Technosoftware AG, Farmweg 4, CH-5702 Niederlenz
PHONE: +41 62 888 40 40, FAX: +41 62 888 40 45
www.technosoftware.com
E-Mail: sales@technosoftware.com

OPC Framework .NET

Developer Guide Part I

- General Information
- OPC DA
- OPC XML-DA

Technosoftware AG

Farmweg 4

CH-5702 Niederlenz, AG

Phone: +41 62 888 40 40

Fax: +41 62 888 40 45

sales@technosoftware.com

www.technosoftware.com

Document Control

Version	Author	Date	Comment
1.0	TJ	21-11-2006	Initial version of Part I

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 based on the .NET programming model.

The main reason for this API is to provide interoperability between existing OPC specifications and applications developed in the .NET environment. The API does support COM based (OPC Data Access, OPC Alarms&Events, OPC Historical Data Access) and SOAP/XML based (OPC XML-DA) servers via the same set of interfaces.

This document is intended as reference material for developers of OPC compliant Client and server applications. It is assumed that the reader is familiar with the various OPC specifications, Microsoft COM/DCOM technology, XML Schemas, Microsoft's .NET Framework and the needs of the Process Control industry.

Summary

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

Disclaimer

This document was created strictly for information purposes. No guarantee, contractual specification or condition shall be derived from this document unless agreed to in writing. Technosoftware AG reserves the right to make changes in the products and services described in this document at any time without notice and this document does not represent a commitment on the part of Technosoftware AG in the future.

While Technosoftware AG uses reasonable efforts to ensure that the information and materials contained in this document are current and accurate, Technosoftware AG makes no representations or warranties as to the accuracy, reliability or completeness of the information, text, graphics, or other items contained in the document. Technosoftware AG expressly disclaims liability for any errors or omissions in the materials contained in the document and would welcome feedback as to any possible errors or inaccuracies contained herein.

Technosoftware AG shall not be liable for any special, indirect, incidental, or consequential damages, including without limitation, lost revenues or lost profits, which may result from the use of these materials. All offers are non-binding and without obligation unless agreed to in writing.

TABLE OF CONTENTS

1 Introduction

Over the last seven years OPC has become the de facto standard for open communication between SCADA systems and field devices.

The first versions of the OPC standards are based on the DCOM architecture defined by Microsoft. These standards are limited to Microsoft Windows based operating systems because of the use of the DCOM architecture. Only a few implementations based on other operating systems exist, like Technosoftware AG solutions.

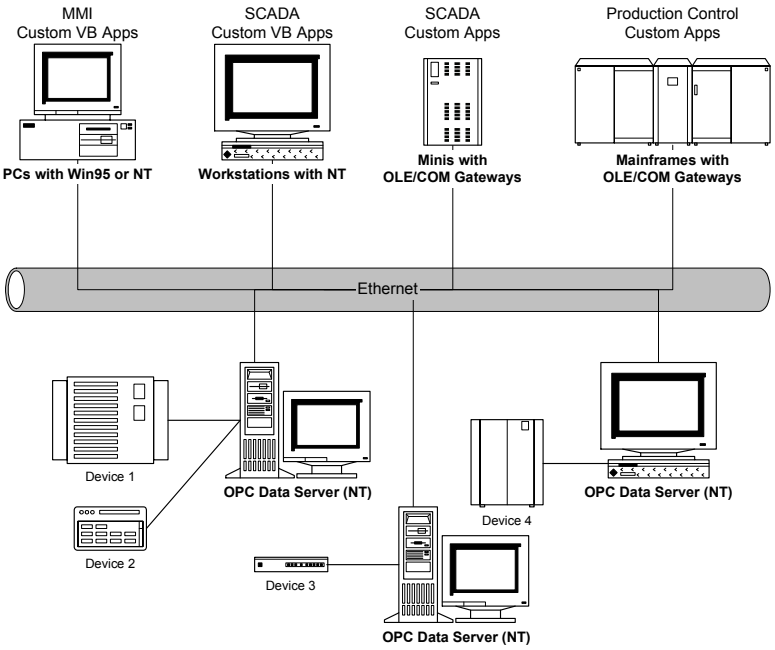
With the .NET initiative, Microsoft is now promoting another distributed architecture beside DCOM: the web services environment based on XML and W3C protocols. In a similar move, OPC is also specifying XML based communication.

The OPC Foundation also announced the move to a "unified architecture" that will now be totally interoperable with full internet connectivity and unlimited scalability across platforms of the end-users choice.

Most of the upcoming implementations are based on Microsoft Windows and the .NET architecture. At least for servers based on the new specifications this results in large and complex environments required by Microsoft OSs to run web services (.NET framework and IIS web server). But also for clients the .NET framework is still required. Implementing OPC clients or servers for these new specifications is complex if the target operating system is not Microsoft based, e.g. Linux.

1.1 Why OPC?

In the automation industry very often devices from different hardware suppliers and software packages like visualization systems and process-control software from several software suppliers have to be combined to build a complete system. Within this system the different software components need to communicate. The application software should communicate with I/O devices as well as other applications. Getting the different software modules to work together is the biggest problem for process systems manufacturers.



These problems are due to missing or incompatible standards for data exchange interfaces. In the past vendors developed proprietary hardware and software solutions. All process-control and information systems today have their own interface to access the information.

Often a driver for one I/O device was written several times by different vendors. This can cause inconsistencies among different custom drivers or upgrades. It may also be impossible to use different software packages with one device at the same time because they use independent drivers and hardware features that are not supported by a custom driver.

In the past hardware vendors tried to solve some of these problems by delivering their own drivers.

The solution today is having a standard plug-and-play software technology for process-control and automation. Having such a standard makes it possible that different software packages can freely connect and communicate with different devices. This results in a truly open and easy enterprise-wide communication between systems and devices on the field, process or business management hierarchy.

1.2 What is OPC?

OPC is open connectivity in industrial automation and the enterprise systems that support industry. Interoperability is assured through the creation and maintenance of open standards specifications. There are currently seven standards specifications completed.

Currently the following specifications are available:

1.2.1 OPC Common V1.0

This specification contains common rules and design criteria and the specification of interfaces which are common for several other specifications, e.g., for Data Access, Alarm&Event Handling or Historical Data Access.

1.2.2 OPC Data Access V1.0a, V2.05 and V3.0

This specification is used to move real-time data from PLCs, DCSs, and other control devices to HMIs and other display clients.

1.2.3 OPC Alarms&Events V1.1

This specification provides alarm and event notifications on demand (in contrast to the continuous data flow of Data Access). These include process alarms, operator actions, informational messages, and tracking/auditing messages.

1.2.4 OPC Historical Data Access V1.1

Where OPC Data Access provides access to real-time, continually changing data, OPC Historical Data Access provides access to data already stored. From a simple serial data logging system to a complex SCADA system, historical archives can be retrieved in a uniform manner.

1.2.5 OPC Batch V2.0

This specification carries the OPC philosophy to the specialized needs of batch processes. It provides interfaces for the exchange of equipment capabilities (corresponding to the S88.01 Physical Model) and current operating conditions.

1.2.6 OPC Security V1.0

All the OPC servers provide information that is valuable to the enterprise and if improperly updated, could have significant consequences to plant processes. OPC Security specifies how to control client access to these servers in order to protect this sensitive information and to guard against unauthorized modification of process parameters.

1.2.7 OPC Data eXchange V1.0

This specification takes us from client/server to server-to-server with communication across Ethernet fieldbus networks. This provides multi-vendor interoperability! And, oh by the way, adds remote configuration, diagnostic and monitoring/management services.

1.2.8 OPC XML-DA V1.0

Provides flexible, consistent rules and formats for exposing plant floor data using XML, leveraging the work done by Microsoft and others on SOAP and Web Services.

1.3 What is COM?

The 'Component Object Model' provides interfaces and inter-component communication. Through COM, an application may use features of any other application object. COM is the core of DCOM, ActiveX and OLE.

1.4 What is OLE?

Object Linking and Embedding is used to provide integration among applications and allows the development of reusable objects that are interoperable between multiple applications. It also provides reusable, component-based software solutions, where as the software-components can be written in any language.

1.5 What is DCOM?

The Distributed Component Object Model extends COM to work over a network. It is a protocol where remote components appear to be local.

1.6 What is ActiveX?

ActiveX is an open, integrated platform for portable applications and interactive content for the World Wide Web.

1.7 What are Web Services?

The web services framework is today the de facto standard for open, platform and language independent communication in IT systems. The main IT companies (SUN, IBM, Microsoft…) integrate it into their products.

1.8 OPC Specifications

1.8.1 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 NET-WORK 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.

1.8.2 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.

1.8.3 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 a 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.

1.8.4 Batch (BA)

As products are developed for the batch processing industry based on the "IEC 61512-1 Batch Control – Part 1: Models and Terminology standard" there is an increasing need to exchange data between these products and other systems. Interfaces occur at all levels; with Field Management devices (e.g. monitoring stations, control stations…), Process Management systems (e.g. lab systems, batch control systems, loading, unloading, dispensing, weighing systems…), and with Business Management systems (e.g. ERP and MES). The data exchange needs to cover four basic types of information; equipment capabilities, current operating conditions, historical and recipe contents.

This specification defines interfaces for the exchange of:

- Current operating conditions with related equipment capabilities,
- Historical records of batch execution and
- Master recipe contents as well as
- Batch specific event attributes for the OPC Alarms and Events Specification.

1.8.5 Security (SEC)

OLE for Process Control has defined interfaces for Data Access Servers, Event Servers, and Historical Data Access Servers. These servers provide information that is valuable to the enterprise and if improperly updated, could have significant consequences to plant processes. Therefore, there is a need to control client access to these servers in order to protect this sensitive information and to guard against unauthorized modification of process parameters.

Security must be provided in a standard manner, consistent among implementations of OPC Servers by various vendors, to permit the implementation of portable client applications. Security must be well integrated with Windows NT and be as transparent as possible to the client application. Ideally, security should "just be there" with no special actions by the client application required in order for security to be enforced.

The purpose of this specification is to specify how OPC Servers should implement security using operating system facilities. In addition, usage guidelines are provided for the OPC Client implementation to interact with a security aware OPC Server.

1.8.6 Data eXchange (DX)

OPC Data eXchange (OPC DX) has been designed to move plant floor data horizontally between OPC DA servers. By presenting this new technology, the OPC DX enables data interoperability between OPC based systems (including DCOM and XML based systems running over Ethernet) including PLCs, HMI/SCADA, Devices, and PCs.

OPC DX, in contrast to OPC DA, is used primarily for horizontal data flows between OPC servers. OPC DX is designed to provide for the direct transfer of data from one or more OPC DA and DX servers to an OPC DX server, without the need for intermediate clients or servers to access the data from one server and forward it to another. To support these capabilities, OPC DX has the following goals:

- Exchange data between OPC DA servers in environments containing multiple bus technologies.
- Define a standard interface for OPC DX server configuration.
- Use OPC DA where practical.
- Provide an easy migration path for existing OPC DA vendors.

This specification defines abstract services used to configure horizontal data transfers from servers with OPC DA interfaces to OPC DX servers. It contains appendices that map these abstract services to specific interface technologies (e.g. Web Services and COM). OPC DX does not specify a new method for these data transfers. Instead, it relies on OPC Data Access (OPC DA) data transfer capabilities already in use today. This specification defines the behavior of the OPC DX server as it relates to control and monitoring the data transfer from DA servers to itself.

1.8.7 XML-DA

The OPC Foundation has defined interfaces to Data Access Servers, Event Servers, Batch Servers, and History Data Access Servers. These servers have information that is valuable to the enterprise, and is currently being provided to enterprise applications via OLE/COM based interfaces.

XML, the eXtensible Markup Language, and XML-based schema languages provide another means to describe and exchange structured information between collaborating applications. XML is a technology that is more readily available across a wide range of platforms. OPC XML-Data Access (OPC XML-DA) is the OPC Foundation's adoption of the XML set of technologies to facilitate the exchange of plant data across the internet, and upwards into the enterprise domain.

The purpose of this specification is to continue OPC's goal of enabling and promoting interoperability of applications. The XML-DA based interfaces will simplify sharing and exchange of OPC data amongst the various levels of the plant hierarchy (low level devices and up to enterprise systems), and to a wider range of platforms. The goal for this specification is to provide:

- Support for OPC Data Access 2.0x/3.0 data
- Support for HTTP, and SOAP
- Support for Subscription based services
- Support for a Security approach

2 Framework Concepts

This chapter describes the base concepts of the OPC Framework .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.

2.1 APIs

The classes and interfaces defined in the Framework 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.

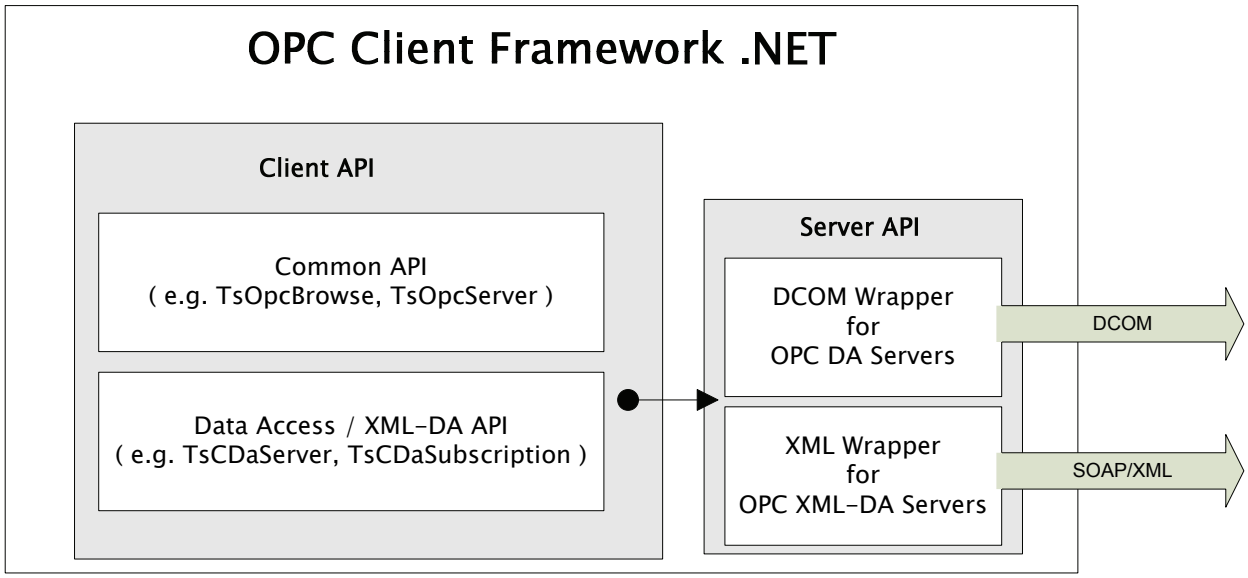
The first category called **Client API** consists of base classes that provide transparent access to all server functions but also maintain local state information such as server assigned handles for items.

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

The following diagram uses the Data Access specification as an example in order to illustrate the internal structure of the **Client API**.



2.2 Naming Conventions

The Framework API uses the following name space structure for its classes. There is one root namespace: **TsOpcNet**. The **TsOpcNet** namespace contains all classes used by more then one specification. For each specification there is another namespace which contains all classes for client and server applications used by one specification.

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

- All names 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
TsOpc	TsOpcNet	Used by more then one specification
TsDa	TsOpcNet.Da	Client and Server specific OPC Data Access and OPC XML-DA definition
TsCDa	TsOpcNet.Da	Client specific OPC Data Access and OPC XML-DA definition
TsCAe	TsOpcNet.Ae	Client specific OPC Alarms&Events definition
TsCHda	TsOpcNet.Hda	Client specific OPC Historical Data Access definition
TsCCpx	TsOpcNet.Cpx	Client specific OPC Complex Data definition
TsSXda	TsOpcNet.SXda	Server specific OPC XML-DA definition

2.3 Requirements

The system requirements of the Framework and the build applications are described in the following chapters.

2.3.1 Run-time Requirements

- Windows 2000 or Windows-XP Professional
- .NET Framework 1.1 or .NET Framework 2.0

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

2.3.1.1 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 2003.

2.3.1.2 OPC Core Components

This setup application "OPC Core Components 2.00 Redistributable 2.30.msi" installs the OPC Core components required by applications based on the OPC Client & Server Framework .NET. It can be found in the bin\redist directory.

Please be sure that you run this application after you have installed the .NET Framework. Otherwise important files are not copied.

2.3.1.3 OPC Framework .NET Components

Depending on the .NET Framework version you intend to use, one of the following components are required:

1. TsOpcNet11.v#.dll OPC Framework .NET for .NET Framework 1.1
   You can find these files in the bin or bin\strong directory. It should be copied to your application directory.
2. TsOpcNet20.v#.dll OPC Framework .NET for .NET Framework 2.0
   You can find these files in the bin20 or bin20\strong directory. It should be copied to your application directory.

2.3.1.4 OPC Framework .NET Core Components

Depending on the .NET Framework version you intend to use, one of the following set of core components are required:

1. TsOpcNetRcw*.dll OPC Framework .NET Core for .NET Framework 1.1
   You can find these files in the bin\redist directory. It should be copied to your application directory.
2. TsOpcNetRcw*.dll OPC Framework .NET Core for .NET Framework 2.0
   You can find these files in the bin20\redist directory. It should be copied to your application directory.

Please ensure that you copy the correct files depending on the .NET Framework version you use.

File	Description
TsOpcNetRcw.Comn.dll	OPC Common 1.10 .NET Wrapper
TsOpcNetRcw.Da.dll	OPC Data Access 3.00 .NET Wrapper
TsOpcNetRcw.Ae.dll	OPC Alarms & Events 1.10 .NET Wrapper
TsOpcNetRcw.Hda.dll	OPC Historial Data Access 1.10 .NET Wrapper
TsOpcNetRcw.Dx.dll	OPC Data eXchange 1.00 .NET Wrapper
TsOpcNetRcw.Batch.dll	OPC Batch Custom 2.00 .NET Wrapper
TsOpcNetRcw.Sec.dll	OPC Security 1.00 .NET Wrapper.

2.3.2 Development Requirements

For development one of the following compilers are supported

- Microsoft Visual Studio .NET 2003 or
- Microsoft Visual Studio 2005 or
- Borland C# Builder or
- Borland Delphi 8.0

2.4 Server Identification

The Framework API uses two mechanisms to identify a server:

- The servername, e.g. “TsOpcSource.DataSample.4”, plus the machine name/hostname, e.g. “R34”, identifies a server. This mechanism is only available for the COM/DCOM based specifications.
- 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 Framework API defines a URL syntax for COM servers which has 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
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 indentifier 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/TsOpcSource.DataSample.4
```

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//TsOpcXmlDaSampleServer/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.

2.5 Server Browsing

The Framework API has a single discover interface (ITsOpcDiscovery) that can be used for all protocols, however, the mechanism used to discover servers is highly dependent on the protocol; as a result, the Framework API provides the class TsOpcBrowse which implements different functions for browsing COM based servers for the different 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 Framework API implementation automatically connects to the COM Server Enumerator services on remote machines as necessary.

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

The class TsOpcBrowse provides functions just returning lists of available servers for a specification, e.g. *ListDAServers.*

An example printing out all OPC DA 1.0 / DA 2.0 and DA 3.0 servers on the local computer is:

```
TsOpcBrowse listDaServer = new TsOpcBrowse();
TsOpcResult res;
TsOpcServerInfo[] si;

res = listDaServer.ListDaServers(out si);
if (res.IsSuccess())
{
       for (int i = si.GetLowerBound(0); i <= si.GetUpperBound(0); i++)
       {
              Console.WriteLine(si[i].ServerName);
       }
}
```

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

An example using the ITsOpcDiscovery interface getting TsOpcServer objects for all DA 3.0 servers on the local computer is:

```
TsOpcSpecification[] specifications = new TsOpcSpecification[1];
ITsOpcDiscovery discovery = null;

specifications[0]  = TsOpcSpecification.OPC_DA_30;

TsOpcServer[] servers = discovery.GetAvailableServers(specifications);
```

2.6 Server Connections

2.6.1 General

The Framework API requires that clients first instantiate a sub-class of TsOpcServer (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 TsOpcFactory that is passed to the TsOpcServer object constructor. Any implementation of the ITsOpcDiscovery interface should return a set of TsOpcServer objects that contains a reference to the appropriate class factory.

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 a in-process COM wrapper object.

The actual event of creating an instance of the remote server object occurs when the *Connect* method on the TsOpcServer object is called. This design allows the TsOpcServer 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 TsOpcFactory object passed to the constructor of the TsOpcServer object.

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

```
const string serverName = "TsOpcSource.DASample.40";
const string ipAddress = "localhost";

try
{
       TsOpcComputerInfo    host = new TsOpcComputerInfo(ipAddress);
       TsCDaServer          myDaServer = new TsCDaServer();

       myDaServer.Connect( serverName, host );

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

2.6.2 Security

The client application passes user authentication information in the *Credentials* property of the TsOpcConnectData class. This property contains the user name, password and domain for Windows account that the Framework API will use to authenticate the client when connecting to the server. The Framework API only supports integrated Windows authentication.

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

```
const string serverName = "TsOpcSource.DASample.40";
const string computerName = "R40";
TsOpcNet.URL url;

try
{
       url = new TsOpcNet.URL("opcda://"+computerName+"/"+serverName);
       System.Net.NetworkCredential credentials =
              new System.Net.NetworkCredential("username","password");
       WebProxy proxy  = null;
       TsOpcNet.Com.Factory factory = new TsOpcNet.Com.Factory();
       TsOpcComputerInfo    host = new TsOpcComputerInfo(ipAddress);
       TsCDaServer myDaServer = new TsCDaServer(factory, url);

       myDaServer.Connect( url, new TsOpcConnectData(credentials, proxy));

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

2.6.3 HTTP Proxy

The client application may override the default HTTP proxy server configuration used by the Framework API with the *SetProxy* method on the TsOpcConnectData class. The MSDN documentation for the *WebProxy* class has more information on HTTP proxy server configuration parameters

2.6.4 Error Handling

2.6.4.1 TsOpcResult

The TsOpcResult 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. A TsOpcResult is uniquely identified by an XML qualified name and a 32 bit integer. A TsOpcResult 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 framework 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 TsOpcResult 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.

2.6.4.2 Exceptions

The framework 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 framework will map that error onto a TsOpcResult and throw a TsOpcResultException.

Other types of exceptions, such as ArgumentNullException, may be thrown by the framework when appropriate.

2.6.5 Garbage Collection

Many of the interfaces and classes defined in the Framework 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 Framework 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.

2.6.6 Item Identifiers

The Framework API uses the TsOpcItem 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 create 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.

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.

The documentation indicates which of the fields are required for any particular method call and what values will be in the response. The Framework API allows client applications to control whether the *ItemName*, *ItemPath* and *ClientHandle* are returned with the results to a request. Server API

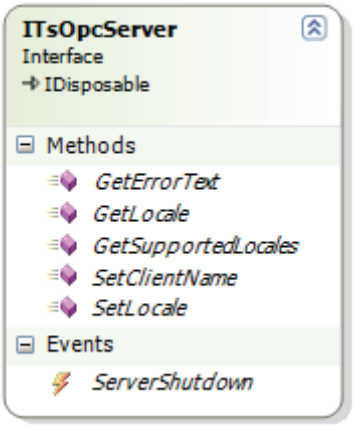
2.7 Server API

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.

2.7.1 Interfaces

2.7.1.1 ITsOpcServer 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 ITsOpcServer interface is the base for the following sub-classes:

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

2.7.1.1.1 Methods

The ITsOpcServer 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

<table>
<tr><td></td><td>
<table>
<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>
<tr><td>GetLocale</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>
<tr><td>GetSupportedLocales</td><td>Returns the locales supported by the server.
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>An array of locales with the format "[language code]-[country/region code]".</td></tr>
</table>
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:
<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>
<tr><td>SetClientName</td><td>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:
<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>

<table>
<tr><td>SetLocale</td><td>Sets 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>locale</td><td>The locale name in the format "[language code]-[country/region code]".</td></tr>
<tr><td>[Return Value]</td><td>A locale that the server supports and is the best match for the requested locale.</td></tr>
</table>
If the server does not support the requested locale, the server must find a best match by ignoring the country/region code and find a locale with the same language. If the server does not support any locales with the requested language then it chooses its default locale.

The client may specify a blank locale (“”) which is also known as the ‘invariant’ locale. The server must always map this locale onto the a real locale that represents the default for the server such as “en-US” (English-US) or de-DE (German-Germany). 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>

2.7.1.1.2 Events

The ITsOpcServer interface has the following events:

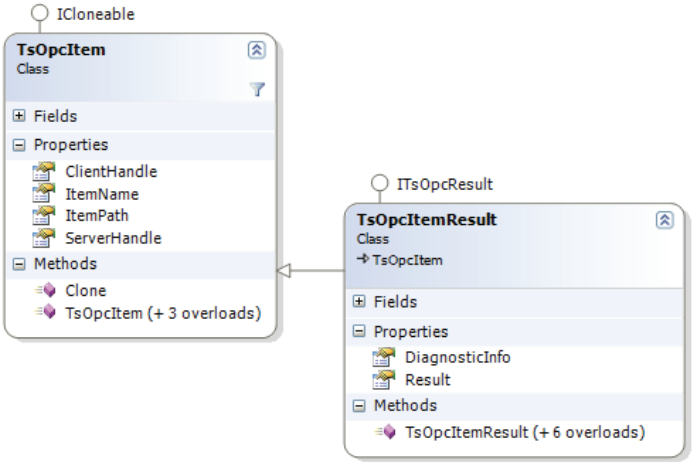
Name	Description
ServerShutdown	An event to receive server shutdown notifications.

2.7.2 Classes

2.7.2.1 TsOpcItem 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 TsOpcItem 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.



2.7.2.1.1 Properties

The TsOpcItem 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 TsOpcItemResult 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.

2.8 Client API

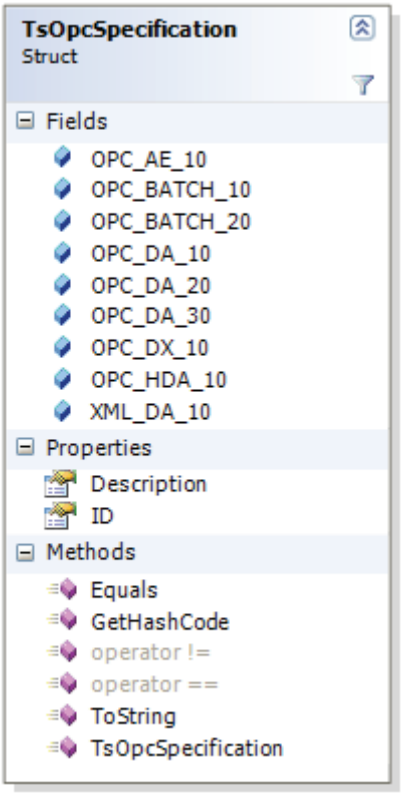
The **Client API** consists of a set of class that always resides inside the client process. These classes provide access to the remote servers and maintain client side state information. The main objective of the **Client API** is to provide a convenient, easy to use .NET programming interface for .NET 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.

2.8.1 Structures

2.8.1.1 TsOpcSpecification Structure

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



2.8.1.1.1 Properties

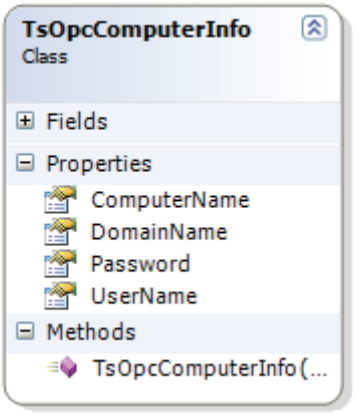
The TsOpcSpecification structure has the following fields:

Name	Description
OPC_AE_10	OPC Alarms&Events 1.0 OPC Alarms&Events 1.1.
OPC_BATCH_10	OPC Batch 1.0
OPC_BATCH_20	OPC Batch 2.0
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_DX_10	OPC Data Exchange 1.0.
OPC_HDA_10	OPC Historical Data Access 1.0.
XML_DA_10	OPC XML-DA 1.0.

2.8.2 Classes

2.8.2.1 TsOpcComputerInfo Class

This class store computer information like name of computer and is be used for browsing of OPC servers and connecting to OPC servers.



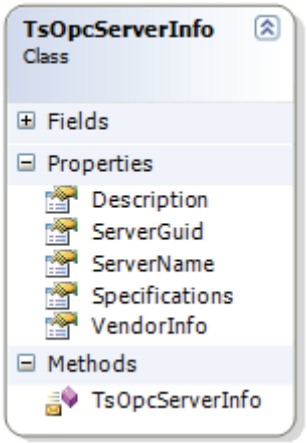
2.8.2.1.1 Properties

The TsOpcComputerInfo class has the following properties:

Name	Description
ComputerName	The Computer Name to use.
DomainName	The Domain Name to use.
Password	The Password to use.
UserName	The Username to use.

2.8.2.2 TsOpcServerInfo Class

This class store OPC server information like name of computer and is returned by the server browsing functions.



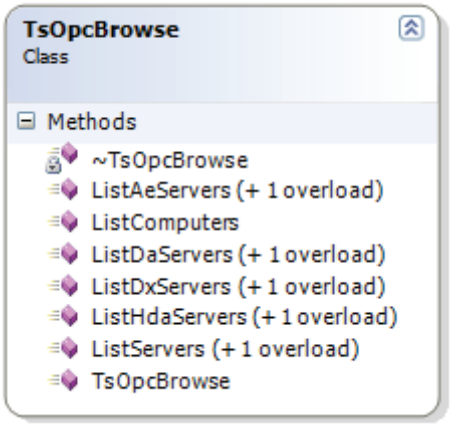
2.8.2.2.1 Properties

The TsOpcServerInfo class has the following properties:

Name	Description
Description	A description about the OPC Server
ServerGuid	Globally Unique Identifier (GUID) of the OPC Server
ServerName	The OPC Server name
Specifications	Supported specifications by the OPC Server as TsOpcSpecification structure.
VendorInfo	Vendor specific information about the OPC Server

2.8.2.3 TsOpcBrowse Class

This class provides functions to browse the computers within a network and the installed OPC servers on the local machine or the specified computer.



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 Framework API implementation automatically connects to the COM Server Enumerator services on remote machines as necessary.

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

The class TsOpcBrowse provides functions just returning lists of available servers for a specification, e.g. *ListDAServers.*

2.8.2.3.1 Methods

The TsOpcBrowse class has the following methods:

Name	Description
ListAeServers	Get the names of the registered OPC AE 1.00 and AE 1.10 servers. The ProgId of all OPC Alarms & servers is returned as a string array. This array can e.g. be directly displayed in a Windows control. This method has the following parameters: **Name** / **Description** serverInfos: OPC Server Information as an array of TsOpcServerInfo objects. computerInfo: Remote Computer Information as an TsOpcComputerInfo object. [Return Value]: A TsOpcResult object with the result of the operation. Possible return codes are: **Name** / **Description** E_FAIL: The operation failed. REGDB_E_CLASSNOTREG: Unable to create an instance of the Component Categories Manager on the remote machine. REGDB_E_READREGDB: There was an error reading the registry. OLE_E_REGDB_KEY: The ProgID = MainUserTypeName or CLSID = MainUserTypeName keys are missing from the registry. E_INVALIDARG: One or more arguments are incorrect. E_OUTOFMEMORY: Insufficient memory to create and return an enumerator object. S_OK: The operation succeeded.
ListComputers	Enumerates hosts that may be accessed for server discovery. This method has the following parameters: **Name** / **Description**

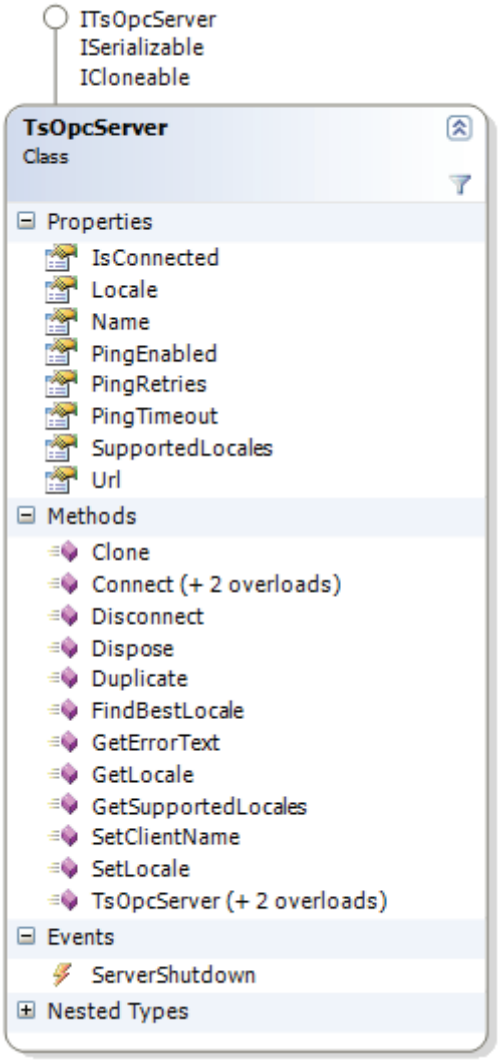
	computers: Names of found computers as string array. [Return Value]: A TsOpcResult object with the result of the operation. Possible return codes are: ERROR_NO_BROWSER_SERVERS_FOUND: No browser servers found. S_OK: The operation succeeded.
ListDaServers	Get the names of the registered OPC DA 1.0 / DA 2.0 and DA 3.0 servers. The ProgId of all OPC Data Access is returned as a string array. This array can e.g. be directly displayed in a Windows control. This method has the following parameters: serverInfos: OPC Server Information as an array of TsOpcServerInfo objects. computerInfo: Remote Computer Information as an TsOpcComputerInfo object. [Return Value]: A TsOpcResult object with the result of the operation. Possible return codes are: E_FAIL: The operation failed. REGDB_E_CLASSNOTREG: Unable to create an instance of the Component Categories Manager on the remote machine. REGDB_E_READREGDB: There was an error reading the registry. OLE_E_REGDB_KEY: The ProgID = MainUserTypeName or CLSID = MainUserTypeName keys are missing from the registry. E_INVALIDARG: One or more arguments are incorrect. E_OUTOFMEMORY: Insufficient memory to create and return an enumerator object. S_OK: The operation succeeded.
ListDxServers	Get the names of the registered OPC DX 1.00 servers. The ProgId of all OPC Data eXchange servers is returned as a string array. This array can e.g. be directly displayed in a Windows control. This method has the following parameters: serverInfos: OPC Server Information as an array of TsOpcServerInfo objects. computerInfo: Remote Computer Information as an TsOpcComputerInfo object. [Return Value]: A TsOpcResult object with the result of the operation. Possible return codes are: E_FAIL: The operation failed. REGDB_E_CLASSNOTREG: Unable to create an instance of the Component Categories Manager on the remote machine. REGDB_E_READREGDB: There was an error reading the registry.

		OLE_E_REGDB_KEY	The ProgID = MainUserTypeName or CLSID = MainUserTypeName keys are missing from the registry.
		E_INVALIDARG	One or more arguments are incorrect.
		E_OUTOFMEMORY	Insufficient memory to create and return an enumerator object.
		S_OK	The operation succeeded.
ListHdaServers	Get the names of the registered OPC HDA 1.00 and HDA 1.10 servers. The ProgId of all OPC Historical Data Access servers is returned as a string array. This array can e.g. be directly displayed in a Windows control. This method has the following parameters:		
		Name	**Description**
		serverInfos	OPC Server Information as an array of TsOpcServerInfo objects.
		computerInfo	Remote Computer Information as an TsOpcComputerInfo object.
		[Return Value]	A TsOpcResult object with the result of the operation.
	Possible return codes are:		
		Name	**Description**
		E_FAIL	The operation failed.
		REGDB_E_CLASSNOTREG	Unable to create an instance of the Component Categories Manager on the remote machine.
		REGDB_E_READREGDB	There was an error reading the registry.
		OLE_E_REGDB_KEY	The ProgID = MainUserTypeName or CLSID = MainUserTypeName keys are missing from the registry.
		E_INVALIDARG	One or more arguments are incorrect.
		E_OUTOFMEMORY	Insufficient memory to create and return an enumerator object.
		S_OK	The operation succeeded.
ListServers	Get the names of the OPC servers with one of the specifications listed in the Specifications parameter. The ProgId of all OPC Servers is returned as a string array. This array can e.g. be directly displayed in a Windows control. This method has the following parameters:		
		Name	**Description**
		specifications	Array with the TsOpcSpecification object of each specification to include.
		serverInfos	OPC Server Information as an array of TsOpcServerInfo objects.
		computerInfo	Remote Computer Information as an TsOpcComputerInfo object.
		[Return Value]	A TsOpcResult object with the result of the operation.
	Possible return codes are:		
		Name	**Description**
		E_FAIL	The operation failed.

		REGDB_E_CLASSNOTREG	Unable to create an instance of the Component Categories Manager on the remote machine.
		REGDB_E_READREGDB	There was an error reading the registry.
		OLE_E_REGDB_KEY	The ProgID = MainUserTypeName or CLSID = MainUserTypeName keys are missing from the registry.
		E_INVALIDARG	One or more arguments are incorrect.
		E_OUTOFMEMORY	Insufficient memory to create and return an enumerator object.
		S_OK	The operation succeeded.

2.8.2.4 TsOpcServer 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 ITsOpcServer 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 Framework 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 TsOpcServer 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 or OPC XML-DA 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

2.8.2.4.1 Properties

The TsOpcServer class has the following properties:

Name	Description
IsConnected	Whether the remote server is currently connected.
Locale	The default of locale used by the remote server.
Name	A short descriptive name for the server assigned by the client.
PingEnabled	Defines whether the ping mechanism is enabled or disabled.
PingRetries	Defines if the number of ping retries. Default is 3.
PingTimeout	Defines if the timeout for one ping. Default is 1 second.
SupportedLocales	The set of locales supported by the remote server.
Url	The URL that describes the network location of the server.

2.8.2.4.2 Methods

The TsOpcServer class has the following methods:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Clone</td><td>Returns an unconnected copy of the server with the same URL.</td></tr>
<tr><td>Connect</td><td>Establishes a physical connection to the remote server.
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>url</td><td>The network address for the remote server.
It replaces the default URL for the server if the method succeeds.</td></tr>
<tr><td>connectData</td><td>Any protocol configuration or user authentication information.</td></tr>
</table></td></tr>
<tr><td>Disconnect</td><td>Disconnects from the server and releases all network resources. Also Dispose() is called.</td></tr>
<tr><td>Dispose</td><td>This must be called explicitly by clients to ensure the remote server is released. Is automatically called by the Disconnect method.</td></tr>
<tr><td>Duplicate</td><td>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:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>A duplicate of the server object.</td></tr>
</table></td></tr>
<tr><td>FindBestLocale</td><td>Finds the best matching locale given a set of supported locales.
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.</td></tr>
<tr><td>GetErrorText
(ITsOpcServer)</td><td>Returns the localized text for the specified result code.
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>locale</td><td>The locale name in the format "[language code]-[country/region code]".</td></tr>
<tr><td>resultID</td><td>The result code identifier.</td></tr>
<tr><td>[Return Value]</td><td>A message localized for the locale that is the best match for the requested locale.</td></tr>
</table>
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</td></tr>
</table>

<table>
<tr><td></td><td>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:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>E_FAIL</td><td>The operation failed.</td></tr>
<tr><td>E_OUTOFMEMORY</td><td>Not enough memory</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>
<tr><td>GetLocale
(ITsOpcServer)</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>
<tr><td>GetSupportedLocales
(ITsOpcServer)</td><td>Returns the locales supported by the server
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>An array of locales with the format "[language code]-[country/region code]".</td></tr>
</table>
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:
<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>
<tr><td>SetClientName
(ITsOpcServer)</td><td>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:
<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>

<table>
<tr>
<td>SetLocale
(ITsOpcServer)</td>
<td>Sets 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>locale</td><td>The locale name in the format "[language code]-[country/region code]".</td></tr>
<tr><td>[Return Value]</td><td>A locale that the server supports and is the best match for the requested locale.</td></tr>
</table>
If the server does not support the requested locale, the server must find a best match by ignoring the country/region code and find a locale with the same language. If the server does not support any locales with the requested language then it chooses its default locale.

The client may specify a blank locale (“”) which is also known as the ‘invariant’ locale. The server must always map this locale onto the a real locale that represents the default for the server such as “en-US” (English-US) or de-DE (German-Germany). 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>

2.8.2.4.3 Events

The TsOpcServer class has the following events:

Name	Description
ServerShutdown (ITsOpcServer)	An event to receive server shutdown notifications.

3 OPC DA/XML-DA Client Development

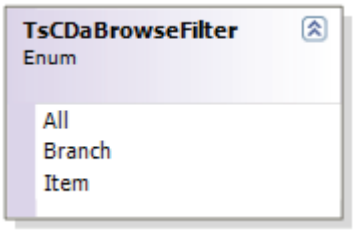
This chapter describes the OPC Data Access and OPC XML-DA specific classes of the **Server API** and **Client API.** Knowledge of the corresponding specifications is required for the understanding of this chapter.

3.1 Server API

3.1.1 Enumerations

3.1.1.1 TsCDaBrowseFilter enumeration

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



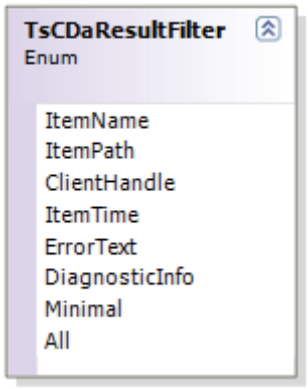
3.1.1.1.1 Properties

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.

3.1.1.2 TsCDaResultFilter enumeration

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



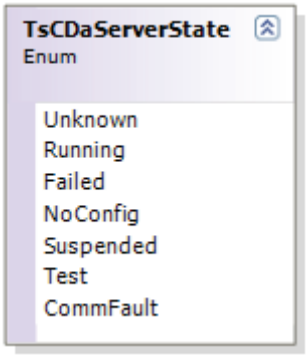
3.1.1.2.1 Properties

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 ItemIden-tifier 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.

3.1.1.3 TsCDaServerState enumeration

This enumeration defines the set of possible server states.



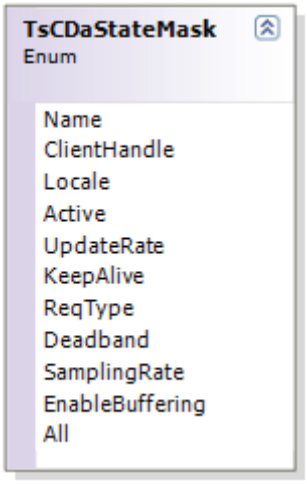
3.1.1.3.1 Properties

The TsCDaServerState enumeration has the following properties:

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.

3.1.1.4 TsCDaStateMask enumeration

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



3.1.1.4.1 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.

3.1.1.5 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.

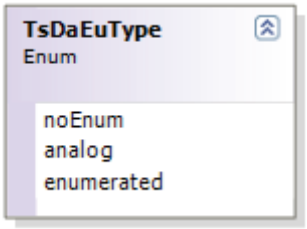
3.1.1.5.1 Properties

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.

3.1.1.6 TsDaEuType enumeration

This enumeration defines the possible item engineering unit types.



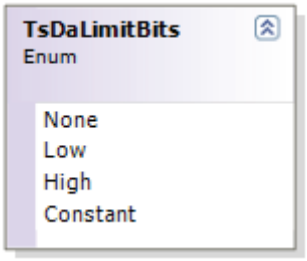
3.1.1.6.1 Properties

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.)

3.1.1.7 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.

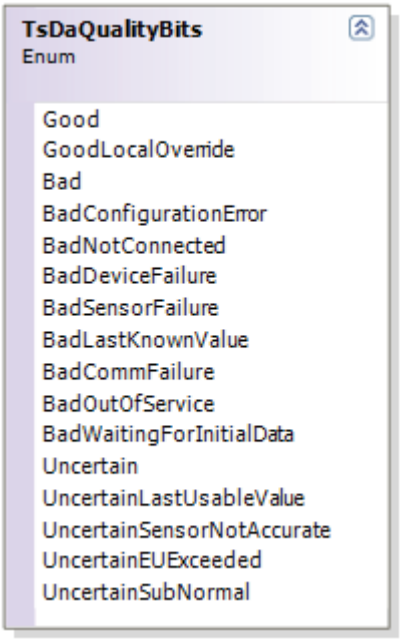
3.1.1.7.1 Properties

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

3.1.1.8 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.

3.1.1.8.1 Properties

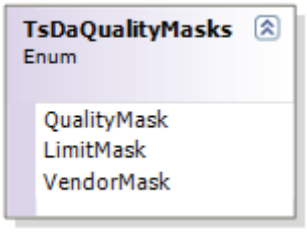
The TsDaQualityBits enumeration has the following properties:

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.

3.1.1.9 TsDaQualityMasks enumeration

This enumeration defines bit masks for the quality.



3.1.1.9.1 Properties

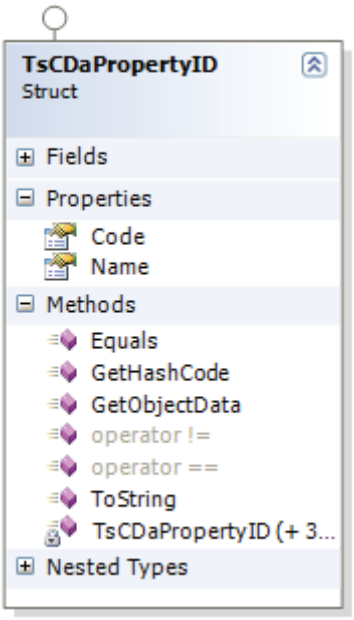
The TsDaQualityMasks enumeration has the following properties:

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

3.1.2 Structures

3.1.2.1 TsCDaPropertyID Structure

This structure contains a unique identifier for a property.



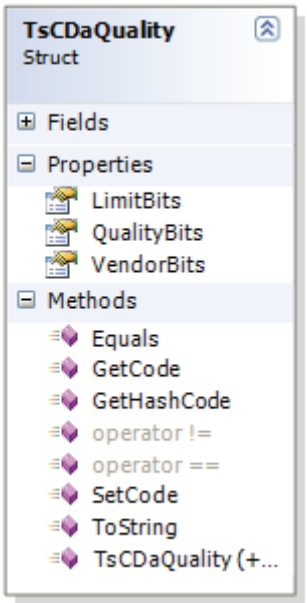
3.1.2.1.1 Properties

The TsCDaPropertyID structure has the following fields:

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

3.1.2.2 TsCDaQuality Structure

This structure contains the quality field for an item value.



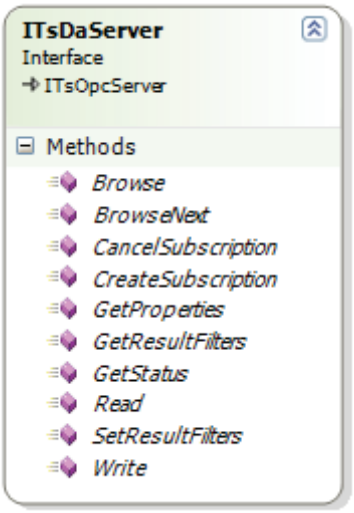
3.1.2.2.1 Properties

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.

3.1.3 Interfaces

3.1.3.1 ITsDaServer Interface



This interface defines functionality specific to OPC Data Access and XML-DA servers.

This interface inherits from the ITsOpcServer interface.

This interface is intended to merge the functionality defined by the OPC DA and OPC XML-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 and OPC XML-DA servers

3.1.3.1.1 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 3.1.4.4.
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* op-

			erations. 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 TsOpcResultException 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 TsOpcItem if bit is set.
ItemPath	0x02	Include the ItemPath in the TsOpcItem if bit is set.
ClientHandle	0x04	Include the ClientHandle in the TsOpcItem 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 TsOpcItem has no meaning when used at the server level.

The filters only affect results returned from the Read and Write methods. They are

<table>
<tr><td></td><td>also used as the default for new Subscriptions.</td></tr>
<tr><td>GetStatus</td><td>This method returns the current status of the server.

This method has the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>The current server status.</td></tr>
</table>
The server status is an object that has the following properties:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>VendorInfo</td><td>The vendor name and product name for the server.</td></tr>
<tr><td>ProductVersion</td><td>The vendor’s software version number.</td></tr>
<tr><td>ServerState</td><td>The current server state (see the server state enumeration below).</td></tr>
<tr><td>StatusInfo</td><td>More information about the current server state.</td></tr>
<tr><td>StartTime</td><td>The UTC time when the server started.</td></tr>
<tr><td>CurrentTime</td><td>The current UTC time at the server.</td></tr>
<tr><td>LastUpdateTime</td><td>The last time the server sent a data update to the client.</td></tr>
</table>
The server state enumeration has the following values:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Unknown</td><td>The server state is not known.</td></tr>
<tr><td>Running</td><td>The server is running normally.</td></tr>
<tr><td>Failed</td><td>The server is not functioning due to a fatal error.</td></tr>
<tr><td>NoConfig</td><td>The server cannot load its configuration information.</td></tr>
<tr><td>Suspended</td><td>The server has halted all communication with the underlying hardware.</td></tr>
<tr><td>Test</td><td>The server is disconnected from the underlying hardware.</td></tr>
<tr><td>CommFault</td><td>The server cannot communicate with the underlying hardware.</td></tr>
</table></td></tr>
<tr><td>Read</td><td>The method reads the current values for a set of items.

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 read.
Each item must have an ItemName
Each item may have an ItemPath, a ReqType or MaxAge.</td></tr>
<tr><td>[Return Value]</td><td>The results of the read operation for each item.</td></tr>
</table>
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.</td></tr>
</table>

<table>
<tr><td></td><td>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.</td></tr>
<tr><td>SetResultFilters</td><td>This method sets the filters applied by the server to any item results returned to the client.

This method has the following parameters:

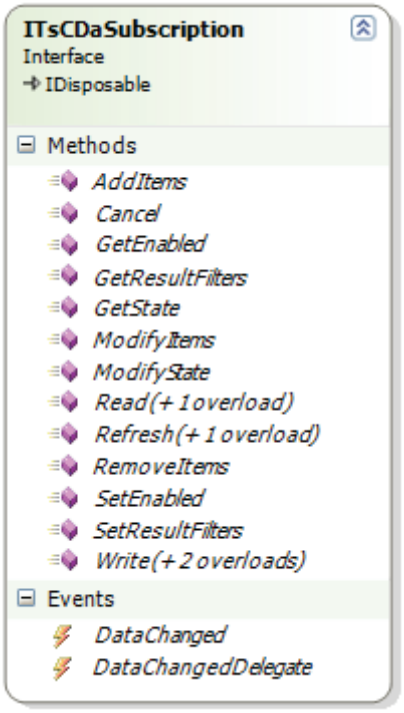
<table><tr><th>Name</th><th>Description</th></tr><tr><td>filters</td><td>A bit mask indicating which fields should be returned in any item results.</td></tr></table></td></tr>
<tr><td>Write</td><td>This method writes the value, quality and timestamp for a set of items.

This method has the following parameters:

<table><tr><th>Name</th><th>Description</th></tr><tr><td>itemValues</td><td>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.</td></tr><tr><td>[Return Value]</td><td>The results of the write operation for each item.</td></tr></table>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.</td></tr>
</table>

3.1.3.2 ITsCDaSubscription Interface



This interface allows clients to control subscriptions to data items exposed by a Data Access and/or XML-DA server.

The Framework subscription mechanism unifies the group/subscription mechanisms from COM-DA and XML-DA. The Framework API implements a fully asynchronous callback mechanism for XML-DA servers via a client side in-process wrapper.

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 and XML-DA servers.

3.1.3.2.1 Methods

The ITsDaSubscription 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 3.1.4.1. 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.
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 3.1.3.1. 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.

<table>
<tr><td>ModifyItems</td><td>This method modifies the state of items in the subscription.
This method has the following parameters:<table><tr><th>Name</th><th>Description</th></tr><tr><td>masks</td><td>A bit mask indicating which item state parameters are being modified.</td></tr><tr><td>items</td><td>The new state for the items being modified.
The TsCDaItem object is described in Section 3.1.4.1.
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.</td></tr><tr><td>[Return Value]</td><td>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.</td></tr></table></td></tr>
<tr><td>ModifyState</td><td>This method changes the state of a subscription.
This method has the following parameters:<table><tr><th>Name</th><th>Description</th></tr><tr><td>masks</td><td>A bit mask that indicates which elements of the subscription state are changing.</td></tr><tr><td>state</td><td>The new subscription state.
The TsCDaSubscriptionState object is described in Section 3.1.4.8.
The TsCDaStateMask enumeration contains the bit masks used to indicate which properties of the TsCDaSubscriptionState object contain valid information.</td></tr><tr><td>[Return Value]</td><td>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.</td></tr></table></td></tr>
<tr><td>Read</td><td>Read(TsCDaItem[] items)
This method reads the values for a set of items in the subscription.
This method has the following parameters:</td></tr>
</table>

		Name	**Description**
		items	The set of items to read. The TsCDaItem object is described in Section 3.1.4.1. 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 TsOpcItemResult Object.
Read	**Read(TsCDaItem[] items, object requestHandle, TsCDaReadCompleteHandler callback, ITsOpcRequest 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 3.1.4.1. 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 TsOpcItemResult 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.

<table>
<tr><td>Refresh</td><td>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.</td></tr>
<tr><td>Refresh</td><td>Refresh(object requestHandle, ITsOpcRequest 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:

<table><tr><th>Name</th><th>Description</th></tr><tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr><tr><td>request</td><td>An identifier for the request (may be used to cancel the request).</td></tr></table></td></tr>
<tr><td>RemoveItems</td><td>This method modifies the state of items in the subscription.

This method has the following parameters:

<table><tr><th>Name</th><th>Description</th></tr><tr><td>items</td><td>The identifiers (i.e. server handles) for the items being removed.
The TsOpcItem object is described in Section 2.7.2.1.
Each item must have the ServerHandle specified.</td></tr><tr><td>[Return Value]</td><td>An array of TsOpcItemResult containing the results of the remove item operation for each item.</td></tr></table></td></tr>
</table>

<table>
<tr><td>SetEnabled</td><td>This method enables or disables data change notifications from the server.
This method takes the following parameters:<table><tr><th>Name</th><th>Description</th></tr><tr><td>enabled</td><td>Whether data changed notifications should be sent.</td></tr></table></td></tr>
<tr><td>SetResultFilters</td><td>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 3.1.3.1.
The filters specified at the subscription level override filters specified at the server level.</td></tr>
<tr><td>Write</td><td>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:<table><tr><th>Name</th><th>Description</th></tr><tr><td>items</td><td>The set of item values to write.
The TsCDaItemValue object is described in Section 3.1.4.2.
Each item must have its ServerHandle and Value specified.
Each item may have a Quality and/or a Timestamp specified.</td></tr><tr><td>[Return Value]</td><td>The results of the write operation for each item as TsOpcItemResult Object.</td></tr></table></td></tr>
</table>

<table>
<tr><td>Write</td><td>
Write(TsCDaItemValue[] items, object requestHandle,
TsCDaWriteCompleteHandler callback, ITsOpcRequest 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:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>items</td><td>The set of items to write.
The TsCDaItemValue object is described in Section 3.1.4.2.
Each item must have its ServerHandle specified.
Each item may have the ReqType and/or MaxAge specified.</td></tr>
<tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr>
<tr><td>callback</td><td>The function to invoke when the request completes.
The TsCDaWriteCompleteHandler delegate is described below.</td></tr>
<tr><td>request</td><td>An identifier for the request (may be used to cancel the request).</td></tr>
<tr><td>[Return Value]</td><td>An array of TsOpcItemResult containing any errors encountered when the server validated the items.</td></tr>
</table>
The TsCDaWriteCompleteHandler delegate has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr>
<tr><td>results</td><td>The results of the write operation for each item as TsOpcItemResult array.
The item results always contain the ClientHandle.</td></tr>
</table>
</td></tr>
</table>

3.1.3.2.2 Events

The ITsDaSubscription interface has the following events:

Name	Description
DataChanged	An event to receive data change updates.

3.1.4 Classes

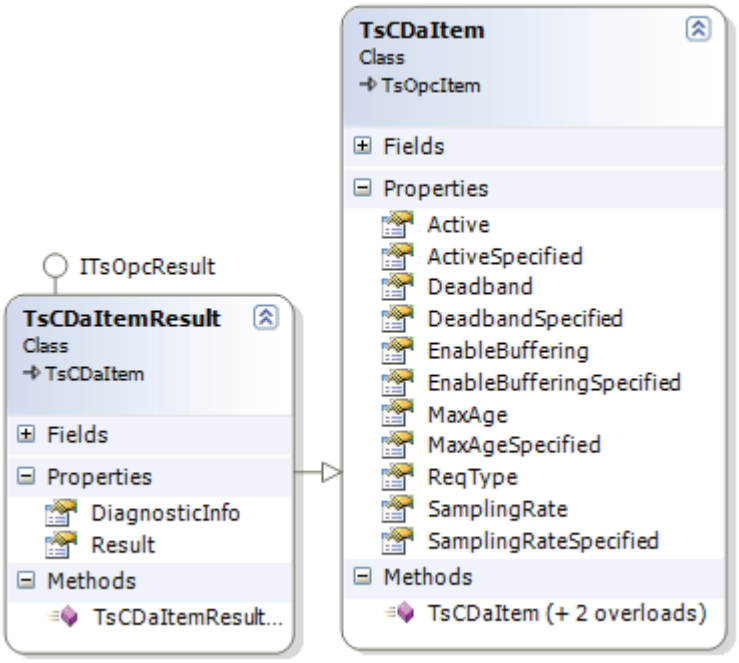
3.1.4.1 TsCDaItem class

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.

3.1.4.1.1 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) > (pPercent-Deadband/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 and OPC XML-DA servers!**
EnableBufferingSpecified	Whether the Enable Buffering is specified. **Only supported for OPC DA 3.0 and OPC XML-DA 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 framework will issue a read from cache. If MaxAgeSpecified is true the framework 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 / OPC XML-DA:** 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. 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 group update rate. Multiple values for the same item must be in chronological order within the callback array. In other words, if the group has an update rate of 10 seconds and there is an item within the group 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 group. In the case where an item has a different update rate than the group, 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 group update rate and the value and/or quality change more often than the group 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 group. 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 group update rate, then the item will only be collected from the underlying device at the sampling rate, as opposed to the group update rate. **Only supported for OPC DA 3.0 and OPC XML-DA servers!**
SamplingRateSpecified	Whether the Sampling Rate is specified. **Only supported for OPC DA 3.0 and OPC XML-DA servers!**

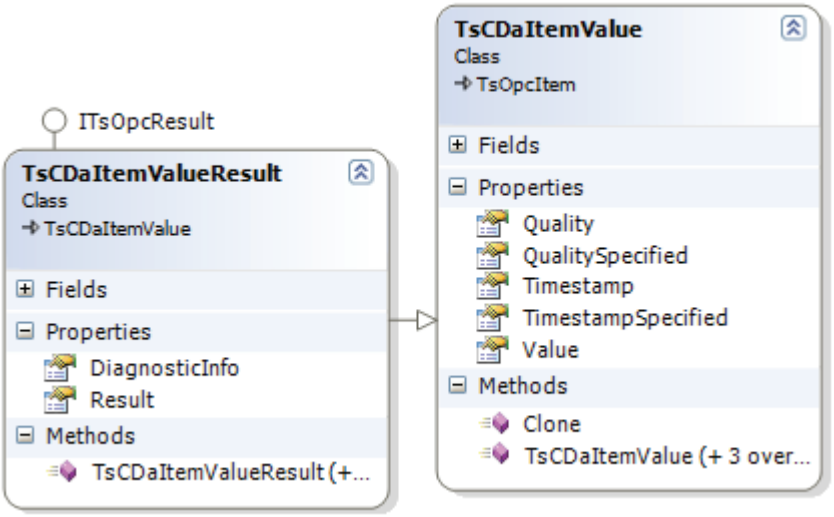
The TsCDaItemResult class extends the TsCDaItem class by adding the following properties:

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.

3.1.4.2 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.



3.1.4.2.1 Properties

The TsCDaItemValue class has the following properties:

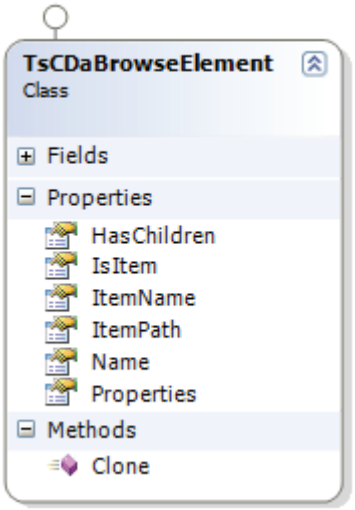
Name	Description
Quality	The quality of the item value.
QualitySpecified	Whether the quality is specified.
Timestamp	The timestamp for the item value.
TimestampSpecified	Whether the timestamp is specified.
Value	The item value.

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.

3.1.4.3 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.

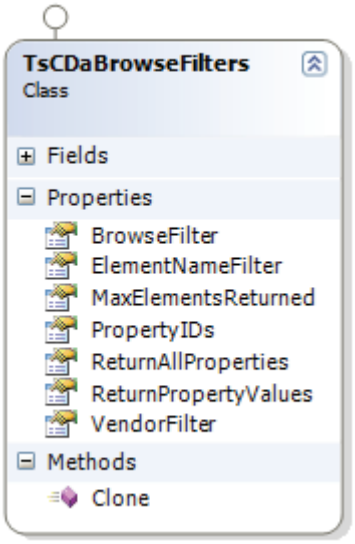
3.1.4.3.1 Properties

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.

3.1.4.4 TsCDaBrowseFilters class

This class defines a set of filters to apply when browsing



3.1.4.4.1 Properties

The TsCDaBrowseFilters object has the following properties:

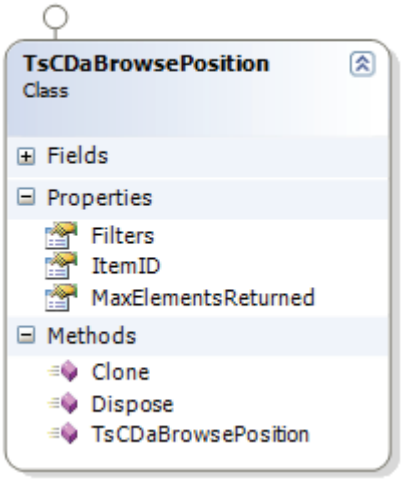
Name	Description
MaxElementsReturned	The maximum number of elements to return. Zero means no limit.
BrowseFilter	The type of elements to return (Branch, Item or All).
ElementNameFilter	An expression used to match the name of the element.
VendorFilter	A filter which has semantics that defined by the server.
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.

3.1.4.5 TsCDaBrowsePosition class

This class stores the state of a browse operation.

3.1.4.5.1 Properties

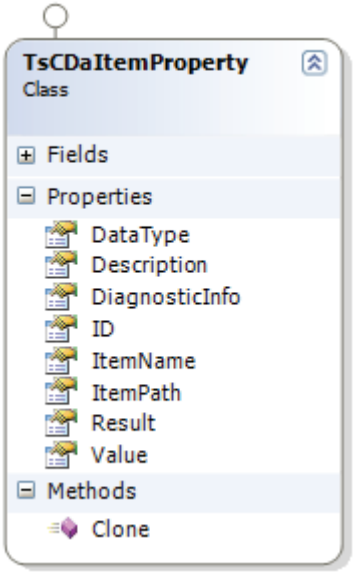
The TsCDaBrowsePosition object has the following properties:



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.

3.1.4.6 TsCDaItemProperty class

This class describes a property of an item.



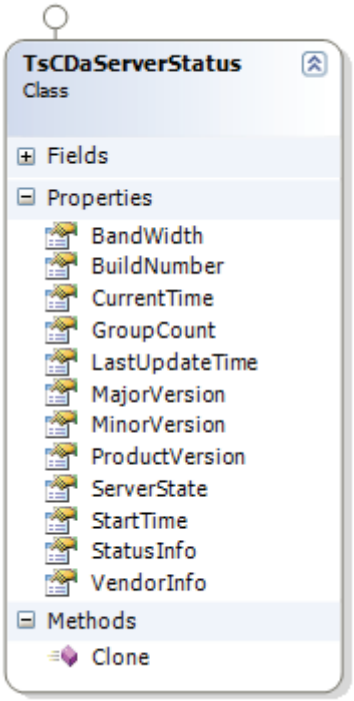
3.1.4.6.1 Properties

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 TsOpcResult object with the result of an operation on an property.
Value	The value of the property.

3.1.4.7 TsCDaServerStatus class

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



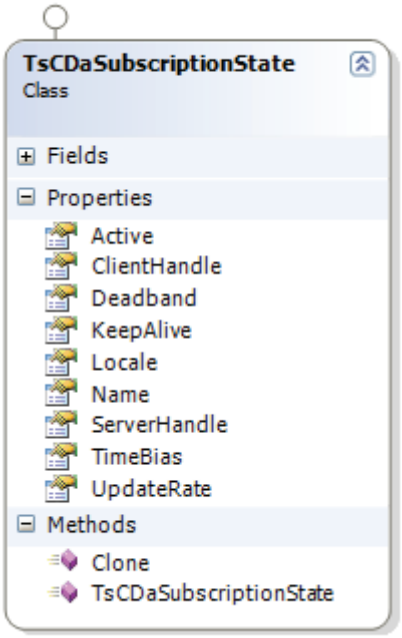
3.1.4.7.1 Properties

The TsCDaServer 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
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 TsCDaServerState values in Section 3.1.1.3.
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.

3.1.4.8 TsCDaSubscriptionState class

This class describes the state of a subscription.



3.1.4.8.1 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.

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.

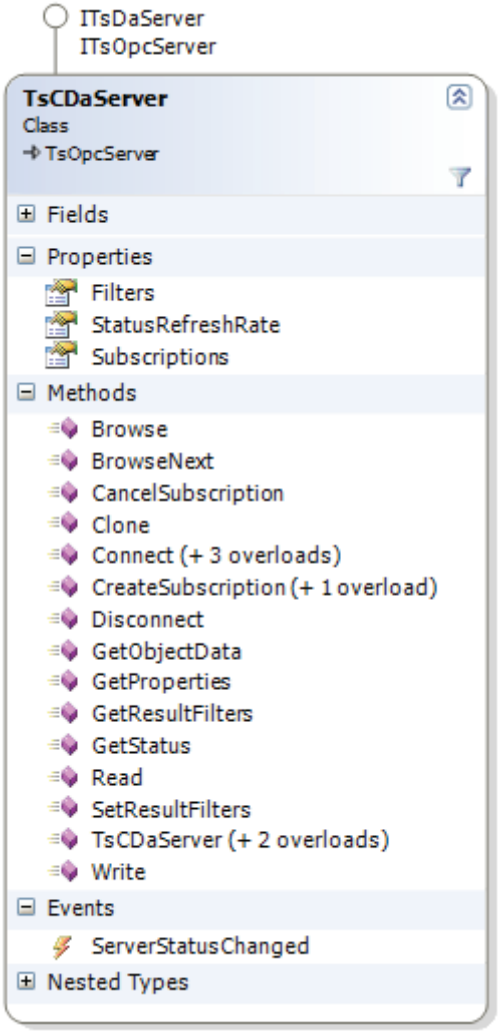
3.2 Client API

3.2.1 Classes

3.2.1.1 TsCDaServer class

This class is a base class for an in-process object used to access OPC Data Access and OPC XML-DA servers and is an in-process wrapper for a remote server (i.e. a server that implements the ITsOpcServer 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.



3.2.1.1.1 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.

3.2.1.1.2 Methods

The TsCDaServer class has the following methods:

<table>
<tr><td>Browse</td><td>This method fetches the children of a branch that meet the filter criteria.

This method has the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>itemID</td><td>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).</td></tr>
<tr><td>filters</td><td>The filters to use to limit the set of child elements returned.
The TsCDaBrowseFilters object is described in section 3.1.4.4.</td></tr>
<tr><td>position</td><td>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.</td></tr>
<tr><td>[Return Value]</td><td>The set of elements found.</td></tr>
</table>
</td></tr>
<tr><td>BrowseNext</td><td>This method continues a browse operation with previously specified search criteria.

This method has the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>position</td><td>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 TsOpcResultException 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.</td></tr>
<tr><td>[Return Value]</td><td>The set of elements found.</td></tr>
</table>
</td></tr>
<tr><td>CancelSubscription</td><td>This method cancels a subscription and releases all resources allocated for it.</td></tr>
</table>

<table>
<tr><td></td><td>Clients must always explicitly cancel all subscriptions that it creates.

This method has the following parameters:
<table><tr><th>Name</th><th>Description</th></tr><tr><td>subscription</td><td>The subscription to cancel.</td></tr></table></td></tr>
<tr><td>Clone</td><td>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.</td></tr>
<tr><td>Connect</td><td>Connect(string url)

Connects the object to an OPC XML-DA Server.

This method has the following parameters:
<table><tr><th>Name</th><th>Description</th></tr><tr><td>url</td><td>Name of the XML-DA server. The usual form is http::/xxx/yyy, e.g. http://localhost//TsOpcXSampleServer/Service.asmx.</td></tr></table></td></tr>
<tr><td>Connect</td><td>Connect(string url, TsOpcConnectData connectData)

Establishes a physical connection to the remote server.
This method has the following parameters:
<table><tr><th>Name</th><th>Description</th></tr><tr><td>url</td><td>The network address for the remote server.
It replaces the default URL for the server if the method succeeds.</td></tr><tr><td>connectData</td><td>Any protocol configuration or user authentication information.</td></tr></table></td></tr>
<tr><td>Connect</td><td>Connect(string serverName, TsOpcComputerInfo computerInfo)

Connects the object to an OPC Data Access 2.0/3.0 Server.

This method has the following parameters:
<table><tr><th>Name</th><th>Description</th></tr><tr><td>serverName</td><td>Name of the server (ProgID). The usual form is xxx.yyyy.n or xxx.yyy. The registry is checked if the specified name is a version independent name. If so, the associated name is used instead. Version independent server names can be used only on the local machine.</td></tr><tr><td>computerInfo</td><td>Information about the computer to use, like computer name, domain, username and password. For the computer name you can also use DNS names ("domain.com", "server.technosoftware.ch", or "209.130.112.180"). If no computer name is specified then the OPC Server on the local machine is used.</td></tr></table></td></tr>
<tr><td>CreateSubscription</td><td>This method creates a new subscription.

A subscription allows a client to receive asynchronous notifications from the</td></tr>
</table>

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

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>state</td><td>The initial state of the subscription.
The TsCDaSubscriptionState object is described below.</td></tr>
<tr><td>[Return Value]</td><td>The new subscription object.</td></tr>
</table>
</td></tr>
<tr><td>Disconnect</td><td>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.</td></tr>
<tr><td>GetProperties</td><td>This method returns the item properties for a set of items.

This method has the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>itemIDs</td><td>A list of item identifiers.</td></tr>
<tr><td>propertyIDs</td><td>A list of properties to fetch for each item.
If this parameter is null then all available properties are returned.</td></tr>
<tr><td>returnValues</td><td>Whether the property values should be returned with the properties.</td></tr>
<tr><td>[Return Value]</td><td>A list of properties for each item.
The TsCDaItemPropertyCollection object is described below.</td></tr>
</table>
The TsCDaItemPropertyCollection object extends ArrayList and has the following properties/methods:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>ItemName</td><td>The primary identifier for the item within the server namespace.</td></tr>
<tr><td>ItemPath</td><td>The secondary identifier for the item within the server namespace.</td></tr>
<tr><td>Result</td><td>A result code that indicates any item-level errors.</td></tr>
<tr><td>operator[]</td><td>Returns the TsCDaItemProperty object at the specified index.</td></tr>
</table>
</td></tr>
<tr><td>GetResultFilters</td><td>This method returns the filters applied by the server to any item results returned to the client.

This method has the following parameters:</td></tr>
</table>

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 TsOpcItem if bit is set.
ItemPath	0x02	Include the ItemPath in the TsOpcItem if bit is set.
ClientHandle	0x04	Include the ClientHandle in the TsOpcItem 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 TsOpcItem 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.

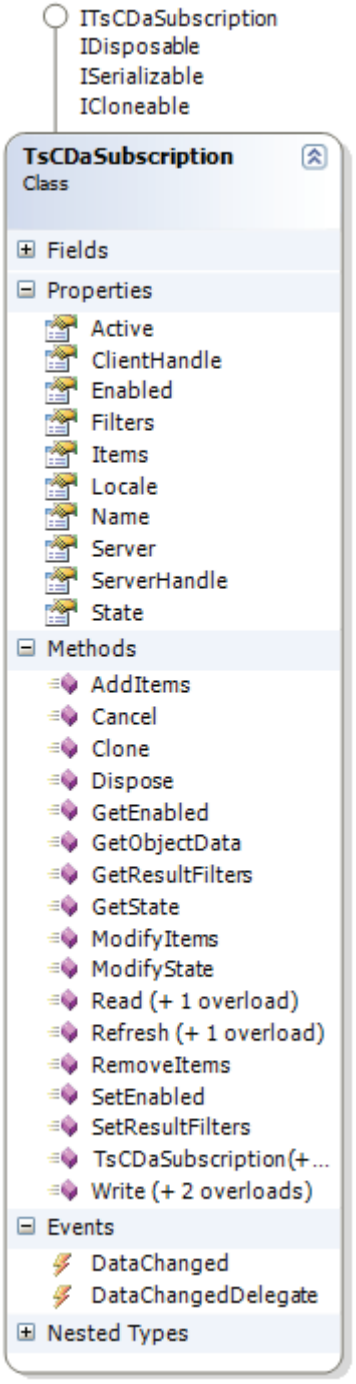
	The server may support writing to the quality and/or timestamp. In these cases, the server does not write the value and returns 'E_NO_WRITEQT' for the item.

3.2.1.1.3 Events

The TsCDaServer class has the following events:

Name	Description
ServerStatusChanged	An event to receive server status notifications.

3.2.1.2 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 and XML-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.

3.2.1.2.1 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.

3.2.1.2.2 Methods

The TsCDaSubscription 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 TsCDaItem object is described in Section 3.1.4.1.
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 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.</td></tr>
</table>
The index in the array is used to associate a result with a specific item.</td></tr>
<tr><td>Cancel</td><td>This method cancels an asynchronous read or writes operation.

This method takes the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>request</td><td>The object returned from the Read or Write request.</td></tr>
<tr><td>callback</td><td>The function to invoke when the cancel completes.</td></tr>
</table></td></tr>
<tr><td>Clone</td><td>Returns an unconnected copy of the subscription with the same items.</td></tr>
<tr><td>GetEnabled</td><td>This method checks whether data change notifications from the server are enabled.

This method takes the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>Whether data changed notifications should be sent.</td></tr>
</table></td></tr>
<tr><td>GetResultFilters</td><td>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 3.1.3.1.

The filters specified at the subscription level override filters specified at the server</td></tr>
</table>

<table>
<tr><td></td><td>level. This method takes the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>A bit mask indicating which fields should be returned in any item results.</td></tr>
</table></td></tr>
<tr><td>GetState</td><td>This method returns the current state of the subscription.
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>[Return Value]</td><td>Returns the current state of the subscription as TsCDaSubscriptionState object.</td></tr>
</table></td></tr>
<tr><td>ModifyItems</td><td>This method modifies the state of items in the subscription.
This method has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>masks</td><td>A bit mask indicating which item state parameters are being modified.</td></tr>
<tr><td>items</td><td>The new state for the items being modified.
The TsCDaItem object is described in Section 3.1.4.1.
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.</td></tr>
<tr><td>[Return Value]</td><td>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.</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>ModifyState</td><td>This method changes the state of a subscription.

This method has the following parameters:

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>masks</td><td>A bit mask that indicates which elements of the subscription state are changing.</td></tr>
<tr><td>state</td><td>The new subscription state.
The TsCDaSubscriptionState object is described in Section 3.1.4.8.
The TsCDaStateMask enumeration contains the bit masks used to indicate which properties of the TsCDaSubscriptionState object contain valid information.</td></tr>
<tr><td>[Return Value]</td><td>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.</td></tr>
</table></td></tr>
<tr><td>Read</td><td>Read(TsCDaItem[] items)

This method reads the values for a set of items in 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 read.
The TsCDaItem object is described in Section 3.1.4.1.
Each item must have its ServerHandle specified.
Each item may have the ReqType and/or MaxAge specified.</td></tr>
<tr><td>[Return Value]</td><td>The results of the read operation for each item as TsOpcItemResult Object.</td></tr>
</table></td></tr>
</table>

<table>
<tr>
<td>Read</td>
<td>
Read(TsCDaItem[] items, object requestHandle,
TsCDaReadCompleteHandler callback, ITsOpcRequest 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:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>items</td><td>The set of items to read.
The TsCDaItem object is described in Section 3.1.4.1.
Each item must have its ServerHandle specified.
Each item may have the ReqType and/or MaxAge specified.</td></tr>
<tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr>
<tr><td>callback</td><td>The function to invoke when the request completes.
The TsCDaReadCompleteHandler delegate is described below.</td></tr>
<tr><td>request</td><td>An identifier for the request (may be used to cancel the request).</td></tr>
<tr><td>[Return Value]</td><td>An array of TsOpcItemResult containing any errors encountered when the server validated the items.</td></tr>
</table>
The TsCDaReadCompleteHandler delegate has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr>
<tr><td>results</td><td>The value of each item as TsCDaItemValueResult array.
The item results always contain the ClientHandle.</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td>Refresh</td><td>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.</td></tr>
<tr><td>Refresh</td><td>Refresh(object requestHandle, ITsOpcRequest 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:

<table><tr><th>Name</th><th>Description</th></tr><tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr><tr><td>request</td><td>An identifier for the request (may be used to cancel the request).</td></tr></table></td></tr>
<tr><td>RemoveItems</td><td>This method modifies the state of items in the subscription.

This method has the following parameters:

<table><tr><th>Name</th><th>Description</th></tr><tr><td>items</td><td>The identifiers (i.e. server handles) for the items being removed.
The TsOpcItem object is described in Section 2.7.2.1.
Each item must have the ServerHandle specified.</td></tr><tr><td>[Return Value]</td><td>An array of TsOpcItemResult containing the results of the remove item operation for each item.</td></tr></table></td></tr>
<tr><td>SetEnabled</td><td>This method enables or disables data change notifications from the server.

This method takes the following parameters:

<table><tr><th>Name</th><th>Description</th></tr><tr><td>enabled</td><td>Whether data changed notifications should be sent.</td></tr></table></td></tr>
<tr><td>SetResultFilters</td><td>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 3.1.3.1.

The filters specified at the subscription level override filters specified at the server level.</td></tr>
</table>

<table>
<tr><td>Write</td><td>
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:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>items</td><td>The set of item values to write.
The TsCDaItemValue object is described in Section 3.1.4.2.
Each item must have its ServerHandle and Value specified.
Each item may have a Quality and/or a Timestamp specified.</td></tr>
<tr><td>[Return Value]</td><td>The results of the write operation for each item as TsOpcItemResult Object.</td></tr>
</table>
</td></tr>
<tr><td>Write</td><td>
Write(TsCDaItemValue[] items, object requestHandle,
TsCDaWriteCompleteHandler callback, ITsOpcRequest 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:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>items</td><td>The set of items to write.
The TsCDaItemValue object is described in Section 3.1.4.2.
Each item must have its ServerHandle specified.
Each item may have the ReqType and/or MaxAge specified.</td></tr>
<tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr>
<tr><td>callback</td><td>The function to invoke when the request completes.
The TsCDaWriteCompleteHandler delegate is described below.</td></tr>
<tr><td>request</td><td>An identifier for the request (may be used to cancel the request).</td></tr>
<tr><td>[Return Value]</td><td>An array of TsOpcItemResult containing any errors encountered when the server validated the items.</td></tr>
</table>
The TsCDaWriteCompleteHandler delegate has the following parameters:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>requestHandle</td><td>A client assigned identifier for the request.</td></tr>
<tr><td>results</td><td>The results of the write operation for each item as TsOpcItemResult array.</td></tr>
</table>
</td></tr>
</table>

			The item results always contain the ClientHandle.

3.2.1.2.3 Events

The TsCDaSubscription class has the following events:

Name	Description
DataChanged	An event to receive data change updates.