element of the complex data value is changed. Time Consistency

Window:

Complex data items may be capable of supporting time consistency between their elements. The Consistency Window Item Property is used to indicate support for this capability. To make it possible for clients to request a specific time consistency window the server may expose this property as an OPC Item and allow writes to the property. If the client requests an unsupported time consisteny window, then the server returns a negative response.

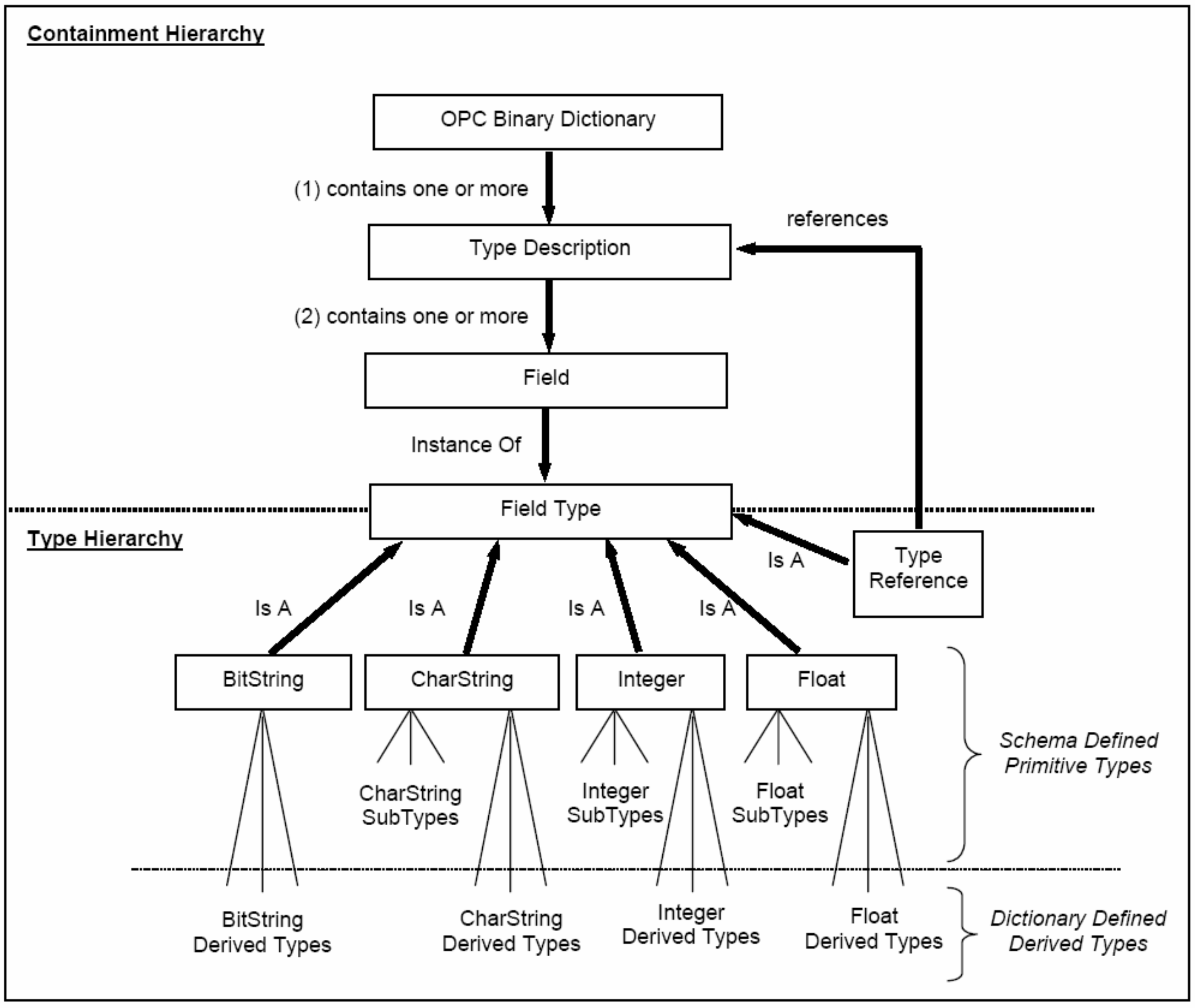
Write:

Complex data items may be written by clients. When written, not all elements of the complex data item may be writable. In this case, if the client supplies the entire buffer in a write request, the server ignores the values of non-writable elements. In the case where the value for one or more of the writable elements is invalid, or cannot be applied, the server rejects the entire write request if the “Write Behavior” Item Property indicates “All or Nothing”. If this property indicates “Best Effort”, then the write succeeds if one element is successfully written by the server.

6.2.6 OPC Binary Type System

The OPC Binary Schema defines the format of OPC Binary dictionaries. Each OPC Binary dictionary is an XML document that describes of one or more OPC Complex Data items whose values are represented in binary. Each description defines the format of the binary value of a complex data item, thereby allowing DA clients to parse complex binary values that it receives from an OPC Server.

The figure below illustrates the structure of the dictionary.



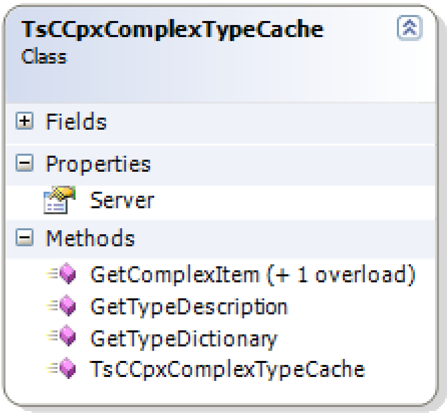
Please see the OPC Complex Specification for more information.

6.3 Client API

6.3.1 Classes

6.3.1.1 TsCCpxComplexTypeCache class

This class is a base class for caching properties of complex data items.



It provides methods for setting up a complex data items cache based on a OpcItem or a TsCDaBrowseElement. To be able to use this class you must set the Server property to the TsCDaServer object and then connect to an OPC DA Server Server, e.g.

```
TsCDaServer myDaServer = new TsCDaServer();
TsCCpxComplexTypeCache.Server = myDaServer;

myDaServer.Connect( serverUrl );
```

6.3.1.1.1 Properties

The TsCCpxComplexTypeCache class has the following properties:

Name	Description
Server	Get or sets the server to use for the cache.

6.3.1.1.2 Methods

The TsCCpxComplexTypeCache class has the following methods:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>GetComplexItem</td><td>

TsCCpxComplexItem GetComplexItem(OpcItem itemID)

Returns the complex item for the specified item id.

This method has the following parameters:

Name	Description
itemID	The item id.
[Return Value]	Either the complex item or null in case of an error.

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.
E_INVALIDARG	The argument to the function was invalid. (For example, the itemID is not a complex item.)

</td></tr>
</table>

GetComplexItem	**TsCCpxComplexItem GetComplexItem(TsCDaBrowseElement element)** Returns the complex item for the specified item browse element. This method has the following parameters: **Name** — **Description** element — The item browse element. [Return Value] — Either the complex item or null in case of an error. Possible errors are: **Name** — **Description** E_FAIL — The operation failed. CONNECT_E_NOCONNECTION — There is no existing connection. E_INVALIDARG — The argument to the function was invalid. (For example, the itemID is not a complex item.)
GetTypeDescription	Fetches the type description for the item. This method has the following parameters: **Name** — **Description** itemID — The item id. [Return Value] — The type description for the item. Possible errors are: **Name** — **Description** E_FAIL — The operation failed. CONNECT_E_NOCONNECTION — There is no existing connection. E_INVALIDARG — The argument to the function was invalid. (For example, the itemID is not a complex item.)
GetTypeDictionary	Fetches the type dictionary for the item. This method has the following parameters: **Name** — **Description** itemID — The item id. [Return Value] — The type dictionary for the item. Possible errors are: **Name** — **Description** E_FAIL — The operation failed. CONNECT_E_NOCONNECTION — There is no existing connection. E_INVALIDARG — The argument to the function was invalid. (For example, the itemID is not a complex item.)

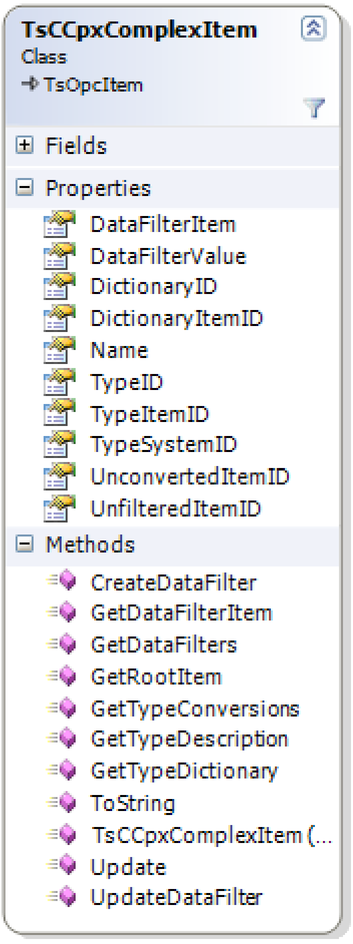
6.3.1.2 TsCCpxComplexItem class

This class contains complex data related properties for an item.

It provides methods for getting a complex item based on a OpcItem, e.g.:

```
TsCDaItem[] items = new TsCDaItem[1];
items[0] = new TsCDaItem();
items[0].ItemName = "Complex Data/Dynamic/FixedArrays";

TsCCpxComplexItem complexItem =
TsCCpxComplexTypeCache.GetComplexItem(items[0]);
```



6.3.1.2.1 Properties

The TsCCpxComplexItem class has the following properties:

Name	Description
DataFilterItem	The item used to create new data filters for the complex data item (null is item does not support it).
DataFilterValue	The current data filter value. Only valid for items apply data filters to the item.
DictionaryID	The dictionary id for the complex item.
DictionaryItemID	The id of the item containing the dictionary for the item.
Name	The name of the item in the server address space.
TypeID	The type id for the complex item.
TypeItemID	The id of the item containing the type description for the item.
TypeSystemID	The type system id for the complex item.
UnconvertedItemID	The id of the unconverted version of the item. Only valid for items which apply type conversions to the item.
UnfilteredItemID	The id of the unfiltered version of the item. Only valid for items apply data filters to the item.

6.3.1.2.2 *Methods*

The TsCCpxComplexItem class has the following methods:

Name	Description
CreateDataFilter	Creates a new data filter. This method has the following parameters:

Name	Description
server	The server object
filterName	The name of the filter
filterValue	The value of the filter
[Return Value]	Either the complex item or null in case of an error.

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.
E_FILTER_ERROR	An error with the specified filter occurred, e.g. new data filter could not be created.
E_INVALIDARG	The argument to the function was invalid. (For example, the itemID is not a complex item.)

Name	Description
GetDataFilterItem	Fetches the item id for the data filters items and stores it in the internal cache. This method has the following parameters:

Name	Description
server	The server object

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.

Name	Description
GetDataFilters	Fetches the set of data filters from the server. This method has the following parameters:

Name	Description
server	The server object
[Return Value]	Either the complex items array or or null in case of an error.

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.
E_FILTER_ERROR	An error with the specified filter occurred, e.g. new data filter could not be created.
E_INVALIDARG	The argument to the function was invalid. (For example, the itemID is not a complex item.)

GetRootItem	Returns the root complex data item for the object. This method has the following parameters: **Name** | **Description** [Return Value] | Either the complex items array or or null in case of an error. Possible errors are: **Name** | **Description** E_FAIL | The operation failed. CONNECT_E_NOCONNECTION | There is no existing connection.
GetTypeConversions	Fetches the set of type conversions from the server. This method has the following parameters: **Name** | **Description** server | The server object [Return Value] | Either the complex items array or or null in case of an error. Possible errors are: **Name** | **Description** E_FAIL | The operation failed. CONNECT_E_NOCONNECTION | There is no existing connection.
GetTypeDescription	Fetches the type description for the item. This method has the following parameters: **Name** | **Description** server | The server object [Return Value] | Either the type description as string or null in case of an error. Possible errors are: **Name** | **Description** E_FAIL | The operation failed. CONNECT_E_NOCONNECTION | There is no existing connection.
GetTypeDictionary	Fetches the type dictionary for the item. This method has the following parameters: **Name** | **Description** itemID | The item id. [Return Value] | The type dictionary as string for the item or null in case of an error.. Possible errors are: **Name** | **Description** E_FAIL | The operation failed. CONNECT_E_NOCONNECTION | There is no existing connection.

Update	Reads the current complex data item properties from the server. This method has the following parameters:

Name	Description
server	The server object

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.
E_INVALIDARG	The argument to the function was invalid. (For example, the itemID is not a complex item.)

UpdateDataFilters	Updates a data filter. This method has the following parameters:

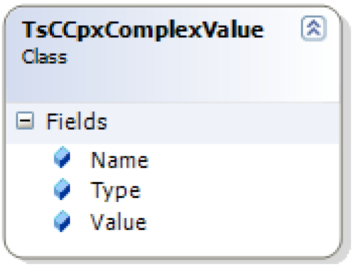
Name	Description
server	The server object
filterValue	The value of the filter

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.
E_FILTER_ERROR	An error with the specified filter occurred, e.g. new data filter could not be created.
E_INVALIDARG	The argument to the function was invalid. (For example, the itemID is not a complex item.)

6.3.1.3 TsCCpxComplexValue class

Stores a value with an associated name and/or type.



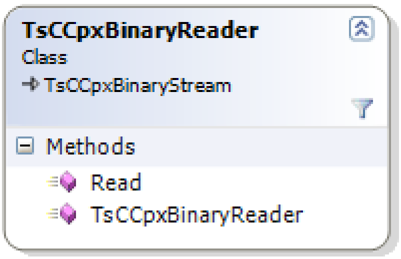
6.3.1.3.1 Fields

The TsCCpxComplexValue class has the following properties:

Name	Description
Name	The name of the value.
Type	The complex or simple data type of the value.
Value	The actual value.

6.3.1.4 TsCCpxBinaryReader class

This class reads a complex data item from a binary buffer.



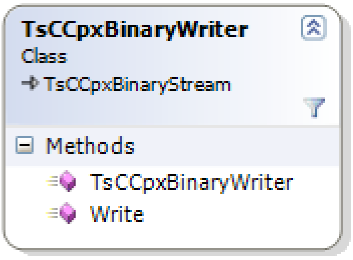
6.3.1.4.1 Methods

The TsCCpxBinaryReader class has the following methods:

Name	Description
Read	Reads a value of the specified type from the buffer. This method has the following parameters: **Name** / **Description** buffer — The buffer containing binary data to read. dictionary — The type dictionary that contains a complex type identified with the type name. typeName — The name of the type that describes the data. [Return Value] — A structured represenation of the data in the buffer. Possible errors are: **Name** / **Description** E_FAIL — The operation failed. CONNECT_E_NOCONNECTION — There is no existing connection. E_INVALIDARG — The argument to the function was invalid. (For example, the itemID is not a complex item.)

6.3.1.5 TsCCpxBinaryWriter class

This class writes a complex data item to a binary buffer.



6.3.1.5.1 Methods

The TsCCpxBinaryWriter class has the following methods:

Name	Description
Write	Writes a complex value to a buffer. This method has the following parameters:

Name	Description
namedValue	The structured value to write to the buffer.
dictionary	The type dictionary that contains a complex type identified with the type name.
typeName	The name of the type that describes the data.
[Return Value]	A buffer containing the binary form of the complex type.

Possible errors are:

Name	Description
E_FAIL	The operation failed.
CONNECT_E_NOCONNECTION	There is no existing connection.
E_INVALIDARG	The argument to the function was invalid. (For example, the itemID is not a complex item.)

T

License Model …

Technosoftware GmbH offers various products around OPC and licenses on a simple per-developer basis and charges no additional royalties or run-time fees unless otherwise specified which maximizes cost savings at application distribution and installations. When you buy a license you receive the following:

- The right for one named and registered developer per license to use and develop commercial applications with the license. All developers in your company using the license will need individual licenses.
- Maintenance: Three or twelve months of upgrades to all new releases of the license
- Support: Three or twelve months of Email- and Helpdesk based Support Incidents.

Depending on the license different permissions for the use of the software are granted.

Developer License:

A developer license grants the use of the Software for the purpose of soft-ware development of OPC enabled products or applications. If you have purchased a developer license the Software may be used within your organization by a single developer to develop applications. Multiple copies of the Software may exist on more than one computer, as long as the use of the Software is by the same developer. In addition, you are permitted to deploy the application making use of the Software. The Software might be used by being compiled into, linked or bind to your application. All use of the Software shall be solely in accordance with the documentation.

You are not allowed to resell, rent, lease or sublicense an unmodified or modified version of this Software as stand-alone product, nor to build toolkit or developer tools from it. You are permitted to use the software to build, deploy, sell and distribute end-user products. Licensee must comply with all of the following:

a. Licensee is permitted to distribute the applications that are built with the Software only in con-junction with and as an integral part of your application, and distribute the binary files only as an integral part of your end-user application. Redistributables, if any, shall be licensed to Licensee's customer "as is"

b. Licensee’s software product(s) are not an OPC Server and/or Client development tool; licensee may not use the Software in such a way that results in development of product(s) that are direct-ly or indirectly (i.e. simplified version or plat-form/language adoption) competitive with the licensed Software itself or other Technosoftware developer products family

c. Licensee’s applications must add primary and substantial functionality to the licensed Software; applications may not pass on functionality which in a simple way makes it possible for others to create software with the licensed Software (e.g. you may not distribute libraries and correspond-ing header files of the Software together with your application)

d. Licensee shall indemnify and hold Technosoftware, its affiliates, contractors, and its suppliers, harmless from and against any claims or liabilities arising out of the use, reproduction or distribution of applications

e. Licensee may not use Technosoftware’s name, logo or trademark to market your application without explicit written agreement with Technosoftware

Unlimited Developer License:

In addition to the Developer License; if you have purchased the unlimited developer license of the Software, the use of the software is granted for all developers that are employees of the specified Site of the company the license was purchased for. You may not make accessible the Software to third parties and you must make sure that no one except your authorized employees has access to the computer(s) where the Software is installed. You are not allowed to use the software by developers that are employees of affiliates or otherwise associated or subsidiary companies. Only for the develop-ment of a component or product of the licensee, the licensee is allowed to give external partici-pants - of this particular development project - access to the software. The licensee must insure that the external development partner is following all rules of this SLA and that the development partner is not using the software for any other purpose than this particular development project with the licensee.

Source License:

The Source license grants the same rights as the Unlimited Developer License. Access to the source code has to be restricted to the persons actually needing it and precautions have to be taken to prevent the software from being illegally copied.

Maintenance and Upgrade Protection

Renewal/Reinstatement

Technosoftware offers customers four options for purchase.

- Without Maintenance & Upgrade Package
- With 3 months Maintenance Package
- With 12 months Maintenance &amp; Upgrade Package
- With 24 months Maintenance &amp; Upgrade Package

What is included in a product purchase without Maintenance & Upgrade Package?
Included in every new product purchase without Maintenance & Upgrade Package, are

- Access to the current line version of the product.
- Minor-Updates of the current line version. If you purchase for example a new license of the OPC UA Client SDK .NET and the current line version is V 5.x, you get all updates for V 5.x but not for new release major versions, e.g. V6.x or newer.
- You can report us errors via Issue Tracker or Email and we handle those errors and fix errors for the current line and long term support version. Issues must always be reproducible at Technosoftware GmbH's company location.
- Upgrading to a newer major version is possible by purchasing a new license.

What is included in a product purchase with 3 months Maintenance?
Included in every new product purchase with 3 months Maintenance Package, are

- Access to the current line version of the product.
- Minor-Updates of the current line version. If you purchase for example a new license of the OPC UA Client SDK .NET and the current line version is V 5.x, you get all updates for V 5.x but not for new release major versions, e.g. V6.x or newer.
- You can report us errors via Issue Tracker or Email and we handle those errors and fix errors for the current line and long term support version. Issues must always be reproducible at Technosoftware GmbH's company location
- Demonstration of Issues via Remote Session. This is for example required if you can't provide us an OPC Server or OPC Client needed for reproducing the error and it happens only within your environment.
- You can discuss Issues via Phone at +41 56 536 93 46. This is only possible during our business hours. During Company Holidays this is not possible.
- Upgrading to a newer major version is possible by purchasing a new license.

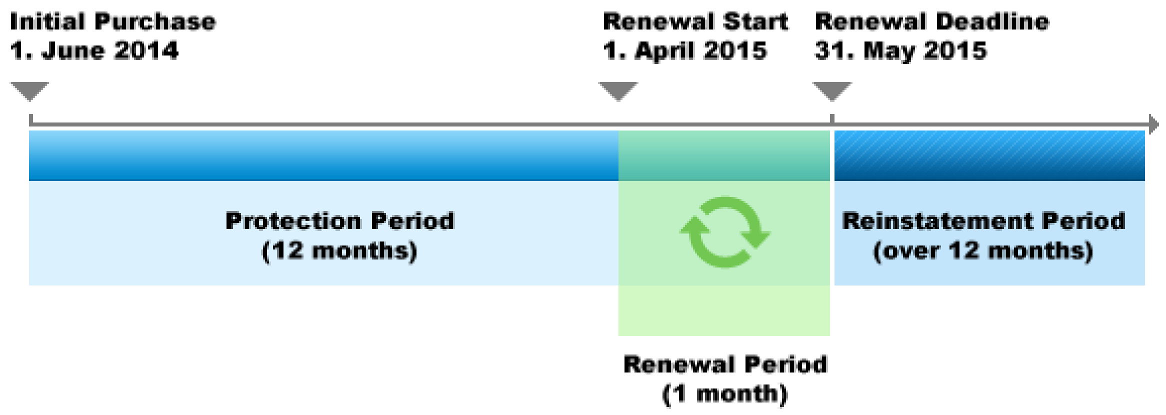
What is included in a product purchase with 12 or 24 months Maintenance & Upgrade Package?
Included in every new product purchase with 12 or 24 months Maintenance & Upgrade Package are:

- Access to the current line and long term support version of the product.
- Minor- and Major-Updates of the current line and long term support version. If you purchase for example a new license of the OPC UA Client SDK .NET and the current line version is V 5.x, you get all updates for V 5.x as well as for the long term support version V4.x. If we release a new major version, e.g. V6.x, during the 12 or 24 months you get also access to this version.
- You can report us errors via Issue Tracker or Email and we handle those errors and fix errors for the current line and long term support version. Issues must always be reproducible at Technosoftware GmbH's company location. Issues are handled with a higher priority than those of customers without Maintenance & Upgrade Package.
- Demonstration of Issues via Remote Session. This is for example required if you can't provide us an OPC Server or OPC Client needed for reproducing the error and it happens only within your environment.
- You can discuss Issues via Phone at +41 56 536 93 46. This is only possible during our business hours. During Company Holidays this is not possible.
- Extending the included Maintenance & Upgrade period of 12 or 24 months can be done by purchasing a product license without Maintenance & Upgrade Package.

Detailed example of how the Maintenance & Upgrade Package is handled.

Technosoftware customers are provided 12 months of maintenance and upgrade protection coverage with the purchase of any new license. Maintenance and Upgrade protection includes access to all major and minor version upgrades for 12 months from the date of purchase at no additional charge.

For example, if a new OPC UA Client SDK .NET license is purchased on 1. June 2014, upgrade protection will expire on 31. May 2015. During this time, the customer can download and install any minor version upgrades—and, if Technosoftware issues a major release of OPC UA Client SDK .NET before expiration date, the license can be upgraded to the latest version at no additional charge. With Technosoftware's maintenance and upgrade protection coverage, customers will always have access to the latest features and fixes.



Does maintenance and upgrade protection automatically renew?

Actually, no. Auto-renewal can be a hassle for many businesses. Therefore, Technosoftware begins contacting customers well ahead of their expiration date, and a number of automatic messages are sent out with varying frequency reminding customers of their upgrade protection expiration date and how to renew. Technosoftware makes every effort to notify customers of their renewal date so that they can take advantage of the significant savings available.

I have current maintenance and upgrade protection coverage. How do I upgrade my software to the latest version?

If you have current maintenance and upgrade protection coverage, Technosoftware will automatically upgrade your license key to the most recent version when new versions of our products are released. This will allow you to download and install the newest version of the product from the Technosoftware website.

How do I extend/renew my maintenance and upgrade protection?

Customers can renew their upgrade protection up to 60 days before their current coverage expires, ensuring continuous coverage for an additional 12 months at only **30% of the new product price**. The product license option without Maintenance & Upgrade Package is used also for extend/renew a maintenance and upgrade protection.

For example, the customer that purchased a new OPC UA Client SDK .NET license on 1. June 2014, may renew his upgrade protection as early as 1. April 2015 (2 months before his current upgrade protection coverage expires) to extend his upgrade protection coverage through 31. May 2016.

My maintenance and upgrade protection expired. How do I reinstate maintenance and upgrade protection coverage?

Technosoftware offers maintenance and upgrade protection reinstatements to customers whose maintenance and upgrade protection coverage has expired for **80% of the new product price**. Maintenance and Upgrade protection reinstatements allow the customer to upgrade to the most recent version of the product and include 12 months of maintenance and upgrade protection coverage at no additional charge.

For example, if a customer purchases a new OPC UA Client SDK .NET license on 1. June 2014, and does not renew maintenance and upgrade protection coverage, the initial coverage term will expire on 31. May 2015. After this date, a maintenance and upgrade protection reinstatement is required. If he purchases a reinstatement on 1. November, 2015, he will be able to upgrade to the latest version of OPC UA Client SDK .NET and will receive upgrade protection coverage through 31. October 2016 at no additional charge.

Because reinstatements are priced much higher than renewals, it is generally more cost effective to maintain current upgrade protection and remain on the latest version than to allow upgrade protection coverage to expire and then reinstate later.

You are recommended to re-order a Maintenance Package before expiry date. If you re-order Maintenance Package after the expiry date, the expiry date of the original Maintenance Package will be considered as the start date of the re-ordered Maintenance Package.

T

Why Technosoftware GmbH?…

- Professionalism
  Technosoftware GmbH is, measured by the number of employees, truly not a big company. However when it comes to flexibility, service quality, and adherence to schedules and reliability, we are surely a great company which can compete against the so called leaders in the industry. And this is THE crucial point for our customers.

- Continuous progress
  Lifelong learning and continuing education is, especially in the information technology, essential for future success. Concerning our customers, we will constantly accepting new challenges and exceeding their requirements again and again. We will continue to do everything to fulfill the needs of our customers and to meet our own standards.

- High Quality of Work
  We reach this by a small, competent and dynamic team of coworkers, which apart from the satisfaction of the customer; take care of a high quality of work. We concern the steps necessary for it together with consideration of the customers' requirements.

- Support
  We support you in all phases – consultation, direction of the project, analysis, architecture & design, implementation, test and maintenance. You decide on the integration of our coworkers in your project; for an entire project or for selected phases.

Technosoftware GmbH
Windleweg 3, CH-5235 Rüfenach
Tel.: +41 56 536 93 46
sales@technosoftware.com
www.technosoftware.com