Data Access Custom Interface Standard

Version 3.00

Released

March 4, 2003

Specification Type	Industry Standard Specification		
Title:	**OPC Data Access Custom Interface Specification**	Date:	March 4, 2003
Version:	3.0	Soft Source:	MS-Word Opcda30_cust
Author:	Opc Foundation	Status:	**Released**

Synopsis:

This specification is the specification of the interface for developers of OPC Data Access clients and OPC servers.. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of servers and clients by multiple vendors that shall inter-operate seamlessly together.

Trademarks:

Required Runtime Environment:

This specification requires Windows 95, Windows NT 4.0 or later

NON-EXCLUSIVE LICENSE AGREEMENT

The OPC Foundation, a non-profit corporation (the "OPC Foundation"), has established a set of standard OLE/COM interface protocols intended to foster greater interoperability between automation/control applications, field systems/devices, and business/office applications in the process control industry.

The current OPC specifications, prototype software examples and related documentation (collectively, the "OPC Materials"), form a set of standard OLE/COM interface protocols based upon the functional requirements of Microsoft's OLE/COM technology. Such technology defines standard objects, methods, and properties for servers of real-time information like distributed process systems, programmable logic controllers, smart field devices and analyzers in order to communicate the information that such servers contain to standard OLE/COM compliant technologies enabled devices (e.g., servers, applications, etc.).

The OPC Foundation will grant to you (the "User"), whether an individual or legal entity, a license to use, and provide User with a copy of, the current version of the OPC Materials so long as User abides by the terms contained in this Non-Exclusive License Agreement ("Agreement"). If User does not agree to the terms and conditions contained in this Agreement, the OPC Materials may not be used, and all copies (in all formats) of such materials in User's possession must either be destroyed or returned to the OPC Foundation. By using the OPC Materials, User (including any employees and agents of User) agrees to be bound by the terms of this Agreement.

LICENSE GRANT:

Subject to the terms and conditions of this Agreement, the OPC Foundation hereby grants to User a non-exclusive, royalty-free, limited license to use, copy, display and distribute the OPC Materials in order to make, use, sell or otherwise distribute any products and/or product literature that are compliant with the standards included in the OPC Materials.

All copies of the OPC Materials made and/or distributed by User must include all copyright and other proprietary rights notices include on or in the copy of such materials provided to User by the OPC Foundation.

The OPC Foundation shall retain all right, title and interest (including, without limitation, the copyrights) in the OPC Materials, subject to the limited license granted to User under this Agreement.

WARRANTY AND LIABILITY DISCLAIMERS:

User acknowledges that the OPC Foundation has provided the OPC Materials for informational purposes only in order to help User understand Microsoft's OLE/COM technology. THE OPC MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. USER BEARS ALL RISK RELATING TO QUALITY, DESIGN, USE AND PERFORMANCE OF THE OPC MATERIALS. The OPC Foundation and its members do not warrant that the OPC Materials, their design or their use will meet User's requirements, operate without interruption or be error free.

IN NO EVENT SHALL THE OPC FOUNDATION, ITS MEMBERS, OR ANY THIRD PARTY BE LIABLE FOR ANY COSTS, EXPENSES, LOSSES, DAMAGES (INCLUDING, BUT NOT LIMITED TO, DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL, SPECIAL OR PUNITIVE DAMAGES) OR INJURIES INCURRED BY USER OR ANY THIRD PARTY AS A RESULT OF THIS AGREEMENT OR ANY USE OF THE OPC MATERIALS.

GENERAL PROVISIONS:

This Agreement and User's license to the OPC Materials shall be terminated (a) by User ceasing all use of the OPC Materials, (b) by User obtaining a superseding version of the OPC Materials, or (c) by the OPC Foundation, at its option, if User commits a material breach hereof. Upon any termination of this Agreement, User shall immediately cease all use of the OPC Materials, destroy all copies thereof then in its possession and take such other actions as the OPC Foundation may reasonably request to ensure that no copies of the OPC Materials licensed under this Agreement remain in its possession.

User shall not export or re-export the OPC Materials or any product produced directly by the use thereof to any person or destination that is not authorized to receive them under the export control laws and regulations of the United States.

The Software and Documentation are provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor/ manufacturer is the OPC Foundation, P.O. Box 140524, Austin, Texas 78714-0524.

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, the OPC Materials.

Revision 3.0 Highlights

This revision contains the changes made to the Data Access Custom Interface (2.05A Version Baseline). It will be determined at a later time as to whether a Separate Document (The OPC Data Access Automation Specification 3.0) will be provided to describe the OPC Automation Interfaces which facilitate the use of Visual Basic, Delphi and other Automation enabled products to interface with OPC Servers. The following functionality has been added to this version of the OPC Data Access Custom Interface Specification

- Added new interfaces
  - IOPCBrowse
  - IOPCItemDeadbandMgt
  - IOPCItemSamplingMgt
  - IOPCItemIO
  - IOPCSyncIO2
  - IOPCAsyncIO3
  - IOPCGroupStateMgt2
- Clarified startup issues and add WAITING_FOR_INITIAL_DATA quality status mask.
- Added Item Property #7, #8 for EUTYPE
- Clarify SetActiveState to indicate that an item transition from inactive to active will ultimately result in a quality change, triggering a callback.
- Add new server status enumeration (OPC_STATUS_COMM_FAULT)
- Heading labels added to methods for easier access via the Table of Contents
- Removed Legacy Interfaces from this version of the specification
  - IOPCServerPublicGroups
  - IOPCBrowseServerAddressSpace
  - IOPCPublicGroupStateMgt
  - IOPCAsyncIO
  - IOPCItemProperties
- In addition, spelling, grammer, formatting and minor clarifications were added to improve the quality of the specification
- Added Item Properties section
- Clarify RemoveGroup
- Added CATIDs to idl
- Provided the ability to read and write the quality and timestamp
- KeepAlive mechanism was added to confirm the callback connection health

Revision 2.05A Highlights

This revision clarifies section 4.2.13 Note (5) regarding roundup when converting floats and doubles to integers. Also correct an error in the property definitions in Appendix D (200-207 were off by 1).

Revision 2.05 Highlights

This revision includes numerous clarifications to Section 4.2.13 regarding data conversion between Native and Requested data types.

Revision 2.04 Highlights

This revision includes additional minor clarifications to certain ambiguities which were discovered during Interoperability sessions and during the development of the Compliance Test. The affected sections include: TimeBias and DeadBand discussion in Group Object: General Properties (4.5.1). LocaleID for SetState (to make it clear the behaviour is optional). Addition or Clarification of error returns E_INVALIDARG and S_FALSE return for GetItemProperties, LookupItemIDs, AddItems, ValidateItems,

RemoveItems, SetActiveState, SetClientHandles, SetDataTypes, both SyncIO and AsyncIO Read and Write. In particular for S_FALSE: change 'was partially successful' to 'completed with one or more errors'. This now clearly implies that the method outputs (specifically the ppErrors returns) are defined in this case. Other adjustments to the text were to make error returns more consistant across functions. Clarify GetItemID behavior. In Refresh2 and IOPCDataCallback::OnDataChange the Transaction ID parameter is clarified. Specifically: 0 is an allowed value. See also the introduction to OPCAsyncIO (4.5.6). Also add section 4.2.14 as a general discussion of Client and Server responsibilites regarding LocaleID.

Revision 2.03 Highlights

This revision includes minor clarifications to the Deadband discussion (4.5.1.6). It also clarifies the behavior of empty enumerators; The descriptions of IOPCServer::CreateGroupEnumerator and IOPCBrowseServerAddressSpace::BrowseAccessPaths have been clarified and corrected. They are now consistant with the existing description of IOPCBrowseServerAddressSpace::BrowseOPCItemIDs.

Revision 2.02 Highlights

This revision includes minor clarifications to the OPCItemProperties Interface discussions (4.4.6), GroupStateMgt::SetState (4.5.3.2) and the old (1.0) Stream Marshalling Discussion (4.6.4.6).

Revision 2.01 Highlights

This revision includes clarifications to the dwAccessRightsFilter in IOPCBrowseServerAddressSpace and also the discussion of access rights in general (section 6.7.6).

Revision 2.0 Highlights

This revision includes enhancements to the 1.0A Specification. Although changes were made throughout the document, the following areas are or particular importance:

- This is now refered to as the OPC Data Access Specification as there are other OPC efforts underway.
- The Automation Interface specification has been separated into a separate document.
- All previous (1.0A) Custom Interfaces remain in place and unchanged except for minor clarifications.
- Async and exception based connections should now be done using ConnectionPoints rather than IDataObject. The existing IOPCAsyncIO, IDataObject and Client side IAdviseSink interfaces support ‘old style’ (Version 1.0) connections. The new IOPCAsyncIO2, IConnectionPointContainer and Client side IOPCDataCallback interfaces support the ‘new style’ Version 2.0 connections.
- The behavior of the existing IOPCAsyncIO, IDataObject and Client side IAdviseSink interfaces is unchanged however their support is optional for OPC 2.0 complaint software. The new IOPCAsyncIO2, IConnectionPointContainer and Client side IOPCDataCallback interfaces are required for 2.0 compliant software.
- A new ‘convenience’ interface is defined. IOPCItemProperties allows easy access to common and vendor specific properties or attributes of an Item or Tag.
- A ShutdownRequest capability is added via a Connection point on the Server object and a Client side IOPCShutdown interface that allows the server to request that all clients disconnect from the server. This interface will also be used by other OPC server types.
- An IOPCCommon interface is added to the server. This interface provides several common LocaleID related functions. This interface will also be used by other OPC server types.
- The OPC_BROWSE_TO capability is added to BrowseServerAddressSpace.

Table of Contents

1 Introduction

A General Introduction to OPC is contained in a separate OPC Overview Document (OPCOVW.DOC). This particular document deals specifically with the OPC Data Access Interfaces.

1.1 Audience

This specification is intended as reference material for developers of OPC compliant Clients and Servers. It is assumed that the reader is familiar with Microsoft OLE/COM technology and the needs of the Process Control industry.

This specification is intended to facilitate development of OPC Servers in C and C++, and of OPC client applications in the language of choice. Therefore, the developer of the respective component is expected to be fluent in the technology required for the specific component.

1.2 Deliverables

The deliverables from the OPC Foundation with respect to the OPC Data Access Specification 3.0 include the OPC Specification itself, OPC IDL files (included in this document as Appendices) and the OPC Error header files (included in this document). As a convenience, standard proxystub DLLs and a standard Data Access Header file for the OPC interfaces generated directly from the IDL will be provided at the OPC Foundation Web Site. OPC Data Access 3.0 sample code will also be available from the OPC Foundation Web Site (members only).

This OPC Data Access specification contains design information for the following:

1. **The OPC Data Access Custom Interface** - This document will describe the Interfaces and Methods of OPC Components and Objects.

2. **The OPC Data Access Automation Interface** - It will be determined at a later time as to whether a Separate Document (The OPC Data Access Automation Specification 3.0) will be provided to describe the OPC Automation Interfaces which facilitate the use of Visual Basic, Delphi and other Automation enabled products to interface with OPC Servers.

2 OPC Data Access Fundamentals

This section introduces OPC Data Access and covers topics which are specific to OPC Data Access. Additional common topics including Windows NT, UNICODE, Threading Models, etc are discussed in the OPC Overview Document (OPCOVW.DOC).

2.1 OPC Overview

This specification describes the OPC COM Objects and their interfaces implemented by OPC Servers. An OPC Client can connect to OPC Servers provided by one or more vendors.

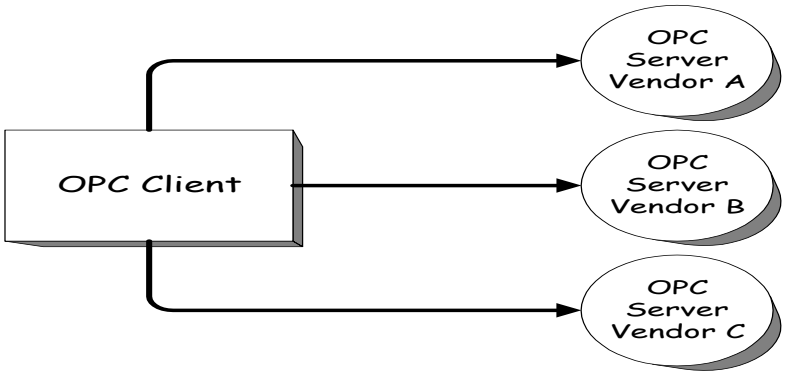


Figure 2-1 OPC Client

Different vendors may provide OPC Servers. Vendor supplied code determines the devices and data to which each server has access, the data names, and the details about how the server physically accesses that data. Specifics on naming conventions are supplied in a subsequent section.

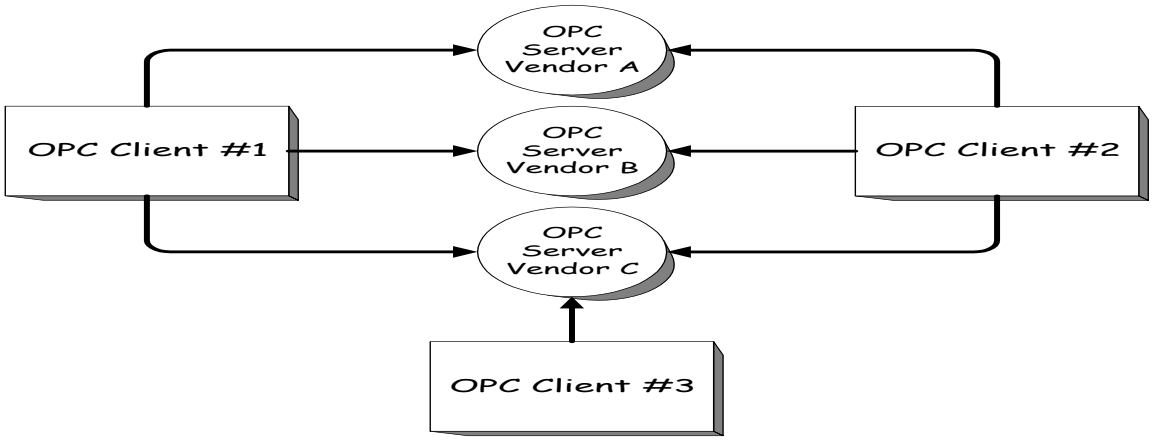


Figure 2-2 OPC Client/Server Relationship

At a high level, an OPC server is comprised of several objects: the server, the group, and the item. The OPC server object maintains information about the server and serves 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 Groups provide a way for clients to organize data. For example, the group might represent items in a particular operator display or report. Data can be read and written. Exception based connections can also be created between the client and the items in the group and can be enabled and disabled as needed. An OPC client can configure the rate that an OPC server should provide the data changes to the OPC client.

Within each Group the client can define one or more OPC Items.

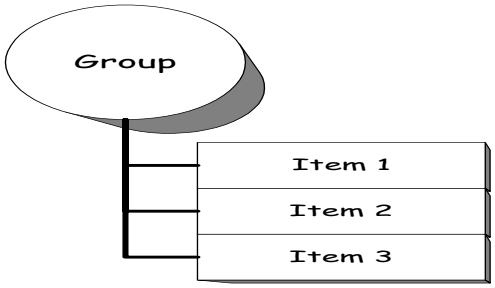


Figure 2-3 - Group/Item Relationship

The OPC Items represent connections to data sources within the server. An OPC Item, from the custom interface perspective, is not accessible as an object by an OPC Client. Therefore, there is no external interface defined for an OPC Item. All access to OPC Items is via an OPC Group object that "contains" the OPC item, or simply where the OPC Item is defined.

Associated with each item is a Value, Quality and Time Stamp. The value is in the form of a VARIANT, and the Quality is similar to that specified by Fieldbus.

Note that the items are not the data sources - they are just connections to them. For example, the tags in a DCS system exist regardless of whether an OPC client is currently accessing them. The OPC Item should be thought of as simply specifying the address of the data, not as the actual physical source of the data that the address references.

2.2 Where OPC Fits

Although OPC is primarily designed for accessing data from a networked server, OPC interfaces can be used in many places within an application. At the lowest level they can get raw data from the physical devices into a SCADA or DCS, or from the SCADA or DCS system into the application. The architecture and design makes it possible to construct an OPC Server which allows a client application to access data from many OPC Servers provided by many different OPC vendors running on different nodes via a single object.

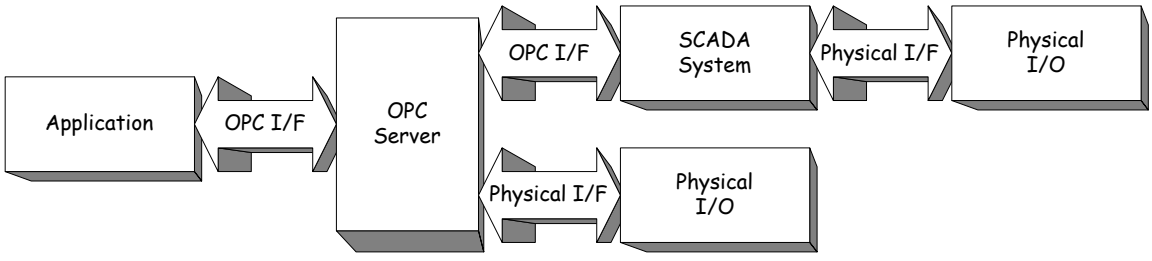


Figure 2-4 - OPC Client/Server Relationship

2.3 General OPC Architecture and Components

OPC is a specification for two sets of interfaces; the OPC Custom Interfaces and the OPC Automation interfaces. A revised automation interface may be provided with release 3.0 of the OPC specification. This is shown below.

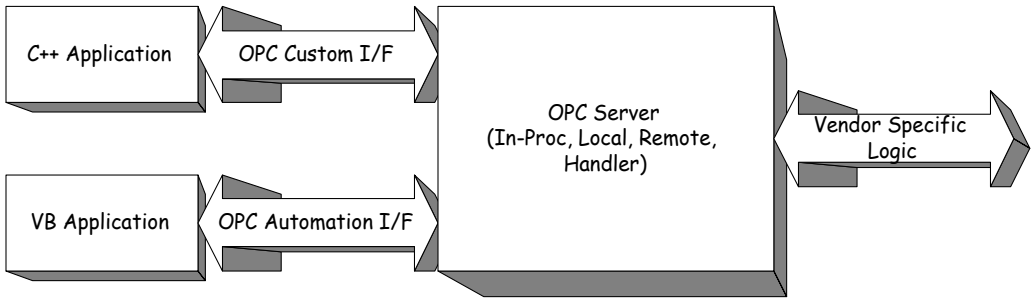


Figure 2-5 - The OPC Interfaces

The OPC Specification specifies COM interfaces (what the interfaces are), not the implementation (not the how of the implementation) of those interfaces. It specifies the behavior that the interfaces are expected to provide to the client applications that use them.

Included are descriptions of architectures and interfaces that seemed most appropriate for those architectures. Like all COM implementations, the architecture of OPC is a client-server model where the OPC Server component provides an interface to the OPC objects and manages them.

There are several unique considerations in implementing an OPC Server. The main issue is the frequency of data transfer over non-sharable communications paths to physical devices. Thus, we expect that the OPC Server will either be a local or remote EXE which includes code that is responsible for efficient data collection from a physical device.

An OPC client application communicates to an OPC server through the specified OPC interfaces. OPC servers must implement the custom interface, and optionally may implement the automation interface if defined.

An inproc (OPC handler) may be used to marshal the interface and provide the additional Item level functionality of the OPC Automation Interface. Refer to the figure below: Typical OPC Architecture.

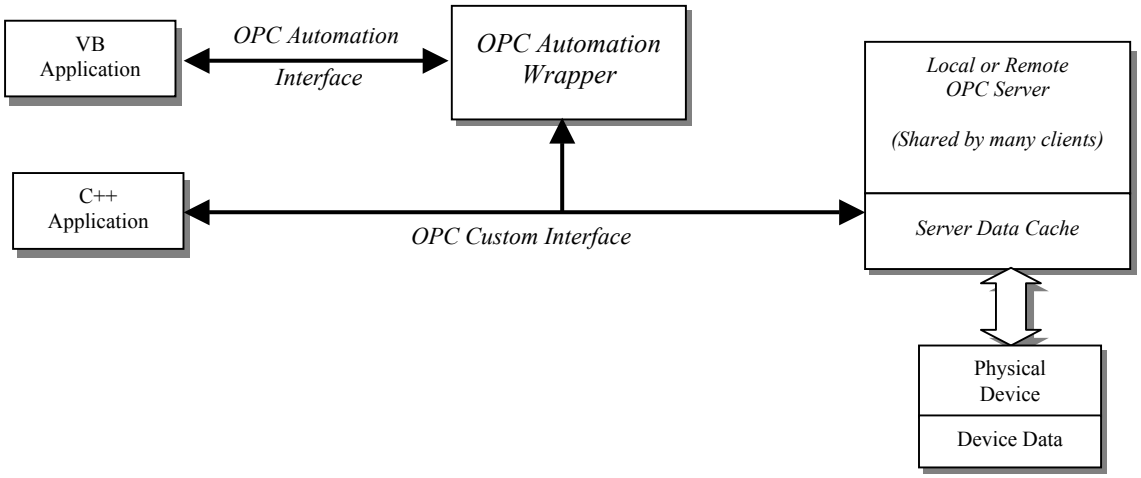


Figure 2-6 - Typical OPC Architecture

It is also expected that the server will consolidate and optimize data accesses requested by the various clients to promote efficient communications with the physical device. For inputs (Reads), data returned by the device is buffered for asynchronous distribution or synchronous collection by various OPC clients. For outputs (writes), the OPC Server updates the physical device data on behalf of OPC Clients.

2.4 OPC Data Access Architecture Companion Specifications

OPC Data Access provides the baseline functionality for accessing (reading and writing) data from various networked devices via a standard set of interfaces. The interfaces facilitate the interoperability between clients and servers discovering each other and communicating the set of capabilities (functionality & name space, information about the items in the names space), in addition to a well define set of interfaces to facilitate various mechanisms to read and write the data items according to the needs of the client application. The primary intent of OPC Data Access is to provide the interfaces for data acquisition (accessing services) in support of the vertical architecture (serve data from a device to a client application on a higher level computer).

The OPC Foundation is working towards extending the capabilities of the OPC Data Access interfaces. Companion interfaces are being added to provide needed functionality to extend the accessing of data. This includes the ability to share server data between devices on different industrial networks.

The companion architectures include:

OPC Common Definitions and Interfaces contains common rules and design criteria and the specification of interfaces which are common for several topics including Data Access.

OPC Complex Data, which specifies how to use Data Access to exchange data structures. It provides the mechanism for communicating the structure of the data as well as the actual values of the data. Complex Data has not been incorporated into the OPC Data Access 3.0 specification.

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 Ethernet-based systems including PLCs, HMI/SCADA, Devices, and PCs.

2.5 Overview of the Objects and Interfaces

The OPC Server object provides a way to access (read/write) or communicate to a set of data sources. The types of sources available are a function of the server implementation.

An OPC client connects to an OPC server and communicates to the OPC server through the interfaces. The OPC server object provides functionality to an OPC client to create and manipulate OPC group objects. These groups allow clients to organize the data they want to access. A group can be activated and deactivated as a unit. A group also provides a way for the client to 'subscribe' to the list of items so that it can be notified when they change.

Note: All COM objects are accessed through Interfaces. The client sees only the interfaces. Thus, the objects described here are 'logical' representations which may not have anything to do with the actual internal implementation of the server. The following figure is a summary of the OPC Objects and their interfaces. Note that some of the interfaces are Optional (as indicated by [ ]).

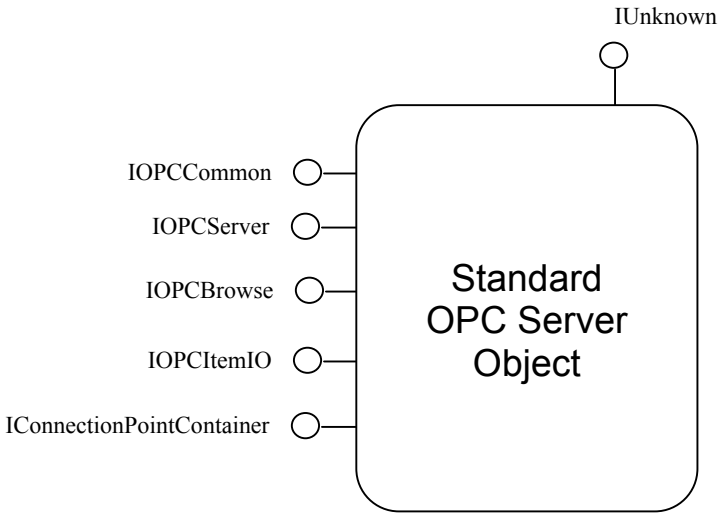


Figure 2-7 - Standard OPC Server Object

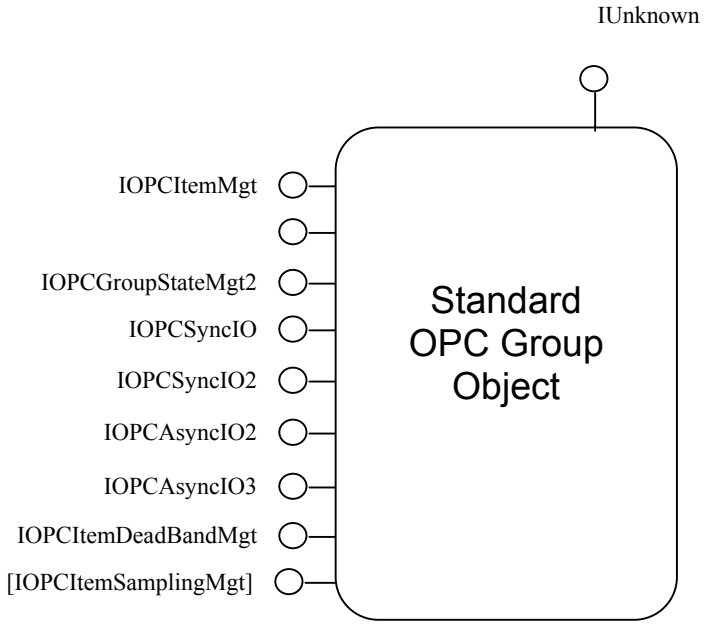


Figure 2-8 - Standard OPC Group Object

2.6 The Address Space and Configuration of the Server

This release of the OPC specification assumes that a server configuration address space may be managed and persistently stored using the IPersistFile interface. Only the server specific information is persistently stored. All client configuration information (Group and Item Definitions) must be persistently stored by the respective client application. All Handles that are defined in the system are not guaranteed to have the same value between sessions of the client and server conversation.

It is important to distinguish the address space of the server (also known as the server configuration) from the small subsets of this space that a particular client may be interested in at a particular time (also known as the 'groups' and 'items'). The details of how these client specific groups are maintained are discussed in detail in this specification. The persistent storage of groups is the responsibility of the respective clients. The details of how the server address space is defined and configured are intentionally left unspecified. For example the server address space might be:

- Entirely fixed (e.g. for a dedicated interface to a particular device such as a scale).
- Configured entirely outside of the OPC environment (e.g. for an interface to an existing external DCS system).
- Automatically configured at startup by an 'intelligent' server which can poll the existing system for installed hardware or interfaces.
- Automatically configured on the fly by an 'intelligent' server based on the names of the data items the client applications are currently requesting.

It is expected that this server address space is stable and is managed within the server. The clients will define and manage the relatively small lists of items called 'groups' as needed from time to time. The

interfaces described here provide the client the ability to easily define, manage, and recreate these lists as needed through the use of 'OPCGroups'. The clients direct the server to create, manage and delete these groups on their behalf (persistence of the groups is the responsibility of the client application).

2.7 Application Level Server and Network Node Selection

OPC Data Access supports the concept of organizing client requests into groups within a server. Such groups can contain requests for data from only one particular OPC Server object. In order to access data, a client application will need to specify the following:

- The name of the OPC Data Access Server (for use by CoCreateInstance, CoCreateInstanceEx, etc.)
- The name of the machine hosting the OPC Data Access Server (for use by CoCreateInstanceEx)
- The vendor specific OPC Item Definition (the name of the specific data item in the server's address space)

It is beyond the scope of this specification to discuss the implications of this on the architecture and user interface of the client program.

2.8 Synchronization and Serialization Issues

By 'synchronization' we mean the ability of a client to read or write values and attributes in a single transaction. For example, most applications want to insure that the value, quality and time stamp attributes of a particular item are in 'sync'. Also, a reporting package might want to insure that a group of several values read together as part of a 'Batch Report' are in fact part of the same batch. Finally, a recipe download package would want to insure that all of the values in the group were sent together and that the recipe was not started until all of the values had been received. These are just a few examples where synchronization is important.

The short answer is that OPC itself cannot insure that all of these synchronization tasks can be accomplished. Additional handshaking and flag passing between the client application and the device server to signal such states as 'ready' and 'complete' will be required. There are also things that need to be specified about the behavior of OPC servers to assure that OPC does not prevent this sort of synchronization from being done.

It will be seen later that OPC allows explicit reads and writes of groups of items or of individual items as well as exception based data connections (OnDataChange). Without jumping ahead too far it is possible to make some general observations about these issues and about server behavior.

1. In general, OPC Servers should try to preserve synchronization of data items and attributes that are read or written in a single operation. Synchronization of items read or written individually in separate operations is not required. Clearly, data read from different physical devices is difficult to synchronize.
2. Reads and writes of data items which can be accessed by more than one thread must be implemented to be thread safe, to the extent that data synchronization is preserved as specified in this specification. Examples of where this is important might include: logic within a server where one thread services method executions while a separate thread performs the physical communications and writes the received data into a buffer area which is shared with the first thread. Another example might be the logic in a handler or proxy where a 'hidden' RPC thread servicing an OnDataChange subscription is writing data into a shared buffer which a thread in the client might be reading.
3. Threading issues are always important but this is especially true on SMP systems.

By 'Serialization' we mean the ability of the client to control the order in which writes are performed.

1. It is STRONGLY RECOMMENDED that write requests to the same device be handled 'in-order' by any server implementation. For example, an application might use a 'recipe download complete' flag which is set by the application after the individual recipe items are sent. In this case, the data must be transmitted to the physical device in the same order it was output to insure that the 'complete' flag is not set before all the data has actually arrived. Where the server buffers the outgoing data and implements a separate communications manager thread to send these outputs to the physical device (as is often the case), the server implementation must take extra care to insure that the order of the outputs is preserved.
2. Where a client can both read values explicitly or receive updates via a callback attention must be given to defining exactly when a callback will or will not occur. This is discussed in more detail later.

Many of these issues will be clarified in the detailed descriptions of the methods below.

2.9 Persistent Storage Story

OPC Servers may implement an optional interface to facilitate OPC clients telling an OPC server to persistent (store) the OPC server configuration information. OPC Server configuration information may include information about the devices and data source necessary to facilitate communication between the data source and the OPC server. Client configuration information, including the groups and items, are not persistently stored by an opc server.

Clients are responsible for the configuration and persistent storage of the groups and items that are required by their application..

3 OPC Data Access Quick Reference

This section includes a quick reference for the methods on the Custom Interface. These interfaces, their parameters and behavior are defined in more detail later in the reference sections.

3.1 Custom Interface

Note: This section does not show additional standard COM Interfaces such as IUnknown, IEnumString and IEnumUnknown used by OPC Data Access.

OPCServer

- **IOPCServer**
- **IOPCBrowse (new 3.0)**
- **IOPCItemIO (new 3.0)**
- **IConnectionPointContainer**
- **IOPCCommon**

OPCGroup

- **IOPCGroupStateMgt**
- **IOPCGroupStateMgt2 (new 3.0)**
- **IOPCASyncIO2**
- **IOPCAsyncIO3 (new 3.0)**
- **IOPCItemMgt**
- **IOPCItemDeadbandMgt (new 3.0)**
- **IOPCItemSamplingMgt (new 3.0, optional)**
- **IConnectionPointContainer**
- **IOPCSyncIO**
- **IOPCSyncIO2 (new 3.0)**
- **EnumOPCItemAttributes**
- **IEnumOPCItemAttributes**

4 OPC Custom Interface

4.1 Overview of the OPC Custom Interface

The OPC Custom Interface Objects include the following custom objects:

- OPCServer
- OPCGroup

The interfaces and behaviors of these objects are described in detail in this chapter. Developers of OPC servers are required to implement the OPC objects by providing the functionality defined in this chapter.

This chapter also references and defines expected behavior for the standard OLE interfaces. Interfaces that an OPC server and an OPC client are required to implement when building OPC compliant components.

Also, standard and custom Enumerator objects are created, and interfaces to these objects are returned. In general the enumerator objects and interfaces are described briefly since their behavior is well defined by OLE.

The OPC specification follows the preferred approach that enumerators are created and returned from methods on objects rather than through QueryInterface. The enumerators are as follows:

- Group Enumerator - (see IOPCServer::CreateGroupEnumerator)
- Item Attribute Enumerator - (see IOPCItemMgt::CreateEnumerator)

Also you will note that in some cases, lists of things are returned via enumerators and in other cases as simple lists of items. Our choice depends on the expected number of items returned. ‘Large’ lists are best returned through enumerators while ‘small’ lists are more easily and efficiently returned via explicit lists.

4.2 General Information

This section provides general information about the OPC Interfaces, and some background information about how the designers of OPC expected these interfaces to be implemented and used.

4.2.1 Version Interoperability

Data Access Servers may be compatible with the requirements of Versions 1.0a, 2.x or 3.0 of the specification or with any combinations of the Data Access Specification Versions. Data Access Clients may also be compatible with the requirements of Versions 1.0a, 2.x or 3.0 of the specification or with any combinations of the Data Access Specification Versions.

The best migration strategy for server and client vendors will depend on their particular business situation. For example a vendor who mostly sells his own client and server components as a packaged system and for whom OPC Compatability represents a long term strategy will have less need to support multiple versions of the interfaces.

As a general guideline it is recommended that existing server vendors add version 3.0 support and continue to support versions 1.0A and 2.x, allowing existing clients to migrate at their own pace.

Data Access Server **Required Interfaces**	**1.0**	**2.0**	**3.0**
OPCServer			
IUnknown	Required	Required	Required
IOPCServer	Required	Required	Required
IOPCCommon	N/A	Required	Required
IConnectionPointContainer	N/A	Required	Required
IOPCItemProperties	N/A	Required	N/A
IOPCBrowse	N/A	N/A	Required
IOPCServerPublicGroups	Optional	Optional	N/A
IOPCBrowseServerAddressSpace	Optional	Optional	N/A
IOPCItemIO	N/A	N/A	Required
OPCGroup			
IUnknown	Required	Required	Required
IOPCItemMgt	Required	Required	Required
IOPCGroupStateMgt	Required	Required	Required
IOPCGroupStateMgt2	N/A	N/A	Required
IOPCPublicGroupStateMgt	Optional	Optional	N/A
IOPCSyncIO	Required	Required	Required
IOPCSyncIO2	N/A	N/A	Required
IOPCAsyncIO2	N/A	Required	Required
IOPCAsyncIO3	N/A	N/A	Required
IOPCItemDeadbandMgt	N/A	N/A	Required
IOPCItemSamplingMgt	N/A	N/A	Optional
IConnectionPointContainer	N/A	Required	Required
IOPCAsyncIO	Required	Optional	N/A
IDataObject	Required	Optional	N/A

4.2.2 Ownership of memory

Per the COM specification, clients must free all memory associated with ‘out’ or ‘in/out’ parameters. This includes memory that is pointed to by elements within any structures. This is very important for client writers to understand, otherwise they will experience memory leaks that are difficult to find. See the IDL files to determine which parameters are out parameters. The recommended approach is for a client to create a subroutine that is used for freeing each type of structure properly.

Independent of success/failure, the server must always return well defined values for 'out' parameters. Releasing the allocated resources is the client's responsibility.

Note: If the error result is any FAILED error such as E_OUTOFMEMORY , the OPC server should return NULL for all `out' pointers (this is standard COM behavior). This rule also applies to the error arrays (ppErrors) returned by many of the functions below. In general, a robust OPC client should check each out or in/out pointer for NULL prior to freeing it.

4.2.3 Standard Interfaces

Per the COM specification, all methods must be implemented on each required interface.

Per the COM specification, any optional interfaces that are supported must have all functions within that interface implemented, even if the implementation is only a stub implementation returning E_NOTIMPL.

4.2.4 Null Strings and Null Pointers

Both of these terms are used below. They are NOT the same thing. A NULL Pointer is an invalid pointer (0) which will cause an exception if used. A NUL String is a valid (non zero) pointer to a 1 character array where that character is a NUL (i.e. 0). If a NUL string is returned from a method as an [out] parameter (or as an element of a structure) it must be freed, otherwise the memory containing the NUL will be lost. Also note that a NULL pointer cannot be passed for an [in,string] argument due to COM marshalling restrictions. In this case a pointer to a NUL string should be passed to indicate an omitted parameter. Whenever possible, NUL strings are used in this specification. C# does not handle NUL pointers or NUL strings.

4.2.5 Returned Arrays

You will note the syntax **size_is(,dwCount)** in the IDL used in combination with pointers to pointers. This indicates that the returned item is a pointer to an actual array of the indicated type, rather than a pointer to an array of pointers to items of the indicated type. This simplifies marshaling, creation, and access of the data by the server and client.

4.2.6 CACHE data, DEVICE data and TimeStamps

For the most part the terms CACHE and DEVICE are treated as 'abstract' within this specification. That is, reading CACHE or DEVICE data simply affects the described behavior of various interfaces in a well defined way. The implementation details of these capabilities is not dictated by this specification.

In practice, however, it is expected that most servers will read data into some sort of CACHE. Also, most clients will read data from this cache via one of several mechanisms discussed later. Access to DEVICE data is expected to be 'slow' and is expected to be used primarily for diagnostics or for particularly critical operations.

The CACHE should reflect the latest value of the data (subject to update rate and deadband optimizations as discussed later) as well as the quality and timestamp. The Timestamp should indicate the time that the value and quality was obtained by the device (if this is available) or the time the server updated or validated the value and quality in its CACHE. Note that if a device or server is checking a value every 10 seconds then the expected behavior would be that the timestamp of that value would be updated every 10 seconds (even if the value is not actually changing). Thus the time stamp reflects the time at which the server knew the corresponding value was accurate.

This is also true regardless of whether the physical device to system interface is exception based. For example suppose it is known that (a) an exception based device is checking values every 0.5 second and that (b) the connection to the device is good and (c) that device sent an update for item FIC101 three minutes ago with a value of 1.234. In this case the value returned from a cache read would be 1.234 and more important, the timestamp returned for this value would be the current time (within 0.5 second) since it is known that the value for the item is in fact still 1.234 as of 0.5 seconds ago.

4.2.7 Time Series Values

The OPC Data Access interfaces are designed primarily to take snapshots of current real time process or automation data. The Timestamp returned with those values is intended primarily as an indication of the quality of that 'current' data. These interfaces are not really intended to deal with buffered time series data for a single point such as historical data. If the server chooses to implement the IOPCItemSamplingMgt interface, buffering of data is allowed. See IOPCItemSamplingMgt for more details.

4.2.8 Asynchronous vs. Synchronous Interfaces

Assuming that most clients want to access Cached data, there are several ways for a client to obtain that data from a server.

- It 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 and where maximum efficiency is not a concern. A client that operates in this way is essentially duplicating the 'scanning' that the server is already doing.
- It can 'subscribe' to cached data using IOPCDataCallback.. This is the recommended behavior for clients because it will minimize use of CPU and NETWORK resources.

4.2.9 The ACTIVE flags, Deadband and Update Rate

These attributes of groups and items can be used to reduce resource use by clients and servers. They are discussed in more detail later under GROUPS. In general, they affect how often the cached data and quality information is updated and how often calls are made to the client's IOPCDataCallback.

4.2.10 Errors and return codes

The OPC specification describes interfaces and corresponding behavior that an OPC server implements, and an OPC client application depends on. A list of OPC Specific errors and return codes is contained in the summary of OPC error codes section in this specification. For each method described below a list of all possible OPC error codes as well as the **most common** OLE error codes is included. It is likely that clients will encounter additional error codes such as RPC and Security related codes in practice and they should be prepared to deal with them.

In two cases (Read and Write) it is also allowed for a server to return Vendor Specific error codes. Such codes can be passed to GetErrorString method. This is discussed in more detail later.

In all cases 'E' error codes will indicate FAILED type errors and 'S' error codes will indicate at least partial success.

4.2.11 Startup Issues

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 an AdviseSink or ConnectionPoint based subscription and/or execute a Refresh on such a subscription before the values

are available. You will see in the later discussions of subscriptions that an initial callback is expected which contains all values in a Group. The expected behavior in this situation is summarized by saying that as items are added to a group, their initial state should be set to OPC_QUALITY_BAD with a OPC_QUALITY_WAITING_FOR_INITIAL_DATA (20) status mask. Any client operation on the group will then behave as it normally would for a group with a mixed set of GOOD and BAD qualities. If the server cannot determine the datatype quickly at AddItem time it will return VT_EMPTY for the canonical datatype. Note that in the case of the sync read and also asyncio2 operations the server can return vendor specific error information which could indicate a vendor specific error such as "SERVER WAITING FOR INITIAL DATA".

4.2.12 VARIANT Data Types and Interoperability

In order to promote interoperability, the following rules and recommendations are presented.

General Recommendations:

- The VARIANT types VT_I2, I4, R4, R8, CY, DATE, BSTR, BOOL, UI1 as well as single arrays of these types (VT_ARRAY) are expected to be most commonly used canonical types (in part because these are the legal types in Visual Basic).
- It is recommended that whenever possible, clients request data in one of these formats and that whenever possible, servers be prepared to return data in one of these formats.
- It is expected that use of other extended types will most likely occur where the Server and Client were written by the same vendor and the server intends to pass some non-portable vendor specific data back to the client. In the interests of interoperability, such transactions should be minimized.
- It has been found in practice that some servers (for example those connecting to remote locations) are unable to determine the Native Datatype at the time an item is added or validated. It has become common practice for such servers to return VT_EMPTY as the native datatype. Such servers will retain the requested type (which may also be VT_EMPTY) and will return the data in the requested type (which may be 'Native') when the data becomes available and they are able to determine its actual type. It is recommeneded (but not required) that clients be prepared to deal with an initial return of VT_EMPTY from AddItems or ValidateItems.

General Rules:

- Servers are allowed to maintain and return any legal Canonical Data Type (any legal permutation of VT_ flags) in addition to the recommended types above.
- Clients are allowed to request any legal Variant Data Type in addition to the recommended types above.
- Servers should be prepared to deal in an elegant way with requested types even when they are unable to convert their data to this type. That is, they should not malfunction, return incorrect results or lose memory. As mentioned elsewhere they may return a variety of errors including any error returned by the Microsoft function: VariantChangeType.
- Clients should always be prepared to deal with servers which are unable to handle a requested datatype. That is, they should not malfunction or lose memory when an error is returned.
- Clients which request VT_EMPTY (which by convention indiciates that the server should return it's canonical type) should likewise be prepared to deal with any returned type. That is, even if they find that they are not able to use or display the returned data, they should properly free the data (using VariantClear) and should probably indicate to the user that a datatype was returned which is not usable by this client.
- Real values in the variant (VT_R4, VT_R8) will contain IEEE floating point numbers. Note that the IEEE standard allows certain non-numeric values (called NANs) to be stored in this format.

While use of such values is rare, they are specifically allowed. If such a value is returned it is required that the QUALITY flag be set to OPC_QUALITY_BAD.

Although the IEEE standard allows NANs to be stored in VT_R4 and VT_R8 format, such values can only be read and written using the exact native format of the target item. They will not be converted to/from other types. When such a value is read (as a native type) the QUALITY flag must be returned by the server as OPC_QUALITY_BAD. If such a value is written (as a native type) then the QUALITY flag provided by the client MUST be OPC_QUALITY_BAD. Whether or not a particular server ever returns or accepts such values is server specific.

Additional Rules regarding Data Conversion

OPC Servers **must** support at least the following conversions between Canonical and Requested datatypes. Reading and Writing should be symmetric. Note that the easiest way for most server implementers to provide this functionality is to use the VariantChangeTypeEx() function available in the COM libraries. In the table below, conversions marked OK can be expected to always work. Other conversions may or may not work depending on the specific value of the source.

As noted elsewhere in this specification the Client can specify a localeID to be used and the server should pass this to VariantChangeTypeEx() for all conversions. Note that it is possible for the end user to override some of the default Locale Settings in the Control Panel Regional Settings Dialog. For example in English it is possible to select date formats of either MM/DD/YY or YY/MM/DD as well other formats. Clearly a date of 03/02/01 is ambiguous in this case. It is the End User's responsibility to insure that the Regional Settings for a given localeID are compatible on different machines within his network.

FROM…	I1	UI1	I2	UI2	I4	UI4	R4	R8	CY	DATE	BSTR	BOOL
TO…												
I1	OK	OK(7)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)(3)	(4)	OK
UI1	OK(7)	OK	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)(3)	(4)	OK
I2	OK	OK	OK	OK(7)	(1)	(1)	(1)	(1)	(1)	(1)(3)	(4)	OK
UI2	(1)	OK	OK(7)	OK	(1)	(1)	(1)	(1)	(1)	(1)(3)	(4)	OK
I4	OK	OK	OK	OK	OK	OK(7)	(1)	(1)	(1)	OK	(4)	OK
UI4	(1)	OK	(1)	OK	OK(7)	OK	(1)	(1)	(1)	OK	(4)	OK
R4	OK	OK	OK	OK	OK	OK	OK	(1)	OK	OK	(4)	OK
R8	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	(4)	OK
CY	OK	OK	OK	OK	OK	OK	(1)	(1)	OK	OK	(4)	OK
DATE	OK	OK	OK	OK	(1)	(1)	(1)	(1)	(1)	OK	(4)	OK
BSTR	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
BOOL	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	(4)	OK

Notes:
(1) Conversion on 'downcast' e.g. from I4 to I2 or R8 to R4 is allowed although Overflow is possible. If overflow occurs an error (DISP_E_OVERFLOW) is returned and in the case of Read the quality is set to BAD. In the case of Write the target value is not changed.

(2) Note that the internal storage type of VT_BOOL is 'short' which must have the values VARIANT_TRUE (0xFFFF – i.e. '-1' when stored in a 'short') and VARIANT_FALSE (0). When converting TO bool any non-zero value converts to VARIANT_TRUE. Converting FROM bool to any signed numeric type, VARIANT_TRUE converts to '-1' or '-1.0'. For unsigned types it converts to the maximum value. The **recommended** OPC standard for conversion of bool to BSTR is "0" or "-1" rather than "True" or "False". If a server chooses to convert to "True" or "False" is **must** account for the Locale (e.g. by passing VARIANT_LOCALBOOL to VariantChangeTypeEx). It should also be noted that the C++ keyword 'true' is an abstract type which converts to '1' when assigned to any other

value (e.g. to a short). Thus it is a coding error to assign 'true' to 'boolVal' which must always be set to VARIANT_TRUE or VARIANT_FALSE.

(3) Note that DATE is stored as a double where the integer part is the date and the fraction is the time. For the DATE, 0.0 is midnight Dec 30, 1899 (i.e. midnight Jan 1, 1900 is 2.0 and Dec 4, 2001 is 37229.0). For the TIME the fraction represents the time of day moving ahead from midnight (e.g. 0.2500 is 6:00 AM, 0.400 is 9:36:00 AM). This fraction is not affected by the sign of the date and always moves 'ahead' from midnight for both positive and negative values. For example –1.4 is Dec 29, 1899 9:36:00 AM. These conversions are supported by VariantChangeTypeEx(). Generally an OVERFLOW will occur if the TO type is UI1, I1, UI2 or I2. In addition the TIME (being a fraction) will be lost on any conversion of DATE to an Integer.

(4) BSTR conversions will give a DISP_E_TYPE error if the string does not make sense for conversion to the target type. For example "1234" converts to any numeric type (except it generates OVERFLOW for UI1), "12/04/2001" converts to DATE (depending on the Locale) but not to a numeric type and "ABCD" does not convert to any other type.

(5) Conversions from non Integers to Integers must round up according to the sign if the factional part exceeds 0.5. For example 1.6 would round up to 2 and -1.6 would round 'up' to -2. In addition, the generally accepted convention is that round up will occur if the fraction equals or exceeds 0.5 **HOWEVER** client and server writers should be aware that VariantChange does not reliably adhere to the 'equals' part of this rule. Experience has shown that some floats and doubles with a fraction exactly equal to .5 will round up while others will round down. For purposes of compliance either round up or round down is acceptable when the fraction exactly equals .5.

(6) Keep in mind that Currency (CY) is stored as a scaled (fixed point x 10,000) 8 byte integer (I.e. a '_huge') With 4 digits of precision to the right of the decimal point. For example $12.34 is stored as 123400.

(7) Conversion between signed and unsigned integers **should** generate an overflow if the requested type cannot hold the value (e.g. I1=-1 should overflow if converted to UI1 and UI1 of 255 should overflow if converted to I1). However some of these conversions behave improperly when performed by ChangeVariantTypeEx. Specifically, for conversions with the same number of bits the value is NOT checked for overflow. So an I1 of –1 turns into a UI1 of 255. Similarly a UI1 value of 254 turns into an I1 value of –2. The same applies for I2 and I4. This is an incorrect behavior by ChangeVariantTypeEx. Client programs and users should be aware that most servers will exhibit this behavior since most servers will use VariantChangeTypeEx. Correcting this is **recommended** but is NOT required for OPC Compliance. Conversions between types with different numbers of bits (e.g. I1, I2, I4) are properly checked for Overflow by VariantChangeTypeEx.

(8) Loss of Precision may occur when converting between various types (e.g. converting from R8 to R4 or from R4 to I4 or I2). However no error is reported as long as there is no OVERFLOW and Quality is returned as GOOD for Reads.

(9) Note that ChangeVariantTypeEx does not handle arrays. Servers which support arrays must implement conversion logic using additional code. For arrays the **required** behavior is that if any element of the array suffers a conversion error then the first error detected is returned (DISP_E_OVER or DISP_E_TYPE). For Read the Quality is set to BAD and an empty Variant is returned. For Write, if conversion of any element fails then none of the elements are written and the first error encountered is returned.

4.2.13 Localization and LocaleID

As mentioned elsewhere in this document, the extent to which a server supports localization is up to the vendor. However certain issues require some discussion. Localization is important not just for error strings and messages. It is also potentially important for values that are read or written as strings. The formating of numbers, dates, currency, etc may all depend on the Locale. The generally expected behavior is that the Client will query the server for the Locales it supports and will chose one to use via SetLocaleID( ) or a similar function. Note that in the case of Data Access, the LocaleID of a Group can be set to be different from the 'default' LocaleID established for the server via IOPCCommon::SetLocaleID( ).

The Client should expect that the server will return strings which are translated and formatted according to the LocaleID in effect for the object (e.g. the Group) most closely associated with the data. This includes strings that are the result of converting a VARIANT to a requested datatype. Servers can easily use the function VariantChangeTypeEx() to accomplish this.

Similarly any Server OPC Object should expect that the Client will pass strings which are formatted according to the LocaleID the Client has told the server to use for that object. This includes BSTRs which arrive in VARIANTs which the server will need to convert to its native data type. Again, the server should be able to use VariantChangeTypeEx() to accomplish this.

So for example, if the client tells the server to return German formatted strings when reading from a particular object then the server can reasonably expect the client to pass German formatted strings when writing to that object.

4.2.14 Item Properties

Overview

Item Properties can be accessed by the legacy IOPCItemProperties interface or by the IOPCBrowse::GetProperties method. These interfaces can be used by clients to browse the available properties (also referred to as attributes or parameters) associated with an ITEMID and to read the current values of these properties. In some respects the functionality is similar to that provided by EnumItemAttributes and by the SyncIO Read function. It differs from these interfaces in two important respects; (a) it is intended to be much easier to use and (b) it is not optimized for efficient access to large amounts of data. Rather it is intended to allow an application to easily browse and read small amounts of additional information specific to a particular ITEMID.

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

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

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

How ‘Properties’ relate to ItemIDs.

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

access can be used for the properties if large amounts of information needs to be obtained more efficiently.

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

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

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

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

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

Typical Use

Typical Client use of these methods would be to obtain an ITEMID either by obtaining a 'child' via IOPCBrowse or via direct input to an edit box by the user. If using the legacy IOPCItemProperties, then that ITEMID would be passed to QueryAvailableProperties(). The resulting list would be presented to the user. He would select the properties he wanted to see from the list. The client would pass this set to GetItemProperties () to get a 'snapshot' of the data. Optionally the client could pass the set to LookupItemIDs and use the resulting set of ITEMIDs to create an OPCGroup to be used to repeatedly obtain the data. If using the IOPCBrowse::GetProperties, then this one method can be used to obtain this information.

Examples

This is just an example. It is not intended to impose any particular structure on any server implementation.

A typical OPC ITEMID might be FIC101.CV. This could represent the current value of a tag or function block called FIC101. This function block commonly has other properties associated with it such as Engineering Units, a loop description, etc. This function block could also have alarm limits and status, a setpoint, tuning parameters as well as documentation cross references, maintenance information, help screens, default operator displays and a limitless set of other properties. All of these properties are associated with each other by virtue of their common association with FIC101. This interface provides a convenient shortcut to accessing those related properties.

For example an Analog Input function block might have a value, hi and low limits and a status. These might have ItemIDs of F100.CV, F100.HI, F100.LO, F100.STAT. Passing any of these ItemIDs to

either IOPCItemProperties::QueryAvailableProperties or IOPCBrowse::GetProperties would likely produce the same result; i.e. a list of 4 property IDs (in addition to the 8 reserved property IDs) corresponding to each attribute. The same property IDs would likely be returned regardless of which ItemID was passed. Passing these 4 property IDs to IOPCItemProperties::LookupItemIDs or IOPCBrowse::GetProperties would also give the same result in each case in that it would return the ItemIDs mentioned here. Passing these property IDs to IOPCItemProperties::GetItemProperties or IOPCBrowse::GetProperties would also produce the same results for the values of each attribute. The one thing that would be different would be that passing any of the 8 reserved property IDs would fetch information related to the specific ItemID (i.e. would return the value, quality and timestamp associated with the specific ItemID passed as the first parameter to GetItemProperties).

An MMI package for example might use these methods to allow the user to indicate that the Hi and Lo Engineering Units values should be used to scale a bar graph representation of the value.

Note that because these associations can be 'many to many' and can also be circular, a client application would not want to automatically investigate them all.

It is NOT intended that property browsing be hierarchical.

Another similar example could be a function block such as a TIMER or COUNTER in a high end PLC where various Properties are associated with each object.

Property IDs

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

The OPC Property Sets

This is a set of property IDs that are common to many servers. Servers that provide the corresponding properties, must do so using the ID codes from this list. Symbolic equates for these properties are provided in the OPCProps.H file. (See Appendix to this document).

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

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

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

ID	DATATYPE of returned VARIANT	STANDARD DESCRIPTION
1	VT_I2	"Item Canonical DataType" (VARTYPE stored in an I2)
2	<varies>	"Item Value" (VARIANT) Note the type of value returned is as indicated by the "Item Canonical DataType" above and depends on the item. This will behave like a read from DEVICE.
3	VT_I2	"Item Quality" (OPCQUALITY stored in an I2). This will behave like a read from DEVICE.
4	VT_DATE	"Item Timestamp" (will be converted from FILETIME). This will behave like a read from DEVICE.
5	VT_I4	"Item Access Rights" (OPCACCESSRIGHTS stored in an I4)
6	VT_R4	"Server Scan Rate" In Milliseconds. This represents the fastest rate at which the server could obtain data from the underlying data source. The nature of this source is not defined but is typically a DCS system, a SCADA system, a PLC via a COMM port or network, a Device Network, etc. This value generally represents the 'best case' fastest RequestedUpdateRate which could be used if this item were added to an OPCGroup. The accuracy of this value (the ability of the server to attain 'best case' performance) can be greatly affected by system load and other factors.
7	VT_I4	"Item EU Type" (OPCEUTYPE stored in an I4) See IOPCItemAttributes for additional information
8	VT_BSTR | VT_ARRAY	"Item EUInfo"If item 7 "Item EU Type" is "Enumerated" - EUInfo will contain a SAFEARRAY of strings (VT_ARRAY | VT_BSTR) which contains a list of strings (Example: "OPEN", "CLOSE", "IN TRANSIT", etc.) corresponding to sequential numeric values (0, 1, 2, etc.)
9-99		Reserved for future OPC use

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

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

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

ID	DATATYPE of returned VARIANT	STANDARD DESCIPTION
		Properties related to the Item Value.
100	VT_BSTR	"EU Units" e.g. “DEGC” or “GALLONS”
101	VT_BSTR	"Item Description" e.g. “Evaporator 6 Coolant Temp”
102	VT_R8	"High EU" Present only for ‘analog’ data. This represents the highest value likely to be obtained in normal operation and is intended for such use as automatically scaling a bargraph display. e.g. 1400.0
103	VT_R8	"Low EU" Present only for ‘analog’ data. This represents the lowest value likely to be obtained in normal operation and is intended for such use as automatically scaling a bargraph display. e.g. -200.0
104	VT_R8	"High Instrument Range" Present only for ‘analog’ data. This represents the highest value that can be returned by the instrument. e.g. 9999.9
105	VT_R8	"Low Instrument Range" Present only for ‘analog’ data. This represents the lowest value that can be returned by the instrument. e.g. -9999.9
106	VT_BSTR	"Contact Close Label" Present only for ‘discrete' data. This represents a string to be associated with this contact when it is in the closed (non-zero) state e.g. "RUN", "CLOSE", "ENABLE", "SAFE" ,etc.
107	VT_BSTR	"Contact Open Label" Present only for ‘discrete' data. This represents a string to be associated with this contact when it is in the open (zero) state e.g. "STOP", "OPEN", "DISABLE", "UNSAFE" ,etc.
108	VT_I4	"Item Timezone" The difference in minutes between the items UTC Timestamp and the local time in which the item value was obtained.

		See the OPCGroup TimeBias property. Also see the WIN32 TIME_ZONE_INFORMATION structure.
109-199		Reserved for future OPC use. Additional IDs may be added without revising the interface ID.
		Properties Related to Alarm and Condition Values (preliminary)… **IDs 300 to 399 are reserved for use by OPC Alarms and Events.** **See the OPC Alarm and Events specification for additional information.**
300	VT_BSTR	"Condition Status" The current alarm or condition status associated with the Item e.g. "NORMAL", "ACTIVE", "HI ALARM", etc
301	VT_BSTR	"Alarm Quick Help" A short text string providing a brief set of instructions for the operator to follow when this alarm occurs.
302	VT_BSTR |VT_ARRAY	"Alarm Area List" An array of stings indicating the plant or alarm areas which include this ItemID.
303	VT_BSTR	"Primary Alarm Area" A string indicating the primary plant or alarm area including this ItemID
304	VT_BSTR	"Condition Logic" An arbitrary string describing the test being performed. e.g. "High Limit Exceeded" or "TAG.PV >= TAG.HILIM"
305	VT_BSTR	"Limit Exceeded" For multistate alarms, the condition exceeded e.g. HIHI, HI, LO, LOLO
306	VT_R8	"Deadband"
307	VT_R8	"HiHi Limit"
308	VT_R8	"Hi Limit"
309	VT_R8	"Lo Limit"
310	VT_R8	"LoLo Limit"
311	VT_R8	"Rate of Change Limit"
312	VT_R8	"Deviation Limit"
313	VT_BSTR	"Sound File" e.g. C:\MEDIA\FIC101.WAV, or .MID

314-399		Reserved for future OPC Alarms and Events use. Additional IDs may be added without revising the interface ID.
400-4999		Reserved for future OPC use. Additional IDs may be added without revising the interface ID.

NOTE the OPC Foundation reserves the right to expand this list from time to time. Clients should be prepared to deal with this.

ID Set 3 - Vendor specific Properties

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

The client should take care dealing with these vendor specific IDs - i.e. not make assumptions about them. Different vendors may not provide the same information for IDs of 5000 and above.

4.2.15 IOPCSyncIO

Interface ::Method	Source	Enable Callbacks	Group Active State	Item Active State	Server Behavior
IOPCSyncIO::Read	Cache	NA	Active	Active	The Values and Quality for the requested items are returned to the client as return values from the method. The Value and Quality are the values that the server has in cache.
IOPCSyncIO::Read	Cache	NA	Active	InActive	A Quality of OPC_QUALITY_OUT_OF_SERVICE for the requested items is returned to the client as return values from the method.
IOPCSyncIO::Read	Cache	NA	InActive	NA	A Quality of OPC_QUALITY_OUT_OF_SERVICE for the requested items is returned to the client as return values from the method.
IOPCSyncIO::Read	Device	NA	NA	NA	The Values and Quality for the requested items are returned to the client as return values from the method. The Value and Quality are the values that the server obtains from the device when this method is called. The cache of the server should be updated with the acquired value and quality.

4.2.16 IOPCASyncIO2

Interface ::Method	Source	Enable Callbacks	Group Active State	Item Active State	Server Behavior
IOPCAsyncIO2::Read	NA	NA	NA	NA	The Values and Quality for the requested items are sent to the client through the IOPCDataCallback::OnReadComplete method. The Value and Quality are the values that the server obtains from the DEVICE when this method is called. The CACHE of the server should be updated with the acquired value and quality.
IOPCAsyncIO2::Refresh2	Cache	NA	Active	Active	The Values and Quality for all the Active items in the group are sent to the client through the IOPCDataCallback::OnDataChange method. The Value and Quality are the values that the server has in cache.
IOPCAsyncIO2::Refresh2	Cache	NA	Active	InActive	The Values and Quality for all the InActive items in the group are not provided to the client. If there are no Active Items in the group then the server returns E_FAIL as the return value from the call.
IOPCAsyncIO2::Refresh2	Cache	NA	InActive	NA	The server returns E_FAIL as the return value from the call.
IOPCAsyncIO2::Refresh2	Device	NA	Active	Active	The Values and Quality for all items in the group are sent to the client through the IOPCDataCallback::OnDataChange method. The Value and Quality are the values that the server obtains from the device when this method is called. The cache of the server should be updated with the acquired values and qualities.
IOPCAsyncIO2::Refresh2	Device	NA	Active	InActive	The Values and Quality for all the InActive items in the group are not provided to the client. If there are no Active Items in the group then the server returns E_FAIL as the return value from the call.
IOPCAsyncIO2::Refresh2	Device	NA	InActive	NA	The server returns E_FAIL as the return value from the call.

4.2.17 SUBSCRIPTION via IOPCDataCallback

OnDataChange

Interface ::Method	Source	Enable Callbacks	Group Active State	Item Active State	Server Behavior
Subscription via IOPCDataCallback:: OnDataChange	NA	TRUE	Active	Active	The Value and Quality are the values that the server obtains from the device at a periodic rate sufficient to accommodate the specified UpdateRate. If the Quality has changed from the Quality last sent to the client, then the new value and new quality will be sent to the client through the IOPCDataCallback::OnDataChange method, and the cache of the server should be updated with the acquired value and quality. If the Quality has NOT changed from the Quality last sent to the client, the server should compare the acquired value for a change that exceeds the Deadband criteria. If the change in value exceeds the deadband criteria, , then the new value and new quality will be sent to the client through the IOPCDataCallback::OnDataChange method, and the cache of the server should be updated with the acquired value and quality.
Subscription via IOPCDataCallback:: OnDataChange		TRUE	Active	InActive	Server only acquires values from physical data sources for active items.
Subscription via IOPCDataCallback:: OnDataChange		TRUE	InActive	NA	Server only acquires values from physical data sources for active items that are contained in active groups.
Subscription via IOPCDataCallback:: OnDataChange	NA	FALSE	Active	Active	The Value and Quality are the values that the server obtains from the device at a periodic rate sufficient to accommodate the specified UpdateRate. If the Quality has changed from the Quality in the cache, then the cache of the server should be updated with the acquired value and quality. If the Quality has NOT changed from the Quality in the cache, the server should compare the acquired value for a change that exceeds the Deadband criteria. If the change in value exceeds the deadband criteria, , then the cache of the server should be updated with the acquired value and quality.
Subscription via IOPCDataCallback:: OnDataChange	NA	FALSE	Active	InActive	Server only acquires values from physical data sources for active items.
Subscription via IOPCDataCallback:: OnDataChange	NA	FALSE	InActive	NA	Server only acquires values from physical data sources for active items that are contained in active groups.

4.3 OPCServer Object

4.3.1 Overview

The OPCServer object is the primary object that an OPC server exposes. The interfaces that this object provides include:

- IUnknown
- IOPCServer
- IOPCBrowse (new)
- IOPCItemIO (new)
- IConnectionPointContainer

The functionality provided by each of the above interfaces is defined in this section.

NOTE: Version 1.0 of this specification listed IEnumUnkown as an interface on the OPC Server. This was an error and has been removed. The semantics of QueryInterface do not allow such an implementation. The proper way to obtain a group enumerator is through IOPCServer::CreateGroupEnumerator.

4.3.2 IUnknown

The server must provide a standard IUnknown Interface. Since this is a well defined interface it is not discussed in detail. See the OLE Programmer's reference for additional information. This interface must be provided, and all functions implemented as required by Microsoft..

4.3.3 IOPCCommon

Other OPC Servers such as alarms and events share this interface design. It provides the ability to set and query a LocaleID that would be in effect for the particular client/server session. That is, as with a Group definition, the actions of one client do not affect any other clients.

A quick reference for this interface is provided below. A more detailed discussion can be found in the OPC Overview Document.

```
HRESULT SetLocaleID (
        [in] LCID dwLcid
        );

HRESULT GetLocaleID (
        [out] LCID *pdwLcid
        );

HRESULT QueryAvailableLocaleIDs (
        [out] DWORD *pdwCount,
        [out, sizeis(, *pdwCount)] LCID **ppdwLcid
        );

HRESULT GetErrorString(
        [in] HRESULT dwError,
        [out, string] LPWSTR *ppString
        );

HRESULT SetClientName (
        [in, string] LPCWSTR szName
        );
```

4.3.4 IOPCServer

This is the main interface to an OPC server. The OPC server is registered with the operating system as specified in the Installation and Registration Chapter of this specification.

This interface must be provided, and all functions implemented as specified.

4.3.4.1 IOPCServer::AddGroup

```
HRESULT AddGroup(
        [in, string] LPCWSTR szName,
        [in] BOOL bActive,
        [in] DWORD dwRequestedUpdateRate,
        [in] OPCHANDLE hClientGroup,
        [unique, in] LONG *pTimeBias,
        [in] FLOAT * pPercentDeadband,
        [in] DWORD dwLCID,
        [out] OPCHANDLE * phServerGroup,
        [out] DWORD *pRevisedUpdateRate,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN * ppUnk
        );
```

Description

Add a Group to a Server.

Parameters	Description
szName	Name of the group. The name must be unique among the other groups created by this client. If no name is provided (szName is pointer to a NUL string) the server will generate a unique name.
bActive	FALSE if the Group is to be created as inactive. TRUE if the Group is to be created as active.
dwRequestedUpdateRate	Client Specifies the fastest rate at which data changes may be sent to OnDataChange for items in this group. This also indicates the desired accuracy of Cached Data. This is intended only to control the behavior of the interface. How the server deals with the update rate and how often it actually polls the hardware internally is an implementation detail. Passing 0 indicates the server should use the fastest practical rate. The rate is specified in milliseconds.
hClientGroup	Client provided handle for this group. [refer to description of data types, parameters, and structures for more information about this parameter]
pTimeBias	Pointer to Long containing the initial TimeBias (in minutes) for the Group. Pass a NULL Pointer if you wish the group to use the default system TimeBias. See discussion of

	TimeBias in General Properties Section See **Comments** below.
pPercentDeadband	The percent change in an item value that will cause a subscription callback for that value to a client. This parameter only applies to items in the group that have dwEUType of Analog. [See discussion of Percent Deadband in General Properties Section]. A NULL pointer is equivalent to 0.0.
dwLCID	The language to be used by the server when returning values (including EU enumeration's) as text for operations on this group. This could also include such things as alarm or status conditions or digital contact states.
phServerGroup	Place to store the unique server generated handle to the newly created group. The client will use the server provided handle for many of the subsequent functions that the client requests the server to perform on the group.
pRevisedUpdateRate	The server returns the value it will actually use for the UpdateRate which may differ from the RequestedUpdateRate. Note that this may also be slower than the rate at which the server is internally obtaining the data and updating the cache. In general the server should 'round up' the requested rate to the next available supported rate. The rate is specified in milliseconds. Server returns HRESULT of OPC_S_UNSUPPORTEDRATE when it returns a value in revisedUpdateRate that is different than RequestedUpdateRate.
riid	The type of interface desired (e.g. IID_IOPCItemMgt)
ppUnk	Where to store the returned interface pointer. NULL is returned for any FAILED HRESULT.

Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_DUPLICATENAME	Duplicate name not allowed.
OPC_S_UNSUPPORTEDRATE	Server does not support requested rate, server returns the rate that it can support in the revised update rate.
E_NOINTERFACE	The interface (riid) asked for is not supported by the server.

Behavior

A Group is a logical container for a client to organize and manipulate data items.

The server will create a group object, and return a pointer to the interface requested by the client. If the client requests an optional interface that the server does not support, the server is expected to return an error indicating the interface is not supported.

The requested update rate / revised update rate behavior should be deterministic between client / server sessions. The client expects that for the same server configuration or workspace; adding a group with a requested update rate will always result in the same RevisedRate independent of the number of clients or items that have been added.

Comments

The expected object lifetime behavior is as follows. Even if all the interfaces are released, the group will not be deleted until RemoveGroup is called. One way for the server to implement this is to assign the group an initial reference count of 2; one for the 'Add' and one for the Interface that was created. However, clients should not make assumptions about the Group's reference count.

The client should not call RemoveGroup without releasing all interfaces for the group. The client should also not release the server without removing all private groups.

Since the server is the 'container' for the groups it is permissible for the server to forcibly remove any remaining groups at the time all of the server interfaces are released. (This should not be necessary for a well-behaved client).

See also the CreateGroupEnumerator function.

The level of localization supported (dwLCID) is entirely server specific. Servers, which do not support dynamic localization, can ignore this parameter.

4.3.4.2 IOPCServer::GetErrorString

```
HRESULT GetErrorString(
        [in] HRESULT dwError,
        [in] LCID dwLocale,
        [out, string] LPWSTR *ppString
        );
```

Description

Returns the error string for a server specific error code.

Parameters	Description
dwError	A server specific error code that the client application had returned from an interface function from the server, and for which the client application is requesting the server's textual representation.
dwLocale	The locale for the returned string.
ppString	Pointer to pointer where server supplied result will be saved

Return Codes

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

Comments

This is essentially the same function as is found in the newer IOPCCommon.

Note that if this method is called on a remote server, an RPC error may result. For this reason it is probably good practice for the client to attempt to call a local Win32 function if this function fails.

The expected behavior is that this will include handling of Win32 errors as well (such as RPC errors).

The Client must free the returned string.

It is recommended that the server put any OPC specific strings into an external resource to simplify translation.

To get the default value for the system, the dwLocale should be LOCALE_SYSTEM_DEFAULT.

4.3.4.3 IOPCServer::GetGroupByName

```
HRESULT GetGroupByName(
        [in, string] LPCWSTR szName,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN * ppUnk
        );
```

Description

Given the name of a private group (created earlier by the same client), return an additional interface pointer.

Parameters	Description
szName	The name of the group. That is the group must have been created by the caller.
riid	The type of interface desired for the group (e.g. IOPCItemMgt)
ppUnk	Pointer to where the group interface pointer should be returned. NULL is returned for any HRESULT other than S_OK.

Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.
E_NOINTERFACE	The interface(riid) asked for is not supported by the server.

Comments

This function can be used to reconnect to a private group for which all interface pointers have been released.

The client must release the returned interface when it is done with it.

If needed, the client can obtain the hServerGroup Handle via IOPCGroupStateMgt::GetState.

4.3.4.4 IOPCServer::GetStatus

HRESULT GetStatus(
[out] OPCSERVERSTATUS ** ppServerStatus
);

Description

Returns current status information for the server.

Parameters	Description
ppServerStatus	Pointer to where the OPCSERVERSTATUS structure pointer should be returned. The structure is allocated by the server.

Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.

Comments

The OPCSERVERSTATUS is described later in this specification.

Client must free the structure as well as the VendorInfo string within the structure.

Periodic calls to GetStatus would be a good way for the client to determine that the server is still connected and available.

4.3.4.5 IOPCServer::RemoveGroup

HRESULT RemoveGroup(
[in] OPCHANDLE hServerGroup,
[in] BOOL bForce
);

Description

Deletes the Group

Parameters	Description
hServerGroup	Handle for the group to be removed
bForce	Forces deletion of the group even if references are outstanding

Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.
OPC_S_INUSE	Was not removed because references exist. Group will be marked as deleted, and will be removed automatically by the server when all references to this object are released.

Comments

A group is not deleted when all the client interfaces are released, since the server itself maintains a reference to the group. The client may still call GetGroupByName after all the interfaces have been released. RemoveGroup() causes the server to release it's `last' reference to the group, which results in the group being truly deleted.

A well behaved COM client should call this function only after either completing or canceling all outstanding async operations, unadvising all callbacks for the group, and releasing all interfaces on the group. It is very strongly recommended that clients follow this guideline to insure reliable operation.

For historical reasons the following rules also exist. However newly coded clients should not rely on the following information but should follow standard COM practice as noted above.

If interfaces still exist, Remove group will mark the group as 'deleted'. Any further calls to this group via these interfaces will return E_FAIL. When all the interfaces are released, the group will actually be deleted by the server.

The bForce flag is supported only for backwards compatibility. It is very strongly recommended that client software avoid use of this flag. Use of the bForce flag is subject to the following rules.

The client must complete or cancel any outstanding ASYNC operations prior to calling RemoveGroup with bForce true.

The client must unAdvise any Callbacks prior to calling RemoveGroup with bForce true.

If bForce is TRUE then the group is deleted unconditionally even if client references (interfaces) still exist. It is recommended that the server call CoDisconnectObject() to allow proper cleanup of proxy/stub connections. Subsequent use of such interfaces by the client will result in an access violation. After a return from RemoveGroup any use, by the client, of a previously requested Interface is not allowed.

4.3.4.6 IOPCServer::CreateGroupEnumerator

```
HRESULT CreateGroupEnumerator(
        [in] OPCENUMSCOPE dwScope,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
        );
```

Description

Create various enumerators for the groups provided by the Server.

Parameters	Description
dwScope	Indicates the class of groups to be enumerated OPC_ENUM_PRIVATE_CONNECTIONS or OPC_ENUM_PRIVATE enumerates all of the private groups created by the client OPC_ENUM_ALL_CONNECTIONS or OPC_ENUM_ALL enumerates all private groups .
riid	The interface requested. This must be IID_IEnumUnknown or IID_IEnumString.
ppUnk	Where to return the interface. NULL is returned for any HRESULT other than S_OK or S_FALSE.

NOTE: Version 1.0 of this specification described slightly different behavior for enumerating connected vs non-connected groups. However this behavior has been found to be difficult or impossible to implement in practice. The description here represents a simplification of this behavior. It is recommended that use of OPC_ENUM_PRIVATE_CONNECTIONS, OPC_ENUM_ALL_CONNECTIONS be avoided by clients.

HRESULT Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.
S_FALSE	There is nothing to enumerate (there are no groups which satisfy the request). However an empty Enumerator is still returned and must be released. Note: In previous versions of the spec there has been some ambiguity about the behavior in the case of S_FALSE. For this reason, it is recommended that when S_FALSE is returned by the server, clients test the returned interface pointer for NULL prior to calling Release on it.
E_NOINTERFACE	The interface (riid) asked for is not supported by the server.

Comments

Connected means an interface pointer exists.

IEnumUnknown creates an additional interface pointer to each group in the enumeration (even if the client already has a connection to the group). In the case of IEnumUnknown (per the COM specification) the client must also release all of the returned IUnknown pointers when he is done with them.

4.3.5 IConnectionPointContainer (on OPCServer)

This interface provides access to the connection point for IOPCShutdown.

The general principles of ConnectionPoints are not discussed here as they are covered very clearly in the Microsoft Documentation. The reader is assumed to be familiar with this technology. OPC DA Compliant Servers, from OPC 2.0 and greater, are REQUIRED to support this interface.

Likewise the details of the IEnumConnectionPoints, IConnectionPoint and IEnumConnections interfaces are well defined by Microsoft and are not discussed here.

Note: OPC Compliant servers are not required to support more than one connection between each Server and the Client. Given that servers are client specific entities it is expected that a single connection will be sufficient for virtually all applications. For this reason (as per the COM Specification) the EnumConnections method for IConnectionPoint interface for the IOPCShutdown is allowed to return E_NOTIMPL.

4.3.5.1 IConnectionPointContainer::EnumConnectionPoints

```
HRESULT EnumConnectionPoints(
        IEnumConnectionPoints **ppEnum
    );
```

Description

Create an enumerator for the Connection Points supported between the OPC Group and the Client.

Parameters	Description
ppEnum	Where to save the pointer to the connection point enumerator. See the Microsoft documentation for a discussion of IEnumConnectionPoints.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
For other codes see the OLE programmers reference	

Comments

OPCServers must return an enumerator that includes IOPCShutdown. Additional vendor specific callbacks are also allowed.

4.3.5.2 IConnectionPointContainer:: FindConnectionPoint

```
HRESULT FindConnectionPoint(
        REFIID riid,
        IConnectionPoint **ppCP
        );
```

Description

Find a particular connection point between the OPC Server and the Client.

Parameters	Description
ppCP	Where to store the Connection Point. See the Microsoft documentation for a discussion of IConnectionPoint.
riid	The IID of the Connection Point. (e.g. IID_IOPCShutdown)

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
For other codes see the OLE programmers reference	

Comments

OPCServers must support IID_IOPCShutdown. Additional vendor specific callbacks are also allowed.

4.3.6 IOPCBrowse

IOPCBrowse interace provides improved methods for browsing the server address space and for obtaining the item properties. The methods in this interface are designed to mirror the corresponding methods in the XML-DA interface.

4.3.6.1 IOPCBrowse:: Browse

```
HRESULT Browse(
    [in, string] LPWSTR szItemID,
    [in,out, string] LPWSTR * pszContinuationPoint,
    [in] DWORD dwMaxElementsReturned,
    [in] OPCBROWSEFILTER dwBrowseFilter,

    [in, string] LPWSTR szElementNameFilter,
    [in, string] LPWSTR szVendorFilter,
    [in] BOOL bReturnAllProperties,
    [in] BOOL bReturnPropertyValues,
    [in] DWORD dwPropertyCount,
    [in, size_is(dwPropertyCount)] DWORD * pdwPropertyIDs,
    [out] BOOL * pbMoreElements,
    [out] DWORD * pdwCount,
    [out, size_is(,*pdwCount)] OPCBROWSEELEMENT ** ppBrowseElements
);
```

Description

Browses a single branch of the address space and returns zero or more OPCBROWSEELEMENT structures.

It is assumed that the underlying server address space is hierarchical. A flat space will always be presented to the client as not having children. A hierarchical space can be presented to the client as either not having children or having children.

A hierarchical presentation of the server address space would behave much like a file system, where the directories are the branches or paths, and the files represent the leaves or items. For example, a server could present a control system by showing all the control networks, then all of the devices on a selected network, and then all of the classes of data within a device, then all of the data items of that class. A further breakdown into vendor specific ‘Units’ and ‘Lines’ might be appropriate for a BATCH system.

The browse position is initially set to the ‘root’ of the address space. On subsequent calls, the client may choose to browse from the continuation point.

This browse can also be filtered by a vendor specific filter string.

Parameters	Description
szItemID	The name of the branch in the hierarchy to browse. If the root branch is to be browsed then a NUL string is passed.
pszContinuationPoint	If this is a secondary call to Browse, the previous call might have returned a Continuation Point where the Browse can restart from. Clients must pass a NUL string in the initial call to Browse. This is an opaque

	value, which the server creates. A Continuation Point will be returned if a Server does support Continuation Point, and the reply is larger than **dwMaxElementsReturned**. The Continuation Point will allow the Client to resume the Browse from the previous completion point..
dwMaxElementsReturned	Server must not return any more elements than this value.If the server supports Continuation Points, then the Server may return a Continuation Point at a value less than **dwMaxElementsReturned**. If the server does not support Continuation Points, and more than **dwMaxElementsReturned** are available, then the Server shall return the first **dwMaxElementsReturned** elements and set the **pbMoreElements** parameter to TRUE. If **dwMaxElementsReturned** is 0 then there is no client side restriction on the number of returned elements.
dwBrowseFilter	An enumeration {All, Branch, Item} specifying which subset of browse elements to return. See the table in the comments section below to determine which combination of bits in **OPCBROWSEELEMENT::dwFlagValue** are returned for each value in **dwBrowseFilter**.
szElementNameFilter	A wildcard string that conforms to the Visual Basic LIKE operator, which will be used to filter Element names. A NUL String implies no filter.
szVendorFilter	A server specific filter string. This is entirely free format and may be entered by the user via an EDIT field. A pointer to a NUL string indicates no filtering.
bReturnAllProperties	Server must return all properties which are available for each of the returned elements. If true, **pdwPropertyIDs** is ignored.
bReturnPropertyValues	Server must return the property values in addition to the property names.
dwPropertyCount	The size of the **pdwPropertyIDs** array.
pdwPropertyIDs	An array of Property IDs to be returned with each element. if **bReturnAllProperties** is true, **pdwPropertyIDs** is ignored and all properties are returned.
pbMoreElements	If the Server does not support a Continuation Point then the server will set **pbMoreElements** to True if there are more elements than **dwMaxElementsReturned**.
pdwCount	The size of the returned **ppBrowseElements** array.
ppBrowseElements	Array of **OPCBROWSEELEMENT** returned by the Server.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The operation succeeded but there are one or more errors in **ppBrowseElements**. Refer to individual error returns for more information.
OPC_E_UNKNOWNITEMID	The item is not known to the server.
OPC_E_INVALIDITEMID	The szItemID does not conform to the server's syntax.
OPC_E_INVALIDFILTER	The filter string (element or vendor) is not valid.
OPC_E_INVALIDCONTINUATIONPOINT	The specified continuation point is not valid.
E_OUTOFMEMORY	The operation could not be executed due to memory limitations.
E_INVALIDARG	An invalid argument was passed.
E_FAIL	The operation failed.

Comments

See section 6.7.10 for a description of the OPCBROWSEELEMENT structure.

If the level specified szItemID is valid, but does not have any children, then the Browse will succeed, but the result will be a zero length **ppBrowseElements** array.

If the filter criteria result in an empty result, then the Browse will still succeed.

The following table describes the relationship between the possible values of **OPCBROWSEELEMENT::dwFlagValue** and the value of **dwBrowseFilter** supplied in the request. The Flag Values, OPC_BROWSE_HASCHILDREN and OPC_BROWSE_ISITEM are abbreviated as HasChildren and IsItem:

dwFlagValue		Description	dwBrowseFilter		
IsItem	HasChildren		All	Branch	Item
0	0	An empty branch	●	●	
0	1	A branch that has children, or possibly has children.	●	●	
1	0	An item that is not a branch	●		●
1	1	A branch that has children, or possibly has children, that is also an item	●	●	●

4.3.6.2 IOPCBrowse::GetProperties

```
HRESULT GetProperties(
    [in] DWORD dwItemCount,
    [in, string, size_is(dwItemCount)] LPWSTR * pszItemIDs,
    [in] BOOL bReturnPropertyValues,
    [in] DWORD dwPropertyCount,
    [in, size_is(dwPropertyCount)] DWORD * pdwPropertyIDs,
    [out, size_is(,dwItemCount)] OPCITEMPROPERTIES **ppItemProperties
);
```

Description

Returns an array of OPCITEMPROPERTIES, one for each ItemID.

Parameters	Description
dwItemCount	The size of the **pszItemIDs** and **ppItemProperties** arrays.
pszItemIDs	The array of item IDs for which the caller wants properties.
bReturnPropertyValues	Server must return the property values in addition to the property names.
dwPropertyCount	The size of the **pdwPropertyIDs** array. 0 if NULL is passed in for **pdwPropertyIDs**
pdwPropertyIDs	Array of Property IDs to return data for. If NULL, then all available Properties will be returned.
ppItemProperties	Array of OPCITEMPROPERTIES returned by the Server.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The operation succeeded but there are one or more errors in **ppItemProperties**. Refer to individual error returns for more information. S_FALSE is also returned, if hrErrorID of one OPCITEMPROPERTIES structure is S_FALSE.
E_OUTOFMEMORY	Not enough Memory.
E_INVALIDARG	An invalid argument was passed.
E_FAIL	The function failed.

Comments

See section 6.7.8 for a description of the OPCITEMPROPERTIES structure.

4.3.7 IOPCItemIO

The purpose of this interface is to provide an extremely easy way for simple applications to obtain OPC data. Programmers should be aware that in most servers, the design of the Group based OPC interfaces will provide much better performance than this interface. In terms of performance, the user of this interface should assume that it will behave as if he were to create a group, add the items, perform a single read or write and then delete the group.

A second purpose of this interface is to provide a method for writing timestamp and quality information into servers that support this functionality.

4.3.7.1 IOPCItemIO::Read

```
HRESULT Read (
   [in]  DWORD dwCount,
   [in, sizeis(dwCount)] LPCWSTR  *pszItemIDs,
   [in, sizeis(dwCount)] DWORD *pdwMaxAge,
   [out, sizeis(,dwCount)] VARIANT  **ppvValues,
   [out, sizeis(,dwCount)] WORD  **ppwQualities,
   [out, sizeis(,dwCount)] FILETIME  **ppftTimeStamps,
   [out, sizeis(,dwCount)] HRESULT  **ppErrors
   );
```

Description

Reads one or more values, qualities and timestamps for the items specified. This is functionally similar to the IOPCSyncIO::Read method.

Parameters	Description
dwCount	The number of items to be read.
pszItemIDs	A list of fully qualified ItemIDs. These might be obtained from IOPCBrowse::GetProperties
dwMaxAge	An array of "staleness" for each item, requested in milliseconds. The server will calculate, for each requested item, the number of milliseconds between "now" and the timestamp on each item. For each item that has not been updated within the last MaxAge milliseconds, 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 0xFFFFFFFF 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. 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 a DWORD of milliseconds, the largest MaxAge value would be approximately 49.7 days).
ppvValues	A pointer to an array of VARIANTs in which the results are returned. Note that the array and its contained variants must be freed by the client after receipt.
ppwQualities	An array of Words in which to store the Quality of each result. Note that these must be freed by the client.
ppftTimeStamps	An array of FILETIMEs in which to store the timestamps of each result. Note that these must be freed by the client.
ppErrors	Array of HRESULTs indicating the success of the individual item reads. The errors correspond to the ItemIDs passed in pszItemIDs. This indicates whether the read succeeded in obtaining a defined value, quality and timestamp. NOTE any FAILED error code indicates that the corresponding Value, Quality and Time stamp are UNDEFINED. Note that these must be freed by the client.

Return Codes

Return Code	Description
S_OK	The operation succeeded.
S_FALSE	The operation succeeded but there are one or more errors in ppErrors. Refer to individual error returns for more information.
E_INVALIDARG	An invalid argument was passed. (e.g. dwCount=0)
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory

ppErrorCodes

Return Code	Description
S_OK	Successful Read.
E_FAIL	The Read failed for this item
OPC_E_BADRIGHTS	The item is not readable
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
OPC_E_INVALIDITEMID	The ItemID doesn't conform to the server's syntax.
S_xxx E_xxx	S_xxx - Vendor specific information can be provided if this item quality is other than GOOD. E_xxx - Vendor specific error if this item cannot be accessed. These vendor specific codes can be passed to GetErrorString().

Comments

The MaxAge will only be tested upon receipt of this call. For example, if 3 items are requested with a MaxAge of 1000 milliseconds and two of the three items are within the appropriate range, then the third item must be obtained from the underlying device. Once this item is obtained, it may take longer than the requested MaxAge, then the three items will be packaged and returned to the client. In this case the original two items that were within the MaxAge threshold are now "stale". The test for MaxAge will not be re-evaluated and therefore the two "stale" items will be returned with the items obtained directly from the device. This functionality is in place to prevent the server from recursively attempting to obtain the values.

Some servers may return always the actual value, if DEVICE = CACHE.

If the HRESULT is S_OK, then ppErrors can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is S_FALSE, then ppErrors will indicate which the status of each individual Item Read.

If the HRESULT is any FAILED code then the contents of all OUT parameters including ppErrors is undefined.

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

For any FAILED (E_) result in ppError the client may not make use of the corresponding Value, Quality and Timestamp except to call VariantClear before destroying the VARIANT. When returning an E_ result in ppError, the Server must set the corresponding Value's VARIANT to VT_EMPTY so that it can be marshaled properly and so that the client can execute VariantClear on it.

Be sure to call **VariantClear()** on the variants in the Values before freeing the memory containing them.

The client is responsible for freeing all memory associated with the out parameters.

4.3.7.2 IOPCItemIO::WriteVQT

```
HRESULT WriteVQT (
  [in]  DWORD dwCount,
  [in, sizeis(dwCount)] LPCWSTR  *pszItemIDs,
  [in, sizeis(dwCount)] OPCITEMVQT *pItemVQT,
  [out, sizeis(,dwCount)] HRESULT  **ppErrors
  );
```

Description

Writes one or more values, qualities and timestamps for the items specified. This is functionally similar to the IOPCSyncIO2::WriteVQT except that there is no associated group. If a client attempts to write VQ, VT, or VQT it should expect that the server will write them all or none at all.

Parameters	Description
dwCount	The Number of ItemIDs to be written.
pszItemIDs	The list of ItemIDs to be written.
pItemVQT	The list of OPCITEMVQT structures. Each structure will potentially contain a value, quality and timestamp to be written to the corresponding ItemID. If the value is equal to VT_EMPTY, then no value should be written. There is a Boolean value associated with each Quality and Timestamp. The Boolean is an indicator as to whether the Quality or Timestamp should be written for the corresponding ItemID. True indicates write, while false indicates do not write.
ppErrors	The list of errors resulting from the write (1 per item). Note that these must be freed by the client.

Return Codes

Return Code	Description
S_OK	The operation succeeded.
S_FALSE	The operation succeeded but there are one or more errors in ppErrors. Refer to individual error returns for more information.
E_INVALIDARG	An invalid argument was passed.
OPC_E_NOTSUPPORTED	If a client attempts to write any value, quality, timestamp combination and the server does not support the requested combination (which could be a single quantity such as just timestamp), then the server will not perform any write and will return this error code.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory

ppErrorCodes

Return Code	Description
S_OK	The function was successful.
E_FAIL	The function was unsuccessful.
OPC_S_CLAMP	The value was accepted but was clamped.
OPC_E_RANGE	The value was out of range.
OPC_E_BADTYPE	The passed data type cannot be accepted for this item
OPC_E_BADRIGHTS	The item is not writeable
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space
OPC_E_INVALIDITEMID	The item ID doesn't conform to the server's syntax.
E_xxx S_xxx	Vendor specific errors may also be returned. Descriptive information for such errors can be obtained from GetErrorString.

Comments

If the HRESULT is S_OK, then ppErrors can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then the contents of ppErrors is undefined.

As an alternative to OPC_E_BADTPYE it is acceptable for the server to return any FAILED error returned by VariantChangeType or VariantChangeTypeEx.

Note: There is no way to validate the writing of the timestamp. If writing the timestamp is supported by the server, then the timestamp will be updated on the device as opposed to the cache. Writing timestamps is generally expected to be used for values which are in some sort of manual override mode or for values which are in some form of holding register. In general it is not useful to write timestamps for values which are actually being generated or scanned by the device since the device will generally re-stamp the data each time it is generated or scanned.

4.4 OPCGroup Object

The OPCGroup object is the object that an OPC server delivers to manage a collection of items. The interfaces that this object provides include:

- IUnknown
- IOPCGroupStateMgt
- IOPCGroupStateMgt2 (new)
- IOPCItemMgt
- IOPCSyncIO
- IOPCSyncIO2 (new)
- IOPCAsyncIO2
- IOPCAsyncIO3 (new)
- IOPCItemDeadbandMgt (new)
- IOPCItemSamplingMgt (new /optional)
- IConnectionPointContainer

The functionality provided by each of these interfaces is defined in this section.

This section also identifies the interfaces required to be implemented to support the OLE mechanism for delivering a COM interface.

4.4.1 General Properties

The OPCGroup has certain general properties and behaviors which affect the operation of the Interfaces and Methods. These are discussed here in order to minimize duplication.

4.4.1.1 Name

Each group has a name. The name must be unique among the other groups that belong to that clientThe client can change the name of a group.

Group names are Case Sensitive. Group1 would be different from group1.

4.4.1.2 Cached data

The methods below allow the client to specify that some operations can be performed on 'CACHE' or 'DEVICE'. It is expected that most servers will implement some sort of CACHE. As discussed earlier these terms are simply part of the interface definition. The way the functions described below behave differs slightly based on which source is specified. The actual details of the implementation of this functionality are up to the server vendor. In most cases, access to CACHE data is expected to be 'fast' while access to the 'DEVICE is expected to be 'slow' but more accurate. CACHE data is affected by the Active state of the group and the items in the group while DEVICE data is not. *Note again that although we sometimes make suggestions, this specification does not dictate any particular implementation or performance.*

4.4.1.3 Active

Groups and Items within Groups have an Active Flag. The active state of the group is maintained separately from the active state of the items. Changing the state of the group 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 is 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 Groups 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 groups and items as a more efficient alternative to adding and removing entire groups 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 earlier in this document for a quick overview of the behavior of a client and server with respect to the active state of a group and items.

OnDataChange within the client's address space can be called whenever any active data item in a active group 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 group that changed (this will be discussed more in later sections).

4.4.1.4 Update Rate

The client can specify an 'update rate' for each group. 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 group may have a different sampling rate. The sampling rate associated with individual items does not effect the callback period. 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 sampling rate than the update rate of the group, 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 group and the value and/or quality change more often than the group 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 for more detail.

4.4.1.5 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 OPCGroup 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 OPCServer 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 timezone in which the data was collected (which may occasionally be different from the timezone in which either the client or server is running). The default TimeBias for the group (if a NULL pointer is passed to AddGroup) will be that of the system in which the group 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 group 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.

4.4.1.6 Percent Deadband

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

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

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

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

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

4.4.1.7 ClientHandle

This handle will be returned in any callback. This allows the client to identify the group 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.

4.4.1.8 Reading and Writing Data

There are basically six ways to get data into a client.

- IOPCSyncIO::Read (from cache or device)
- IOPCAsyncIO2::Read (from device)
- IOPCCallback::OnDataChange() (exception based) which can also be triggered by IOPCAsyncIO2::Refresh.
- IOPCItemIO::Read (from cache or device as determined by "staleness" of cache data)
- IOPCSyncIO2::ReadMaxAge (from cache or device as determined by "staleness" of cache data)
- IOPCAsyncIO3::ReadMaxAge (from cache or device as determined by "staleness" of cache data)

In general the six methods operate independently without 'side effects' on each other. It is intended that newer applications use OnDataChange, IOPCSyncIO2 or IOPCAsynchIO3 for most reads.

There are five ways to write data out:

- IOPCSyncIO::Write
- IOPCAsyncIO2::Write
- IOPCItemIO::WriteVQT
- IOPCSyncIO2::WriteVQT
- IOPCAsyncIO3::WriteVQT

4.4.2 IOPCItemMgt

IOPCItemMgt allows a client to add, remove and control the behavior of items is a group.

4.4.2.1 IOPCItemMgt::AddItems

```
HRESULT AddItems(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCITEMDEF * pItemArray,
        [out, size_is(,dwCount)] OPCITEMRESULT ** ppAddResults,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Add one or more items to a group.

Parameters	Description
dwCount	The number of items to be added
pItemArray	Array of OPCITEMDEFs. These tell the server everything it needs to know about the item including the access path, definition and requested datatype
ppAddResults	Array of OPCITEMRESULTs. This tells the client additional information about the item including the server assigned item handle and the canonical datatype.
ppErrors	Array of HRESULTs. This tells the client which of the items was successfully added. For any item which failed it provides a reason.

HRESULT Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (e.g dwCount=0).
S_OK	The operation succeeded.
S_FALSE	The operation completed with one or more errors. Refer to individual error returns for failure analysis.

ppErrors Return Codes

Return Code	Description
S_OK	The function was successful for this item.
OPC_E_INVALIDITEMID	The ItemID is not syntactically valid
OPC_E_UNKNOWNITEMID	The ItemID is not in the server address space
OPC_E_BADTYPE	The requested data type cannot be returned for this item (See comment)
E_FAIL	The function was unsuccessful.
OPC_E_UNKNOWNPATH	The item's access path is not known to the server.

Comments

It is acceptable to add the same item to the group more than once. This will generate a 2nd item with a unique ServerHandle.

Any FAILED code in ppErrors indicates that the corresponding item was NOT added to the group and that the corresponding OPCITEMRESULT will not contain useful information.

As an alternative to OPC_E_BADTPYE it is acceptable for the server to return any FAILED error returned by VariantChangeType or VariantChangeTypeEx.

The server provided item handle will be unique within the group, but may not be unique across groups. The server is allowed to ‘reuse’ the handles of deleted items.

The client needs to free all of the memory associated with the OPCITEMRESULTs including the BLOB.

If the server supports the BLOB it will return an updated BLOB in the OPCITEMRESULTs. This BLOB may differ in both content and size from the one passed by the client in OPCITEMDEF.

Note that if an Advise is active, the client will begin receiving callbacks for active items. This can occur very quickly, perhaps even before the client has time to process the returned results. The client must be designed to deal with this. One simple solution is for the client to clear the Active state of the group while doing AddItems and to restore it after the AddItems is completed and the results are processed.

4.4.2.2 IOPCItemMgt::ValidateItems

```
HRESULT ValidateItems(
    [in] DWORD dwCount,
    [in, size_is(dwCount)] OPCITEMDEF * pItemArray,
    [in] BOOL bBlobUpdate,
    [out, size_is(,dwCount)] OPCITEMRESULT ** ppValidationResults,
    [out, size_is(,dwCount)] HRESULT ** ppErrors
    );
```

Description

Determines if an item is valid (could it be added without error). Also returns information about the item such as canonical datatype. Does not affect the group in any way.

Parameters	Description
dwCount	The number of items to be validated
pItemArray	Array of OPCITEMDEFs. These tell the server everything it needs to know about the item including the access path, definition and requested datatype
bBlobUpdate	If non-zero (and the server supports Blobs) the server should return updated Blobs in OPCITEMRESULTs. If zero (False) the server will not return Blobs in OPCITEMRESULTs.
ppValidationResults	Array of OPCITEMRESULTs. This tells the client additional information about the item including the canonical datatype.
ppErrors	Array of HRESULTs. This tells the client which of the items was successfully validated. For any item which failed it provides a reason.

HRESULT Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid (e.g dwCount=0).
S_OK	The operation succeeded.
S_FALSE	The operation completed with one or more errors. Refer to individual error returns for failure analysis.

ppErrors Codes

Return Code	Description
S_OK	The function was successful for this item.
OPC_E_INVALIDITEMID	The ItemID is not syntactically valid
OPC_E_UNKNOWNITEMID	The ItemID is not in the server address space
OPC_E_BADTYPE	The requested data type cannot be returned for this item (See comment)
E_FAIL	The function was unsuccessful for this item.
OPC_E_UNKNOWNPATH	The item's access path is not known to the server.

Comments

The client needs to free all of the memory associated with the OPCITEMRESULTs including the BLOB.

As an alternative to OPC_E_BADTPYE it is acceptable for the server to return any FAILED error returned by VariantChangeType or VariantChangeTypeEx.

4.4.2.3 IOPCItemMgt::RemoveItems

```
HRESULT RemoveItems(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Removes (deletes) items from a group. Basically this is the reverse of AddItems.

Parameters	Description
dwCount	Number of items to be removed
phServer	Array of server items handles. These were returned from AddItem.
ppErrors	Array of HRESULTs. Indicates which items were successfully removed.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function completed with one or more errors. See the ppErrors to determine what happened
E_FAIL	The function was unsuccessful.
E_INVALIDARG	An argument to the function was invalid (e.g dwCount=0).

ppError Codes

Return Code	Description
S_OK	The corresponding item was removed.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.

Comments

Adding and removing items from a group does not affect the address space of the server or physical device. It simply indicates whether or not the client is interested in those particular items.

Items are not really objects in the custom interface (do not have interfaces), and there is no concept of a reference count for items. The client should insure that no further references are made to deleted items.

.

4.4.2.4 IOPCItemMgt::SetActiveState

```
HRESULT SetActiveState(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in] BOOL bActive,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Sets one or more items in a group to active or inactive. This controls whether or not valid data can be obtained from Read CACHE for those items and whether or not they are included in the OnDataChange subscription to the group.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
bActive	TRUE if items are to be activated. FALSE if items are to be deactivated.
ppErrors	Array of HRESULTs. Indicates which items were successfully affected.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function completed with one or more errors. See the ppErrors to determine what happened
E_INVALIDARG	An argument to the function was invalid (e.g dwCount=0).
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.

Comments

Deactivating items will not result in a callback (since by definition callbacks do not occur for inactive items). Activating items which were previously inactive must result in a callback at the next UpdateRate period

4.4.2.5 IOPCItemMgt::SetClientHandles

```
HRESULT SetClientHandles(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in, size_is(dwCount)] OPCHANDLE * phClient,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Changes the client handle for one or more items in a group.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
phClient	Array of new Client item handles to be stored. The Client handles do not need to be unique.
ppErrors	Array of HRESULTs. Indicates which items were successfully affected.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function completed with one or more errors. See the itemResults to determine what happened
E_INVALIDARG	An argument to the function was invalid (e.g dwCount=0).
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.

Comments

In general, it is expected that clients will set the client handle when the item is added and not change it later.

4.4.2.6 IOPCItemMgt::SetDatatypes

```
HRESULT SetDatatypes(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in, size_is(dwCount)] VARTYPE * pRequestedDatatypes,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Changes the requested data type for one or more items in a group.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
pRequestedDatatypes	Array of new Requested Datatypes to be stored.
ppErrors	Array of HRESULT's. Indicates which items were successfully affected.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function completed with one or more errors. See the itemResults to determine what happened
E_INVALIDARG	An argument to the function was invalid (e.g dwCount=0).
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.
OPC_E_BADTYPE	The requested datatype cannot be supported for this item. (See comment). The previous requested type is left unchanged.

Comments

In general, it is expected that clients will set the requested datatype when the item is added and not change it later.

As an alternative to OPC_E_BADTPYE it is acceptable for the server to return any FAILED error returned by VariantChangeType or VariantChangeTypeEx.

4.4.2.7 IOPCItemMgt::CreateEnumerator

```
HRESULT CreateEnumerator(
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
        );
```

Description

Create an enumerator for the items in the group.

Parameters	Description
riid	The interface requested. At this time the only supported OPC interface is IID_IEnumOPCItemAttributes although vendors can add their own extensions to this.
ppUnk	Where to return the interface. NULL is returned for any HRESULT other than S_OK

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	There is nothing to enumerate (There are no items in the group).
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid (e.g. a bad riid parameter was passed.)
E_FAIL	The function was unsuccessful.

Comments

The client must release the returned interface pointer when it is done with it.

4.4.3 IOPCGroupStateMgt

IOPCGroupStateMgt allows the client to manage the overall state of the group. Primarily this allows changes to the update rate and active state of the group.

4.4.3.1 IOPCGroupStateMgt::GetState

```
HRESULT GetState(
    [out] DWORD * pUpdateRate,
    [out] BOOL * pActive,
    [out, string] LPWSTR * ppName,
    [out] LONG * pTimeBias,
    [out] FLOAT * pPercentDeadband,
    [out] DWORD * pLCID,
    [out] OPCHANDLE * phClientGroup,
    [out] OPCHANDLE * phServerGroup
    );
```

Description

Get the current state of the group.

Parameters	Description
pUpdateRate	The current update rate. The Update Rate is in milliseconds
pActive	The current active state of the group.
ppName	The current name of the group
pTimeBias	The TimeZone Bias of the group (in minutes)
pPercentDeadband	The percent change in an item value that will cause an exception report of that value to a client. This parameter only applies to items in the group that have dwEUType of Analog. [See discussion of Percent Deadband in General Properties Section]
pLCID	The current LCID for the group.
phClientGroup	The client supplied group handle
phServerGroup	The server generated group handle

HRESULT Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.

Comments

This function is typically called to obtain the current values of this information prior to calling SetState. This information was all supplied by or returned to the client when the group was created. This function is also useful for debugging.

All out arguments must be valid pointers. The marshaling mechanism requires valid pointers for proper behavior. NULL pointers will throw an RPC exception.

The client must free the returned ppName string.

4.4.3.2 IOPCGroupStateMgt::SetState

```
HRESULT SetState(
        [unique, in] DWORD * pRequestedUpdateRate,
        [out] DWORD * pRevisedUpdateRate,
        [unique, in] BOOL *pActive,
        [unique, in] LONG * pTimeBias,
        [unique, in] FLOAT * pPercentDeadband
        [unique, in] DWORD * pLCID,
        [unique, in] OPCHANDLE *phClientGroup
        );
```

Description

Client can set various properties of the group. Pointers to 'in' items are used so that the client can omit properties he does not want to change by passing a NULL pointer.

The pRevisedUpdateRate argument must contain a valid pointer.

Parameters	Description
pRequestedUpdateRate	New update rate requested for the group by the client (milliseconds)
pRevisedUpdateRate	Closest update rate the server is able to provide for this group.
pActive	TRUE (non-zero) to active the group. FALSE (0) to deactivate the group.
pTimeBias	TimeZone Bias of Group (in minutes).
pPercentDeadband	The percent change in an item value that will cause an exception report of that value to a client. This parameter only applies to items in the group that have dwEUType of Analog. See discussion of Percent Deadband in the General Information Section
pLCID	The Localization ID to be used by the group.
phClientGroup	New client supplied handle for the group. This handle is returned in the clients callback.

HRESULT Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.
OPC_S_UNSUPPORTEDRATE	The server does not support the requested data rate but will use the closest available rate.

Comments

Refer to Data Acquistion Section for details on the behavior of an OPC server with respect to the Synchronous and Asynchronous interfaces and Active state of groups.

As noted in AddGroup the level of localization supported (dwLCID) is entirely server specific. Servers that do not support dynamic localization, can ignore this parameter.

4.4.3.3 IOPCGroupStateMgt::SetName

```
HRESULT SetName(
        [in, string] LPCWSTR szName,
        );
```

Description

Change the name of a group. The name must be unique.

Parameters	Description
szName	New name for group.

HRESULT Return Codes

Return Code	Description
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
S_OK	The operation succeeded.
OPC_E_DUPLICATENAME	Duplicate name not allowed.

Comments

Group names are required to be unique with respect to an individual client to server connection.

4.4.3.4 IOPCGroupStateMgt::CloneGroup

```
HRESULT CloneGroup(
        [in, string] LPCWSTR szName,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN * ppUnk
        );
```

Description

Creates a second copy of a group with a unique name. All of the group and item properties are duplicated (as if the same set of AddItems calls had been made for the new group). That is, the new group contains the same update rate, items, group and item clienthandles, requested data types, etc as the original group. Once the new group is created it is entirely independent of the old group. You can add and delete items from it without affecting the old group.

Properties NOT copied to the new group are

- Active Status of the new group is initially set to FALSE
- A new ServerHandle for the group is produced.
- New Item SeverHandles may also be assigned by the server. The client should query for these if it needs them.
- The new group will NOT be connected to any Advise or Connection point sinks. The client would need to establish new connections for the new group.

Parameters	Description
szName	Name of the group. The name must be unique among the other groups created by this client. If no name is provided (szName is a pointer to a NUL string) the server will generate a unique name.
riid	Requested interface type
ppUnk	Place to return interface pointer. NULL is returned for any HRESULT other than S_OK

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_DUPLICATENAME	Duplicate name not allowed.
E_NOINTERFACE	The interface(riid) asked for is not supported by the server.

Comments

This represents a new group which is independent of the original group. See AddGroup for a discussion of Group object lifetime issues. As with AddGroup the group must be deleted with RemoveGroup when the client is done with it.

The client must also release the returned interface when it is no longer needed.

4.4.4 IOPCGroupStateMgt2

This interface was added to enhance the existing IOPCGroupStateMgt interface. IOPCGroupStateMgt2 inherits from IOPCGroupStateMgt and therefore all IOPCGroupStateMgt methods defined in IOPCGroupStateMgt are also part of this interface and will not be documented here. Please refer to the IOPCGroupStateMgt interface methods for further details. It is expected that Data Access 3.0 only servers, will implement this interface as opposed to IOPCGroupStateMgt. The purpose of this interface is to set/get the keep-alive time for a subscription. When a subscription has a non-zero keep-alive time, the server will insure that the client receives a callback on the subscription minimally at the rate indicated by the keep-alive time, even when there are no new events to report. By providing callbacks at a minimum known frequency, the client can be assured of the health of the server and subscription without resorting to pinging the server with calls to GetStatus().

4.4.4.1 IOPCGroupStateMgt2::SetKeepAlive

```
HRESULT SetKeepAlive(
        [in] DWORD dwKeepAliveTime,
        [out] DWORD *pdwRevisedKeepAliveTime
                );
```

Description

Clients can set the keep-alive time for a subscription to cause the server to provide client callbacks on the subscription when there are no new events to report. Clients can then be assured of the health of the server and subscription without resorting to pinging the server with calls to GetStatus().

Using this facility, a client can expect a callback (data or keep-alive) within the specified keep-alive time.

Servers shall reset their keep-alive timers when real data is sent (i.e. it is not acceptable to constantly send the keep-alive callback at a fixed period equal to the keep-alive time irrespective of data callbacks).

Parameters	Description
dwKeepAliveTime	The maximum amount of time (in milliseconds) between subscription callbacks. A value of zero indicates the client does not wish to receive any empty keep-alive callbacks.
pdwRevisedKeepAliveTime	The KeepAliveTime the server is actually providing, which may differ from .dwKeepAliveTime.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
E_INVALIDARG	A bad parameter was passed.
E_FAIL	The function was unsuccessful.
OPC_S_UNSUPPORTEDRATE	The function succeeded but *pdwRevisedKeepAlive does not equal dwKeepAlive.

Comments

The keep-alive callback consists of a call to IOPCDataCallback:: OnDataChange() with dwCount set to zero.

Keep-alive callbacks will not occur when the subscription is inactive.

Keep-alive callbacks do not affect the value of **OPCSERVERSTATUS::ftLastUpdateTime** returned by **IOPCServer::GetStatus()**.

4.4.4.2 IOPCGroupStateMgt2::GetKeepAlive

```
HRESULT GetKeepAlive(
        [out] DWORD *pdwKeepAliveTime
                );
```

Description

Returns the currently active keep-alive time for the subscription.

Parameters	Description
pdwKeepAliveTime	The maximum amount of time (in milliseconds) between subscription callbacks. A value of zero indicates the server will not send any empty keep-alive callbacks.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
E_FAIL	The function was unsuccessful.

Comments

If **SetKeepAlive()** has never been called, the returned value will be zero.

4.4.5 IOPCSyncIO

IOPCSyncIO allows a client to perform synchronous read and write operations to a server. The operations will run to completion.

Refer to the Data Acquisition and Active State Behavior table for an overview of the server data acquisition behavior and it's affect on functionality within this interface.

Also refer to the Serialization and Syncronization issues section earlier in this document.

4.4.5.1 IOPCSyncIO::Read

```
HRESULT Read(
        [in] OPCDATASOURCE dwSource,
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [out, size_is(,dwCount)] OPCITEMSTATE ** ppItemValues,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

This function reads the value, quality and timestamp information for one or more items in a group. The function runs to completion before returning. The data can be read from CACHE in which case it should be accurate to within the 'UpdateRate' and percent deadband of the group or item. The data can be read from the DEVICE in which case an actual read of the physical device is to be performed. The exact implementation of CACHE and DEVICE reads is not defined by this specification.

When reading from CACHE, the data is only valid if both the group and the item are active. If either the group or the item is inactive, then the Quality will indicate out of service (OPC_QUALITY_OUT_OF_SERVICE). Refer to the discussion of the quality bits later in this document for further information.

DEVICE reads are not affected by the ACTIVE state of the group or item.

Refer to the Data Acquisition and Active State Behavior table earlier in this document for an overview of the server data acquisition behavior and it's affect on functionality within this interface.

Parameters	Description
dwSource	The 'data source'; OPC_DS_CACHE or OPC_DS_DEVICE
dwCount	The number of items to be read.
phServer	The list of server item handles for the items to be read
ppItemValues	Array of structures in which the item values are returned.
ppErrors	Array of HRESULTs indicating the success of the individual item reads. The errors correspond to the handles passed in phServer. This indicates whether the read succeeded in obtaining a defined value, quality and timestamp. NOTE any FAILED error code indicates that the corresponding Value, Quality and Time stamp are UNDEFINED.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
S_FALSE	The operation succeeded but there are one or more errors in ppErrors. Refer to individual error returns for more infomation.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (e.g dwCount=0)

ppError Codes

Return Code	Description
S_OK	Successful Read.
E_FAIL	The Read failed for this item
OPC_E_BADRIGHTS	The item is not readable
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
S_xxx E_xxx	S_xxx - Vendor specific information can be provided if this item quality is other than GOOD. E_xxx - Vendor specific error if this item cannot be accessed. These vendor specific codes can be passed to GetErrorString().

Comments

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is S_FALSE, then ppError will indicate which the status of each individual Item Read.

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters including ppErrors.

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

For any FAILED ppError code the client should assume the corresponding ITEMSTATE is undefined. In fact the Server must set the corresponding ITEMSTATE VARIANT to VT_EMPTY so that it can be marshalled properly and so that the client can execute VariantClear on it.

Note that here (as in the OPCItemMgt methods) OPC_E_INVALIDHANDLE on one item will not affect the processing of other items and will cause the main HRESULT to return as S_FALSE

Expected behavior is that a CACHE read should be completed very quickly (within milliseconds). A DEVICE read may take a very long time (many seconds or more). Depending on the details of the

implementation (e.g. which threading model is used) the DEVICE read may also prevent any other operations from being performed on the server by any other clients.

For this reason Clients are expected to use CACHE reads in most cases. DEVICE reads are intended for ‘special’ circumstances such as diagnostics.

The ppItemValues and ppErrors arrays are allocated by the server and must be freed by the client. Be sure to call **VariantClear()** on the variant in the ITEMRESULT.

4.4.5.2 IOPCSyncIO::Write

```
HRESULT Write(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in, size_is(dwCount)] VARIANT * pItemValues,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Writes values to one or more items in a group. The function runs to completion. The values are written to the DEVICE. That is, the function should not return until it verifies that the device has actually accepted (or rejected) the data.

Writes are not affected by the ACTIVE state of the group or item.

Parameters	Description
dwCount	Number of items to be written
phServer	The list of server item handles for the items to be read
pItemValues	List of values to be written to the items. The datatypes of the values do not need to match the datatypes of the target items. However an error will be returned if a conversion cannot be done.
ppErrors	Array of HRESULTs indicating the success of the individual item Writes. The errors correspond to the handles passed in phServer. This indicates whether the target device or system accepted the value. NOTE any FAILED error code indicates that the value was rejected by the device.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
S_FALSE	The operation succeeded but there are one or more errors in ppErrors. Refer to individual error returns for more information.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (e.g dwCount=0)

ppError Codes

Return Code	Description
S_OK	The function was successful.
E_FAIL	The function was unsuccessful.
OPC_S_CLAMP	The value was accepted but was clamped.
OPC_E_RANGE	The value was out of range.
OPC_E_BADTYPE	The passed data type cannot be accepted for this item (See comment)
OPC_E_BADRIGHTS	The item is not writeable
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space
E_xxx S_xxx	Vendor specific errors may also be returned. Descriptive information for such errors can be obtained from GetErrorString.

Comments

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

Note that here (as in the OPCItemMgt methods) OPC_E_INVALIDHANDLE on one item will not affect the processing of other items and will cause the main HRESULT to return as S_FALSE

As an alternative to OPC_E_BADTPYE it is acceptable for the server to return any FAILED error returned by VariantChangeType or VariantChangeTypeEx.

A DEVICE write may take a very long time (many seconds or more). Depending on the details of the implementation (e.g. which threading model is used) the DEVICE write may also prevent any other operations from being performed on the server by any other clients.

For this reason Clients are expected to use ASYNC write rather than SYNC write in most cases.

The ppErrors array is allocated by the server and must be freed by the client.

4.4.6 IOPCSyncIO2

This interface was added to enhance the existing IOPCSyncIO interface. IOPCSyncIO2 inherits from IOPCSyncIO and therefore all IOPCSyncIO methods defined in IOPCSyncIO are also part of this interface and will not be documented here. Please refer to the IOPCSyncIO interface methods for further details. It is expected that Data Access 3.0 only servers, will implement this interface as opposed to IOPCSyncIO. The purpose of this interface is to provide a group level method for writing timestamp and quality information into servers that support this functionality. In addition, the ability to Read from a group based on a "MaxAge" is provided. This interface differs from the IOPCItemIO interface in that it is group based as opposed to server based.

4.4.6.1 IOPCSyncIO2::ReadMaxAge

```
HRESULT ReadMaxAge (
   [in]  DWORD dwCount,
   [in, sizeis(dwCount)] OPCHANDLE *phServer
   [in, sizeis(dwCount)] DWORD *pdwMaxAge,
   [out, sizeis(,dwCount)] VARIANT  **ppvValues,
   [out, sizeis(,dwCount)] WORD  **ppwQualities,
   [out, sizeis(,dwCount)] FILETIME  **ppftTimeStamps,
   [out, sizeis(,dwCount)] HRESULT  **ppErrors
   );
```

Description

Reads one or more values, qualities and timestamps for the items specified. This is functionally similar to the OPCSyncIO::Read method except no source is specified (DEVICE or CACHE). The server will make the determination as whether the information will be obtained from the device or cache. This decision will be based upon the MaxAge parameter. If the information in the cache is within the MaxAge, then the data will be obtained from the cache, otherwise the server must access the device for the requested information.

Parameters	Description
dwCount	The number of items to be read.
phServer	The list of server item handles for the items to be read.
dwMaxAge	An array of "staleness" for each item, requested in milliseconds. The server will calculate, for each requested item, the number of milliseconds between "now" and the timestamp on each item. For each item that 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 0xFFFFFFFF 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 a DWORD of milliseconds, the largest MaxAge value would be approximately is 49.7 days).
ppvValues	A pointer to an array of VARIANTs in which the results are returned. Note that the array and its contained variants must be freed by the client after receipt.
ppwQualities	An array of Words in which to store the Quality of each result. Note that these must be freed by the client.
ppftTimeStamps	An array of FILETIMEs in which to store the timestamps of each result. Note that these must be freed by the client.
ppErrors	Array of HRESULTs indicating the success of the individual item reads. The errors correspond to the items or server item handles passed in phServer. This indicates whether the read succeeded in obtaining a defined value, quality and timestamp. NOTE any FAILED error code indicates that the corresponding Value, Quality and Time stamp are UNDEFINED. Note that these must be freed by the client.

Return Codes

Return Code	Description
S_OK	The operation succeeded.
S_FALSE	The operation succeeded but there are one or more errors in ppErrors. Refer to individual error returns for more information.
E_INVALIDARG	An invalid argument was passed. (e.g. dwCount=0)
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory

ppErrorCodes

Return Code	Description
S_OK	Successful Read.
E_FAIL	The Read failed for this item
OPC_E_BADRIGHTS	The item is not readable
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
S_xxx E_xxx	S_xxx - Vendor specific information can be provided if this item quality is other than

	GOOD. E_xxx - Vendor specific error if this item cannot be accessed. These vendor specific codes can be passed to GetErrorString().

Comments

The MaxAge will only be tested upon receipt of this call. For example, if 3 items are requested with a MaxAge of 1000 milliseconds and two of the three items are within the appropriate range, then the third item must be obtained from the underlying device. Once this item is obtained the three items will be packaged and returned to the client even if the MaxAge of the other two expired while obtaining the third value. The test for MaxAge will not be re-evaluated and therefore the two "stale" items will be returned with the items obtained directly from the device. This functionality is in place to prevent the server from recursively attempting to obtain the values.

Some servers may return always the actual value, if DEVICE = CACHE.

If the HRESULT is S_OK, then ppErrors can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is S_FALSE, then ppErrors will indicate which the status of each individual Item Read.

If the HRESULT is any FAILED code then the contents of all OUT parameters including ppErrors is undefined.

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

For any FAILED (E_) result in ppError the client may not make use of the corresponding Value, Quality and Timestamp except to call VariantClear before destroying the VARIANT. When returning an E_ result in ppError, the Server must set the corresponding Value's VARIANT to VT_EMPTY so that it can be marshaled properly and so that the client can execute VariantClear on it.

Be sure to call VariantClear() on the variants in the Values before freeing the memory containing them.

The client is responsible for freeing all memory associated with the out parameters.

4.4.6.2 IOPCSyncIO2::WriteVQT

```
HRESULT WriteVQT (
  [in]  DWORD dwCount,
  [in, sizeis(dwCount)] OPCHANDLE  *phServer,
  [in, sizeis(dwCount)] OPCITEMVQT *pItemVQT,
  [out, sizeis(,dwCount)] HRESULT  **ppErrors
  );
```

Description

Writes one or more values, qualities and timestamps for the items specified. This is functionally similar to the IOPCSyncIO::Write except that Quality and Timestamp may be written. If a client attempts to write VQ, VT, or VQT it should expect that the server will write them all or none at all.

Parameters	Description
dwCount	The Number of Items to be written.
phServer	The list of server item handles for the items to be read
pItemVQT	The list of OPCITEMVQT structures. Each structure will potentially contain a value, quality and timestamp to be written to the corresponding ItemID. It the value is equal to VT_EMPTY, then no value should be written. There is a Boolean value associated with each Quality and Timestamp. The Boolean is an indicator as to whether the Quality or Timestamp should be written for the corresponding item or server item handle. True indicates write, while false indicates do not write.
ppErrors	The list of errors resulting from the write (1 per item). Note that these must be freed by the client.

Return Codes

Return Code	Description
S_OK	The operation succeeded.
S_FALSE	The operation succeeded but there are one or more errors in ppErrors. Refer to individual error returns for more information. It is also necessary to return S_FALSE, if there are one ore more OPC_S_XXX results in ppErrors.
OPC_E_NOTSUPPORTED	If a client attempts to write any value, quality, timestamp combination and the server does not support the requested combination (which could be a single quantity such as just timestamp), then the server will not perform any write and will return this error code.
E_INVALIDARG	An invalid argument was passed. (e.g. dwCount=0)
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory

ppErrorCodes

Return Code	Description
S_OK	The function was successful.
E_FAIL	The function was unsuccessful.
OPC_S_CLAMP	The value was accepted but was clamped.
OPC_E_RANGE	The value was out of range.
OPC_E_BADTYPE	The passed data type cannot be accepted for this item
OPC_E_BADRIGHTS	The item is not writeable
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
E_xxx S_xxx	Vendor specific errors may also be returned. Descriptive information for such errors can be obtained from GetErrorString.

Comments

If the HRESULT is S_OK, then ppErrors can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then the contents of ppErrors is undefined.

As an alternative to OPC_E_BADTPYE it is acceptable for the server to return any FAILED error returned by VariantChangeType or VariantChangeTypeEx.

Note: There is no way to validate the writing of the timestamp. If writing the timestamp is supported by the server, then the timestamp will be updated on the device as opposed to the cache. Writing timestamps is generally expected to be used for values which are in some sort of manual override mode or for values which are in some form of holding register. In general it is not useful to write timestamps for values which are actually being generated or scanned by the device since the device will generally re-stamp the data each time it is generated or scanned.

4.4.7 IOPCAsyncIO2

IOPCAsyncIO2 allows a client to perform asynchronous read and write operations to a server. The operations will be ‘queued’ and the function will return immediately so that the client can continue to run. Each operation is treated as a ‘transaction’ and is associated with a transaction ID. As the operations are completed, a callback will be made to the IOPCDataCallback in the client. The information in the callback will indicate the transaction ID and the results of the operation.

Also the expected behavior is that for any one transaction to Async Read, Write and Refresh, ALL of the results of that transaction will be returned in a single call to the appropriate function in IOPCDataCallback.

A server must be able to ‘queue’ at least one transaction of each type (read, write, refresh) for each group. It is acceptable for a server to return an error (CONNECT_E_ADVISELIMIT) if more than one transaction of the same type is performed on the same group by the same client. Server vendors may of course support queuing of additional transactions if they wish.

All operations that are successfully started are expected to complete even if they complete with an error. The concept of ‘time-out’ is not explicitly addressed in this specification however it is expected that where appropriate the server will internally implement any needed time-out logic and return a server specific error to the caller if this occurs.

Client Implementation Note:

The Unique Transaction ID passed to Read, Write and Refresh is generated by the Client and is returned to the client in the callback as a way to identify the returned data. To insure proper client operation, this ID should generally be non-zero and should be unique to this particular client/server conversation. It does not need to be unique relative to other conversations by this or other clients. In any case however the transactionID is completely client specific and must not be checked by the server.

Note that the Group's ClientHandle is also returned in the callback and is generally sufficient to identify the returned data.

IMPORTANT NOTE: depending on the mix of client and server threading models used, it has been found in practice that the IOPCDataCallback can occur within the same thread as the Refresh, Read or Write and in fact can occur before the Read, Write or Refresh method returns to the caller.

Thus, if the client wants to save a record of the transaction in some list of ‘outstanding transactions’ in order to verify completion of a transaction it will need to generate the Transaction ID and save it BEFORE making the method call.

In practice most clients will probably not need to maintain such a list and so do not actually need to record the transaction ID.

4.4.7.1 IOPCAsyncIO2::Read

```
HRESULT Read(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in] DWORD dwTransactionID,
        [out] DWORD *pdwCancelID,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Read one or more items in a group. The results are returned via the client's IOPCDataCallback connection established through the server's IConnectionPointContainer.

Reads are from 'DEVICE' and are not affected by the ACTIVE state of the group or item.

Parameters	Description
dwCount	Number of items to be read.
phServer	Array of server item handles of the items to be read
dwTransactionID	The Client generated transaction ID. This is included in the 'completion' information provided to the OnReadComplete.
pdwCancelID	Place to return a Server generated ID to be used in case the operation needs to be canceled.
ppErrors	Array of errors for each item - returned by the server. See below.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded. The read was successfully initiated
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (e.g dwCount=0)
S_FALSE	One or more of the passed items could not be read. The ppError array indicates which items in phServer could not be read. Any items which do not return errors (E) here will be read and results **will** be returned to OnReadComplete. Items which do return errors here **will not** be returned in the callback.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.

ppError Codes

Return Code	Description
S_OK	The corresponding Item handle was valid and the item information will be returned on OnReadComplete.
E_FAIL	The Read failed for this item
OPC_E_BADRIGHTS	The item is not readable
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
E_xxx S_xxx	Vendor specific errors may also be returned. Descriptive information for such errors can be obtained from GetErrorString.

Comments

Some servers will be ‘smarter’ at read time and return ‘early’ errors, others may simply queue the request with minimal checking and return ‘late’ errors in the callback. The client should be prepared to deal with this.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters. Note that in this case no Callback will occur.

If ALL errors in ppError are Failure codes then No callback will take place.

Items for which ppError returns any success code (including S_xxx) will be returned in the OnReadComplete callback. Note that the error result for an item returned in the callback may differ from that returned from Read.

NOTE: the server must return all of the results in a single callback. Thus, if the items in the group require multiple physical transactions to one or more physical devices then the server must wait until all of them are complete before invoking OnReadComplete.

The Client must free the returned ppError array.

4.4.7.2 IOPCAsyncIO2::Write

```
HRESULT Write(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in, size_is(dwCount)] VARIANT * pItemValues,
        [in] DWORD dwTransactionID,
        [out] DWORD *pdwCancelID,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Write one or more items in a group. The results are returned via the client's IOPCDataCallback connection established through the server's IConnectionPointContainer.

Parameters	Description
dwCount	Number of items to be written
phServer	List of server items handles for the items to be written
pItemValues	List of values to be written. The value data types are not required to match the requested or canonical item datatype but must be 'convertible' to the canonical type.
dwTransactionID	The Client generated transaction ID. This is included in the 'completion' information provided to the OnWriteComplete.
pdwCancelID	Place to return a Server generated ID to be used in case the operation needs to be canceled.
ppErrors	Array of errors for each item - returned by the server. See below.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid. (e.g dwCount=0)
S_FALSE	One or more of the passed items could not be written. The ppError array indicates which items in phServer could not be written. Any items which do not return errors (E) here will be written and results **will** be returned to OnWriteComplete. Items which do return errors here **will not** be returned in the callback.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.

ppError Codes

Return Code	Description
S_OK	The corresponding Item handle was valid. The write will be attempted and the results will be returned on OnWriteComplete
E_FAIL	The function was unsuccessful.
OPC_E_BADRIGHTS	The item is not writeable
OPC_E_BADTYPE	The passed data type cannot be accepted for this item.
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space
E_xxx S_xxx	Vendor specific errors may also be returned. Descriptive information for such errors can be obtained from GetErrorString.

Comments

Some servers will be ‘smarter’ at write time and return ‘early’ errors, others may simply queue the request with minimal checking and return ‘late’ errors in the callback. The client should be prepared to deal with this.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters. Note that in this case no Callback will occur.

If ALL errors in ppError are Failure codes then No callback will take place.

Items for which ppError returns any success code (including S_xxx) will also have a result returned in the OnWriteComplete callback. Note that the error result for an item returned in the callback may differ from that returned from Write.

NOTE: all of the results must be returned by the server in a single callback. Thus if the items in the group require multiple physical transactions to one or more physical devices then the server must wait until all of them are complete before invoking the callback.

Client must free the returned ppError array.

4.4.7.3 IOPCAsyncIO2::Refresh2

HRESULT Refresh2(
[in] OPCDATASOURCE dwSource,
[in] DWORD dwTransactionID,
[out] DWORD *pdwCancelID
);

Description

Force a callback to IOPCDataCallback::OnDataChange for all active items in the group (whether they have changed or not). Inactive items are not included in the callback.

Parameters	Description
dwSource	Data source CACHE or DEVICE. If the DEVICE, then all active items in the CACHE are refreshed from the device BEFORE the callback.
dwTransactionID	The Client generated transaction ID. This is included in the 'completion' information provided to the OnDataChange.
pdwCancelID	Place to return a Server generated ID to be used in case the operation needs to be canceled.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed. (See notes below)
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.

Comments

If the HRESULT is any FAILED code then no Callback will occur.

Calling Refresh for an InActive Group will return E_FAIL. Calling refresh for an Active Group, where all the items in the group are InActive also returns E_FAIL.

The behavior of this function is identical to what happens when Advise is called initially except that the OnDataChange Callback will include the transaction ID specified here. (The initial OnDataChange callback will contain a Transaction ID of 0). Thus if it is important to the client to distinguish between OnDataChange callbacks resulting from changes to values and OnDataChange callbacks resulting from a Refresh2 request then a non-zero ID should be passed to Refresh2().

Functionally it is also similar to what could be achieved by doing a READ of all of the active items in a group.

NOTE: all of the results must be returned in a single callback. Thus if the items in the group require multiple physical transactions to one or more physical devices then the server must wait until all of them are complete before invoking OnDataChange.

The expected behavior is that this Refresh will not affect the timing of normal OnDataChange callbacks which are based on the UpdateRate. For example, if the update rate is 1 hour and this method is called after 45 minutes then the server should still do its internal 'checking' at the end of the hour (15 minutes after the Refresh call). Calling this method may affect the contents of that next callback (15 minutes later) since only items where the value or status changed during that 15 minutes would be included. Items which had changed during the 45 minutes preceding the Refresh will be sent (along with all other values) as part of the Refresh Transaction. They would not be sent a second time at the end of the hour. The value sent in response to the Refresh becomes the 'last value sent' to the client when performing the normal subscription logic.

4.4.7.4 IOPCAsyncIO2::Cancel2

```
HRESULT Cancel2(
        [in] DWORD dwCancelID
        );
```

Description

Request that the server cancel an outstanding transaction.

Parameters	Description
dwCancelID	The Server generated Cancel ID which was associated with the operation when it was initiated.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed. Either the Cancel ID was invalid or it was 'too late' to cancel the transaction.

Comments

The exact behavior (for example whether an operation that has actually started will be aborted) will be server specific and will also depend on the timing of the cancel request. Also, depending on the timing, a Callback for the transaction may or may not occur. This method is intended for use during shutdown of a task.

In general, if this operation succeeds then a OnCancelComplete callback will occur. If this operation fails then a read, write or datachange callback may occur (or may already have occurred).

4.4.7.5 IOPCAsyncIO2::SetEnable

```
HRESULT SetEnable(
        [in] BOOL bEnable
        );
```

Description

Controls the operation of OnDataChange. Basically setting Enable to FALSE will disable any OnDataChange callbacks with a transaction ID of 0 (those which are not the result of a Refresh).

Parameters	Description
bEnable	TRUE enables OnDataChange callbacks, FALSE disables OnDataChange callbacks.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.
E_FAIL	The operation failed.

Comments

The initial value of this variable when the group is created is TRUE and thus OnDataChange callbacks are enabled by default.

The purpose of this function is to allow a Connection to be established to an active group without necessarily enabling the OnDataChange notifications. An example might be a client doing an occasional Refresh from cache.

Even if a client does not intend to use the OnDataChange, it should still be prepared to deal with one or more OnDataChange callbacks which might occur before the client has time to disable them (i.e. at least free the memory associated with the 'out' parameters).

If the client really needs to prevent these initial unwanted callbacks then the following procedure can be used. Client creates and populates the group. Client sets the group Active state to FALSE. Client creates connection to group. Client uses this function to disable OnDataChange. Client sets the group Active state back to TRUE.

This does NOT affect operation of Refresh2(). I.e. calling Refresh2 will still result in an OnDataChange callback (with a non-zero transaction ID). Note that this allows Refresh to be used as essentially an Async read from Cache.

4.4.7.6 IOPCAsyncIO2::GetEnable

```
HRESULT GetEnable(
        [out] BOOL *pbEnable
        );
```

Description

Retrieves the last Callback Enable value set with SetEnable.

Parameters	Description
pbEnable	Where to save the returned result.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.
E_FAIL	The operation failed.

Comments

See IOPCAsyncIO2::SetEnable() for additional information.

4.4.8 IOPCAsyncIO3

This interface was added to enhance the existing IOPCAsyncIO2 interface. IOPCAsyncIO3 inherits from IOPCAsyncIO2 and therefore all IOPCAsyncIO2 methods defined in IOPCAsyncIO2 are also part of this interface and will not be documented here. Please refer to the IOPCAsyncIO2 interface methods for further details. It is expected that Data Access 3.0 only servers, will implement this interface as opposed to IOPCAsyncIO2. The purpose of this interface is to provide a group level method for asynchronously writing timestamp and quality information into servers that support this functionality. In addition, the ability to asynchronously Read from a group based on a "MaxAge" is provided. This interface differs from the IOPCItemIO interface in that it is asynchronous and group based as opposed to server based.

4.4.8.1 IOPCAsyncIO3::ReadMaxAge

```
HRESULT ReadMaxAge (
   [in]  DWORD dwCount,
   [in, sizeis(dwCount)] OPCHANDLE *phServer
   [in, sizeis(dwCount)] DWORD *pdwMaxAge,
    [in] DWORD dwTransactionID,
   [out] DWORD *pdwCancelID,
   [out, sizeis(,dwCount)] HRESULT  **ppErrors
   );
```

Description

Reads one or more values, qualities and timestamps for the items specified. This is functionally similar to the OPCSyncIO::Read method except it is asynchronous and no source is specified (DEVICE or CACHE). The server will make the determination as whether the information will be obtained from the device or cache. This decision will be based upon the MaxAge parameter. If the information in the cache is within the MaxAge, then the data will be obtained from the cache, otherwise the server must access the device for the requested information.

Parameters	Description
dwCount	The number of items to be read.
phServer	The list of server item handles for the items to be read.
dwMaxAge	An array of "staleness" for each item, requested in milliseconds. The server will calculate, for each requested item, the number of milliseconds between "now" and the timestamp on each item. For each item that 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 0xFFFFFFFF 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 a DWORD of milliseconds, the largest MaxAge value would be approximately is 49.7 days).
dwTransactionID	The Client generated transaction ID. This is included in the 'completion' information provided to the OnReadComplete.
pdwCancelID	Place to return a Server generated ID to be used in case the operation needs to be canceled.
ppErrors	Array of errors for each item - returned by the server. See below. The errors correspond to the items or server item handles passed in phServer. Note that these must be freed by the client.

Return Codes

Return Code	**Description**
S_OK	The operation succeeded.
E_INVALIDARG	An invalid argument was passed. (e.g. dwCount=0)
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
S_FALSE	One or more of the passed items could not be read. The ppErrors array indicates which items in phServer could not be read. Any items which do not return errors (E) here will be read and results **will** be returned to OnReadComplete. Items which do return errors here **will not** be returned in the callback.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.

ppErrorCodes

Return Code	**Description**
S_OK	Successful Read.
E_FAIL	The Read failed for this item
OPC_E_BADRIGHTS	The item is not readable
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
S_xxx E_xxx	S_xxx - Vendor specific information can be provided if this item quality is other than GOOD. E_xxx - Vendor specific error if this item

	cannot be accessed. These vendor specific codes can be passed to GetErrorString().

Comments

The MaxAge will only be tested upon receipt of this call. For example, if 3 items are requested with a MaxAge of 1000 milliseconds and two of the three items are within the appropriate range, then the third item must be obtained from the underlying device. Once this item is obtained the three items will be packaged and returned to the client even if the MaxAge of the other two expired while obtaining the third value. The test for MaxAge will not be re-evaluated and therefore the two "stale" items will be returned with the items obtained directly from the device. This functionality is in place to prevent the server from recursively attempting to obtain the values.

Some servers will be 'smarter' at read time and return 'early' errors, others may simply queue the request with minimal checking and return 'late' errors in the callback. The client should be prepared to deal with this.

Some servers may return always the actual value, if DEVICE = CACHE.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters. Note that in this case no Callback will occur.

If ALL errors in ppError are Failure codes then No callback will take place.

Items for which ppError returns any success code (including S_xxx) will be returned in the OnReadComplete callback. Note that the error result for an item returned in the callback may differ from that returned from Read.

NOTE: the server must return all of the results in a single callback. Thus, if the items in the group require multiple physical transactions to one or more physical devices then the server must wait until all of them are complete before invoking OnReadComplete.

The Client must free the returned ppError array.

4.4.8.2 IOPCAsyncIO3::WriteVQT

```
HRESULT WriteVQT (
   [in]  DWORD dwCount,
   [in, sizeis(dwCount)] OPCHANDLE  *phServer,
   [in, sizeis(dwCount)] OPCITEMVQT *pItemVQT,
  [in] DWORD dwTransactionID,
  [out] DWORD *pdwCancelID,
   [out, sizeis(,dwCount)] HRESULT  **ppErrors
   );
```

Description

Writes one or more values, qualities and timestamps for the items specified. The results are returned via the client's IOPCDataCallback connection established through the server's IConnectionPointContainer. This is functionally similar to the IOPCAsyncIO2::Write except that Quality and Timestamp may be written. If a client attempts to write VQ, VT, or VQT it should expect that the server will write them all or none at all.

Parameters	Description
dwCount	The Number of Items to be written.
phServer	The list of server item handles for the items to be read
pItemVQT	The list of OPCITEMVQT structures. Each structure will potentially contain a value, quality and timestamp to be written to the corresponding ItemID. It the value is equal to VT_EMPTY, then no value should be written. There is a Boolean value associated with each Quality and Timestamp. The Boolean is an indicator as to whether the Quality or Timestamp should be written for the corresponding ItemID. True indicates write, while false indicates do not write.
dwTransactionID	The Client generated transaction ID. This is included in the 'completion' information provided to the OnWriteComplete.
pdwCancelID	Place to return a Server generated ID to be used in case the operation needs to be canceled.
ppErrors	Array of errors for each item - returned by the server. See below: Note that these must be freed by the client.

Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_INVALIDARG	An invalid argument was passed. (e.g. dwCount=0)
OPC_E_NOTSUPPORTED	If a client attempts to write any value, quality, timestamp combination and the server does not support the requested combination (which could be a single quantity such as just timestamp), then the server will not perform any write and will return this error code.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
S_FALSE	One or more of the passed items could not be written. The ppError array indicates which items in phServer could not be written. Any items which do not return errors (E) here will be written and results **will** be returned to OnWriteComplete. Items which do return errors here **will not** be returned in the callback. It is also necessary to return S_FALSE, if there are one ore more OPC_S_XXX results in ppErrors.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.

ppErrorCodes

Return Code	Description
S_OK	The function was successful.
E_FAIL	The function was unsuccessful.
OPC_E_BADTYPE	The passed data type cannot be accepted for this item
OPC_E_BADRIGHTS	The item is not writeable
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space
OPC_E_INVALIDHANDLE	The passed item handle was invalid.
E_xxx S_xxx	Vendor specific errors may also be returned. Descriptive information for such errors can be obtained from GetErrorString.

Comments

Some servers will be ‘smarter’ at write time and return ‘early’ errors, others may simply queue the request with minimal checking and return ‘late’ errors in the callback. The client should be prepared to deal with this.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters. Note that in this case no Callback will occur.

If ALL errors in ppError are Failure codes then No callback will take place.

Items for which ppError returns any success code (including S_xxx) will also have a result returned in the OnWriteComplete callback. Note that the error result for an item returned in the callback may differ from that returned from Write.

NOTE: all of the results must be returned by the server in a single callback. Thus if the items in the group require multiple physical transactions to one or more physical devices then the server must wait until all of them are complete before invoking the callback.

Client must free the returned ppError array.

4.4.8.3 IOPCAsyncIO3:: RefreshMaxAge

```
HRESULT RefreshMaxAge(
        [in]  DWORD dwMaxAge,
        [in]  DWORD dwTransactionID,
        [out] DWORD *pdwCancelID
        );
```

Description

Force a callback to IOPCDataCallback::OnDataChange for all active items in the group (whether they have changed or not). Inactive items are not included in the callback. The MaxAge value will determine where the data is obtained. There will be only one MaxAge value, which will determine the MaxAge for all active items in the group. This means some of the values may be obtained from cache while others could be obtained from the device depending on the "freshness" of the data in the cache.

Parameters	Description
dwMaxAge	An indicator of "staleness" for all items, requested in milliseconds. The server will calculate the number of milliseconds between "now" and the timestamp on each item. For each item that 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 0xFFFFFFFF 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 a DWORD of milliseconds, the largest MaxAge value would be approximately is 49.7 days).
dwTransactionID	The Client generated transaction ID. This is included in the 'completion' information provided to the

	OnDataChange.
pdwCancelID	Place to return a Server generated ID to be used in case the operation needs to be canceled.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed. (See notes below)
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.

Comments

If the HRESULT is any FAILED code then no Callback will occur.

Calling RefreshMaxAge for an inactive Group will return E_FAIL. Calling RefreshMaxAge for an Active Group, where all the items in the group are inactive also returns E_FAIL.

The behavior of this function is identical to what happens when Advise is called initially except that the OnDataChange Callback will include the transaction ID specified here. (The initial OnDataChange callback will contain a Transaction ID of 0). Thus if it is important to the client to distinguish between OnDataChange callbacks resulting from changes to values and OnDataChange callbacks resulting from a RefreshMaxAge request then a non-zero ID should be passed to RefreshMaxAge().

Functionally it is also similar to what could be achieved by doing a ReadMaxAge of all of the active items in a group using a common MaxAge value.

NOTE: all of the results must be returned in a single callback. Thus if the items in the group require multiple physical transactions to one or more physical devices then the server must wait until all of them are complete before invoking OnDataChange.

The expected behavior is that this RefreshMaxAge will not affect the timing of normal OnDataChange callbacks which are based on the UpdateRate. For example, if the update rate is 1 hour and this method is called after 45 minutes then the server should still do its internal ‘checking’ at the end of the hour (15 minutes after the Refresh call). Calling this method may affect the contents of that next callback (15 minutes later) since only items where the value or status changed during that 15 minutes would be included. Items which had changed during the 45 minutes preceding the Refresh will be sent (along with all other values) as part of the Refresh Transaction. They would not be sent a second time at the end of the hour. The value sent in response to the Refresh becomes the ‘last value sent’ to the client when performing the normal subscription logic.

4.4.9 IOPCItemDeadbandMgt

This interface allows the PercentDeadband to be set for individual items within a group. Once the item PercentDeadband is set, it overrides the PercentDeadband set for the entire group. This provides a mechanism to set the PercentDeadband on a "noisy" item, which may reside in a group that doesn't have the group PercentDeadband set. It also allows individual items to be fine tuned with respect to notifications based on an expected range of change.

4.4.9.1 IOPCItemDeadbandMgt::SetItemDeadband

```
HRESULT SetItemDeadband(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in, size_is(dwCount)] FLOAT * pPercentDeadband,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Overrides the deadband specified for the group for each item.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
pPercentDeadband	Array of deadband values. Each value must be from 0.0 to 100.0, which is the percentage of the change allowed per update period.
ppErrors	Array of HRESULT's. Indicates the results of setting the deadband for each item.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
E_INVALIDARG	An argument to the function was invalid.
OPC_E_DEADBAND NOTSUPPORTED	The server does not support deadband.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.
OPC_E_DEADBANDNOTS UPPORTED	The item handle does not support deadband.
E_INVALIDARG	Requested deadband was not in the range of 0.0 to 100.0.

Comments

The percent change of an item value will cause a subscription callback for that value. Only items in the group that have dwEUType of Analog can have a PercentDeadband set. If the item does not have support deadband, then the ppError for that particular item will be set to OPC_E_DEADBANDNOTSUPPORTED and the setting will have no effect on that particular item.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

If the HRESULT is S_FALSE, then the ppErrors array must be examined, since at least one item has failed and at least one item was successful. The client must examine the ppErrors array to make this determination.

The ppErrors array is allocated by the server and must be freed by the client.

The default deadband for the item is the group deadband. See the Percent Deadband section for additional information.

4.4.9.2 IOPCItemDeadbandMgt:: GetItemDeadband

```
HRESULT GetItemDeadband(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [out, size_is(,dwCount)] FLOAT **ppPercentDeadband,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Gets the deadband values for each of the requested items.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
ppPercentDeadband	Array of deadband values. Each successful value will be from 0.0 to 100.0, which describes the percent deadband allowed for that particular item.
ppErrors	Array of HRESULT's. Indicates the results of getting the deadband for each item.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
E_INVALIDARG	An argument to the function was invalid.
OPC_E_DEADBAND NOTSUPPORTED	The server does not support deadband.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.
OPC_E_DEADBANDNOTSUPP ORTED	The item does not support deadband.
OPC_E_DEADBANDNOTSET	The item deadband has not been set for this item.

Comments

Only items in the group that have dwEUType of Analog can have a PercentDeadband. If the item does not support deadband, then the ppError for that particular item will be set to OPC_E_DEADBANDNOTSUPPORTED.

If SetItemDeadband did not previously set the PercentDeadband, then no PercentDeadband for that particular item will be obtained and the ppError for that item will be set to OPC_E_DEADBANDNOTSET.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

If the HRESULT is S_FALSE, then the ppErrors array must be examined, since at least one item has failed and at least one item was successful. The client must examine the ppErrors array to make this determination.

The ppPercentDeadband and ppErrors arrays are allocated by the server and must be freed by the client.

4.4.9.3 IOPCItemDeadbandMgt:: ClearItemDeadband

```
HRESULT ClearItemDeadband(
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Clears the individual item PercentDeadband, effectively reverting them back to the deadband value set in the group.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
ppErrors	Array of HRESULT's. Indicates the results of clearing the deadband for each item.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
E_INVALIDARG	An argument to the function was invalid.
OPC_E_DEADBAND NOTSUPPORTED	The server does not support deadband.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.
OPC_E_DEADBANDNOTSUPPORTED	The item does not support deadband.
OPC_E_DEADBANDNOTSET	The item deadband has not been set for this item.

Comments

Only items in the group that have dwEUType of Analog can have a PercentDeadband. If the item does not support deadband, then the ppError for that particular item will be set to OPC_E_DEADBANDNOTSUPPORTED.

If SetItemDeadband did not previously set the PercentDeadband, then no PercentDeadband for that particular item will be obtained and the ppError for that item will be set to OPC_E_DEADBANDNOTSET.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

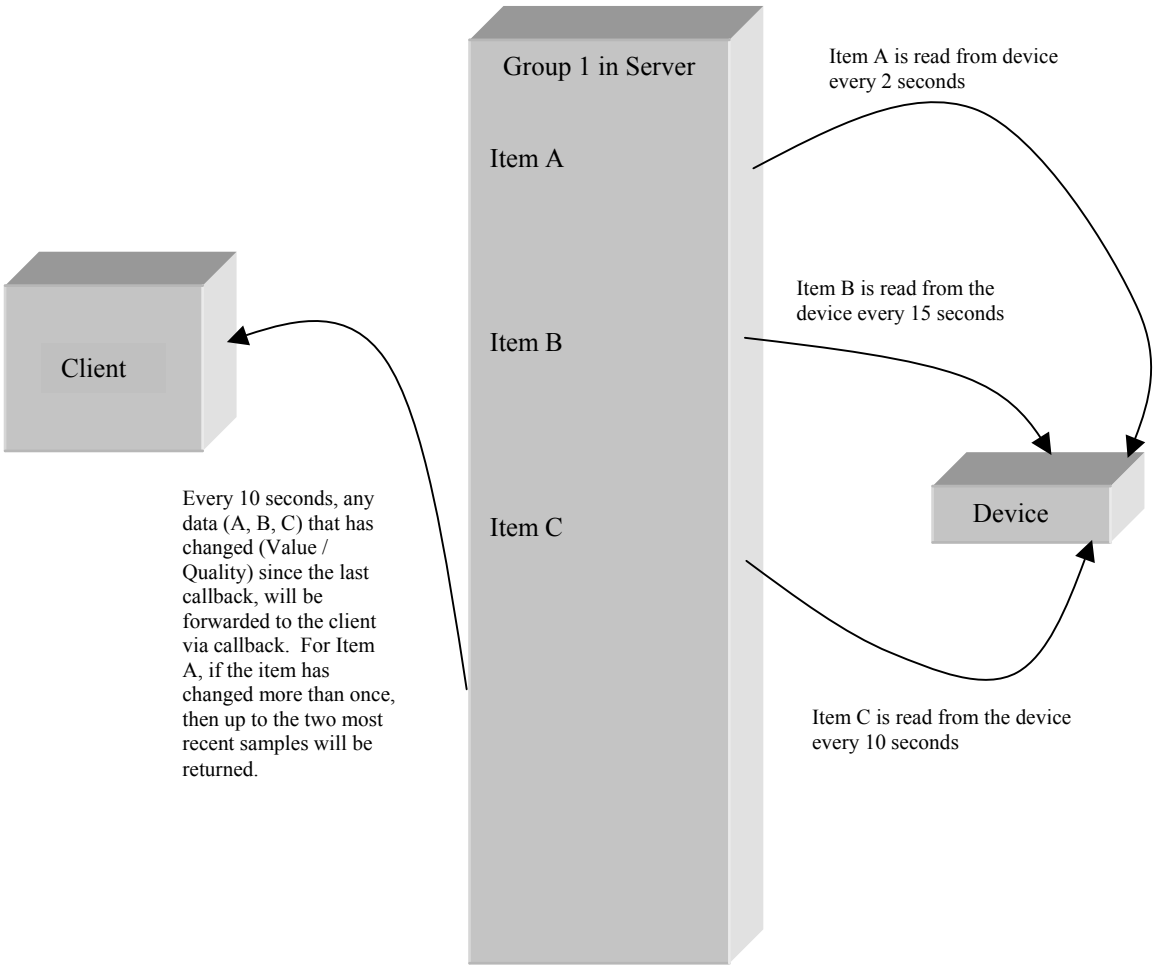
If the HRESULT is S_FALSE, then the ppErrors array must be examined, since at least one item has failed and at least one item was successful. The client must examine the ppErrors array to make this determination.

The ppErrors array is allocated by the server and must be freed by the client.

4.4.10 IOPCItemSamplingMgt (optional)

This optional interface allows the client to manipulate the rate at which individual items within a group are obtained from the underlying device. It does not affect the group update rate of the callbacks for OnDataChange.

Example: Group 1 has 3 items (A, B, C) with an update rate of 10 seconds. The group and items are all active. Item A has a sampling rate of 2 seconds and item B has a sampling rate of 15 seconds. Item C does not have it's sampling rate set. Buffering is enable on item A, and the buffer is of size 2.



The current rules that apply for returning items with a group update are:

1) If the quality has changed since the last update then the item is returned to the client.

2) If the value has changed since the last update and the total change exceeds the deadband (if applicable) then the item is returned to the client.

If the item sampling rate is faster than the group update rate then the server must apply additional logic to determine what is returned to the client at the next update.

If buffering is not enabled and at least one sample meets criteria 1 & 2 above then the latest value is returned to client. The latest sample is returned even if the latest value does not meet criteria 1 & 2.

If buffering is enabled then the server does not start buffing samples until reading a sample that meets criteria 1 & 2. Once a server starts buffering samples for an item it will add a new sample to the buffer if the new sample has a different quality or value when compared to the previous sample. If new sample is the same as the last sample in the buffer then server only updates the timestamp of last sample in the buffer.

In short, the set of samples returned to the client will be a sequence of values which all differ from the previous sample and have a timestamp that reflects the last time the item was known to have that value.

If an item has more than one value/quality/timestamp to be returned with a particular OnDataChange callback, then there will be multiple duplicate ClientHandles, depending on the size of the collection, returned with their corresponding value/quality/timestamp trio.

4.4.10.1 IOPCItemSamplingMgt::SetItemSamplingRate

```
HRESULT SetItemSamplingRate (
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [in, size_is(dwCount)] DWORD * pdwRequestedSamplingRate,
        [out, size_is(,dwCount)] DWORD **ppdwRevisedSamplingRate,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Sets the sampling rate on individual items. This overrides the update rate of the group as far as collection from the underlying device is concerned. The update rate associated with individual items does not effect the callback period.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
pdwRequestedSampling Rate	Requested Sampling period for the item in milliseconds. This also indicates the desired accuracy of the Cached Data.
ppdwRevisedSampling Rate	The server returns the value it will actually use for the SamplingRate in milliseconds, which may differ from

	the RequestedSamplingRate.
ppErrors	Array of HRESULT's. Indicates the results of setting the sampling rates.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened. It is also necessary to return S_FALSE, if there are one ore more OPC_S_UNSUPPORTEDRATE results in ppErrors.
E_INVALIDARG	An argument to the function was invalid.
E_OUTOFMEMORY	Not enough memory to complete the requested operation. This can happen any time the server needs to allocate memory to complete the requested operation.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.
OPC_S_UNSUPPORTEDRATE	Server does not support requested rate, server returns the rate that it can support in the revised sampling rate.

If the item sampling rate is less than the group update rate, the server must buffer multiple values (if supported; see Set/Get ItemBufferEnable) for the item to be included in a single callback performed at the group update rate. Multiple values for the same ItemID 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; see Set/Get ItemBufferEnable) 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.

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

If the HRESULT is S_FALSE, then the ppErrors array must be examined, since at least one item has failed and at least one item was successful. The client must examine the ppErrors array to make this determination.

Because the server may need to buffer an unknown amount of data, the server is allowed to determine the maximum amount of data buffered. The server should maintain the same size buffer for each item. If the server determines that its maximum buffer capacity has been reached, then it will begin to push out the older data, keeping the newest data in the buffer for each item. In the event of a buffer overflow for a particular item, the ppError for this item will be OPC_S_DATAQUEUEOVERFLOW and the HRESULT will be set to S_FALSE for the callback.

If an item has more than one value/quality/timestamp to be returned with a particular OnDataChange callback, then there will be multiple duplicate ClientHandles, depending on the size of the collection, returned with their corresponding value/quality/timestamp trio.

The ppErrors array is allocated by the server and must be freed by the client.

4.4.10.2 IOPCItemSamplingMgt::GetItemSamplingRate

```
HRESULT GetItemSamplingRate (
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [out, size_is(,dwCount)] DWORD * ppdwSamplingRate,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Gets the sampling rate on individual items, which was previously set with SetItemSamplingRate.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
ppdwSamplingRate	Sampling period for the item in milliseconds.
ppErrors	Array of HRESULT's. Indicates the results of setting the sampling rates.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
E_INVALIDARG	An argument to the function was invalid.
E_OUTOFMEMORY	Not enough memory to complete the requested operation. This can happen any time the server needs to allocate memory to complete the requested operation.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.
OPC_E_RATENOTSET	Indicates that there is no sampling rate set for a particular item. In this case, the item defaults to the update rate of the group.

Comments

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

If the HRESULT is S_FALSE, then the ppErrors array must be examined, since at least one item has failed and at least one item was successful. The client must examine the ppErrors array to make this determination.

If the sampling rate has not been set by the SetItemSamplingRate, or ClearSamplingRate has reset the items individual sampling rate back to the overall groups update rate, then a ppError of OPC_E_RATENOTSET will be returned and the HRESULT will be set to S_FALSE.

The ppdwSamplingRate and ppErrors arrays are allocated by the server and must be freed by the client.

4.4.10.3 IOPCItemSamplingMgt::ClearItemSamplingRate

```
HRESULT ClearItemSamplingRate (
        [in] DWORD dwCount,
        [in, size_is(dwCount)] OPCHANDLE * phServer,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Clears the sampling rate on individual items, which was previously set with SetItemSamplingRate. The item will revert back to the update rate of the group.

Parameters	Description
dwCount	The number of items to be affected
phServer	Array of Server items handles.
ppErrors	Array of HRESULT's. Indicates the results of setting the sampling rates.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
E_INVALIDARG	An argument to the function was invalid.
E_OUTOFMEMORY	Not enough memory to complete the requested operation. This can happen any time the server needs to allocate memory to complete the requested operation.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.

Comments

If the HRESULT is S_OK, then ppError can be ignored (all results in it are guaranteed to be S_OK).

If the HRESULT is any FAILED code then (as noted earlier) the server should return NULL pointers for all OUT parameters.

If the HRESULT is S_FALSE, then the ppErrors array must be examined, since at least one item has failed and at least one item was successful. The client must examine the ppErrors array to make this determination.

If the update rate has not been set by the SetItemSamplingRate, or ClearSamplingRate has reset the items individual sampling rate back to the overall groups update rate, then a ppError of OPC_E_RATENOTSET will be returned and the HRESULT will be set to S_FALSE.

The ppErrors array is allocated by the server and must be freed by the client.

4.4.10.4 IOPCItemSamplingMgt::SetItemBufferEnable

```
HRESULT SetItemBufferEnable (
        [in] DWORD dwCount,
        [in, size_is(dwCount)]   OPCHANDLE  * phServer,
        [in, size_is(dwCount)]   BOOL *pbEnable,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Requests that the server turns on or off, depending on the value of the bEnable parameter, the buffering of data for the identified items, which are collected for items that have an update rate faster than the group update rate.

Parameters	Description
dwCount	The number of items to be affected.
phServer	Array of Server items handles.
pbEnable	An array of booleans to set the state of the buffering mechanism within the server. True is requesting that the server buffers data for the corresponding item, while false indicates the server should not buffer the data.
ppErrors	Array of HRESULT's. Indicates the results of setting the buffering.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
OPC_E_NOBUFFERING	The server does not support buffering of data items that are collected at a faster rate than the group update rate
E_INVALIDARG	An argument to the function was invalid.
E_OUTOFMEMORY	Not enough memory to complete the requested operation. This can happen any time the server needs to allocate memory to complete the requested operation.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.

Comments

The default for each server is to have buffering NOT enabled, even if buffering is supported. To enable buffering, this method must be called passing in a true value for those individual items that should be buffered. The amount of data buffered is server dependant. If the server does not support buffering, then an HRESULT of OPC_E_NOBUFFERING will be returned. If this interface is supported, then the server will support the collection rate regardless of whether buffering is supported or not. The ppErrors array is allocated by the server and must be freed by the client.

4.4.10.5 IOPCItemSamplingMgt::GetItemBufferEnable

```
HRESULT GetItemBufferEnable (
        [in] DWORD dwCount,
        [in, size_is(dwCount)]   OPCHANDLE  * phServer,
        [out, size_is(dwCount)]   BOOL ** ppbEnable,
        [out, size_is(,dwCount)] HRESULT ** ppErrors
        );
```

Description

Queries the current state of the servers buffering for requested items.

Parameters	Description
dwCount	The number of items to be affected.
phServer	Array of Server items handles.
ppbEnable	An array of returned booleans representing the state of the buffering mechanism within the server for each requested item. True is requesting that the server buffers data for the corresponding item, while false indicates the server should not buffer the data.
ppErrors	Array of HRESULT's. Indicates the results of setting the buffering.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
S_FALSE	The function was partially successful. See the ppErrors to determine what happened
OPC_E_NOBUFFERING	The server does not support buffering of data items that are collected at a faster rate than the group update rate
E_INVALIDARG	An argument to the function was invalid.
E_OUTOFMEMORY	Not enough memory to complete the requested operation. This can happen any time the server needs to allocate memory to complete the requested operation.
E_FAIL	The function was unsuccessful.

ppError Codes

Return Code	Description
S_OK	The function was successful.
OPC_E_INVALIDHANDLE	The corresponding Item handle was invalid.

Comments

The default for each server is to have buffering NOT enabled, even if buffering is supported. To query the current buffering state for a particular item with a group, this method is used. If buffering is turn on, then true is returned. If buffering is turned off, then false is returned. If the server does not support buffering, then an HRESULT of OPC_E_NOBUFFERING will be returned.

If the result of an method call is E_XXX, all OUT-Parameters are invalid.

The ppbEnable and ppErrors arrays are allocated by the server and must be freed by the client.

4.4.11 IConnectionPointContainer (on OPCGroup)

This interface provides functionality similar to the IDataObject but is easier to implement and to understand and also provides some functionality that was missing from the IDataObject Interface. The client must use the new IOPCAsyncIO2 interface to communicate via connections established with this interface. IOPCAsyncIO2 is described elsewhere. The 'legacy' IOPCAsnyc will continue to communicate via IDataObject connections as in the past.

The general principles of ConnectionPoints are not discussed here as they are covered very clearly in the Microsoft Documentation. The reader is assumed to be familiar with this technology. OPC Compliant Servers, versions 2.0 and higher, are REQUIRED to support this interface.

Likewise the details of the IEnumConnectionPoints, IConnectionPoint and IEnumConnections interfaces are well defined by Microsoft and are not discussed here.

Note that IConnectionPointContainer is implemented on the OPCGROUP rather than on the individual items. This is to allow the creation of a Callback connection between the client and the group using the IOPCDataCallback Interface for the most efficient possible transfer of data (many items per transaction).

One callback object implemented by the client application can be used to service multiple groups. Therefore, information about the group and the particular transaction must be provided to the client application for it to be able to successfully interpret the items that are contained in the callback. Each callback will contain only items defined within the specified group.

Note: OPC Compliant servers are not required to support more than one connection between each Group and the Client. Given that groups are client specific entities it is expected that a single connection (to each group) will be sufficient for virtually all applications. For this reason (as per the COM Specification) the EnumConnections method for IConnectionPoint interface for the IOPCDataCallback is allowed to return E_NOTIMPL.

4.4.11.1 IConnectionPointContainer::EnumConnectionPoints

```
HRESULT EnumConnectionPoints(
        IEnumConnectionPoints **ppEnum
        );
```

Description

Create an enumerator for the Connection Points supported between the OPC Group and the Client.

Parameters	Description
ppEnum	Where to save the pointer to the connection point enumerator. See the Microsoft documentation for a discussion of IEnumConnectionPoints.

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
For other codes see the OLE programmers reference	

Comments

OPCServers must return an enumerator that includes IOPCDataCallback. Additional vendor specific callbacks are also allowed.

4.4.11.2 IConnectionPointContainer:: FindConnectionPoint

```
HRESULT FindConnectionPoint(
        REFIID riid,
        IConnectionPoint **ppCP
        );
```

Description

Find a particular connection point between the OPC Group and the Client.

Parameters	Description
ppCP	Where to store the Connection Point. See the Microsoft documentation for a discussion of IConnectionPoint.
riid	The IID of the Connection Point. (e.g. IID_IOPCDataCallBack)

HRESULT Return Codes

Return Code	Description
S_OK	The function was successful.
For other codes see the OLE programmers reference	

Comments

OPCServers must support IID_IOPCDataCallback. Additional vendor specific callbacks are also allowed.

4.4.12 IEnumOPCItemAttributes

IEnumOPCItemAttributes allows a client to find out the contents (items) of a group and the attributes of those items.

NOTE: most of the returned information was either supplied by or returned to the client at the time it called AddItem.

The optional EU information (see the OPCITEMATTRIBUTES discussion) may be very useful to some clients. This interface is also useful for debugging .This interface is returned only by IOPCItemMgt::CreateEnumerator. It is not available through query interface.

Since enumeration is a standard interface this is described only briefly.

See the OLE Programmer's reference for Enumerators for a list and discussion of error codes.

4.4.12.1 IEnumOPCItemAttributes::Next

```
HRESULT Next(
        [in] ULONG celt,
        [out, size_is(,*pceltFetched)] OPCITEMATTRIBUTES ** ppItemArray,
        [out] ULONG * pceltFetched
        );
```

Description

Fetch the next 'celt' items from the group.

Parameters	Description
celt	Number of items to be fetched.
ppItemArray	Array of OPCITEMATTRIBUTES. Returned by the server.
pceltFetched	Number of items actually returned.

Comments

The client must free the returned OPCITEMATTRIBUTES structure including the contained items; szItemID, szAccessPath, pBlob, vEUInfo.

4.4.12.2 IEnumOPCItemAttributes::Skip

```
HRESULT Skip(
        [in] ULONG celt
        );
```

Description

Skip over the next ‘celt’ attributes.

Parameters	Description
celt	Number of items to skip

Comments

Skip is probably not useful in the context of OPC.

4.4.12.3 IEnumOPCItemAttributes::Reset

```
HRESULT Reset(
        void
        );
```

Description

Reset the enumerator back to the first item.

Parameters	Description
void	

Comments

4.4.12.4 IEnumOPCItemAttributes::Clone

```
HRESULT Clone(
        [out] IEnumOPCItemAttributes** ppEnumItemAttributes
        );
```

Description

Create a 2nd copy of the enumerator. The new enumerator will initially be in the same 'state' as the current enumerator.

Parameters	Description
ppEnumItemAttributes	Place to return the new interface

Comments

The client must release the returned interface pointer when it is done with it.

4.5 Client Side Interfaces

4.5.1 IOPCDataCallback

In order to use connection points, the client must create an object that supports both the IUnknown and IOPCDataCallback Interface. The client would pass a pointer to the IUnknown interface (NOT the IOPCDataCallback) to the Advise method of the proper IConnectionPoint in the server (as obtained from IConnectionPointContainer:: FindConnectionPoint or EnumConnectionPoints). The Server will call QueryInterface on the client object to obtain the IOPCDataCallback interface. Note that the transaction must be performed in this way in order for the interface marshalling to work properly for Local or Remote servers.

All of the methods below must be implemented by the client.

This Interface will be called as a result of changes in the data of the group (OnDataChange) and also as a result of calls to the IOPCAsyncIO2 interface.

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

Note: memory management follows the standard COM rules. That is, the server allocates 'in' parameters and frees them after the client returns. The client only frees 'out' parameters. In the case of these callbacks there are no 'out' parameters so all memory is owned by the server.

4.5.1.1 IOPCDataCallback::OnDataChange

```
HRESULT OnDataChange(
    [in] DWORD  dwTransid,
    [in] OPCHANDLE hGroup,
    [in] HRESULT  hrMasterquality,
    [in] HRESULT  hrMastererror,
    [in] DWORD  dwCount,
    [in, sizeis(dwCount)] OPCHANDLE * phClientItems,
    [in, sizeis(dwCount)] VARIANT * pvValues,
    [in, sizeis(dwCount)] WORD  * pwQualities,
    [in, sizeis(dwCount)] FILETIME  * pftTimeStamps,
    [in, sizeis(dwCount)] HRESULT *pErrors
    );
```

Description

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

Parameters	Description
dwTransid	0 if the call is the result of an ordinary subscription. If the call is the result of a call to Refresh2 then this is the value passed to Refresh2.
hGroup	The Client handle of the group
hrMasterquality	S_OK if OPC_QUALITY_MASK for all ‘qualities’ are OPC_QUALITY_GOOD, S_FALSE otherwise.
hrMastererror	S_OK if all ‘errors are S_OK, S_FALSE otherwise.
dwCount	The number of items in the client handle list A value of zero indicates this is a keep-alive notification (see IOPCGroupStateMgt2::SetKeepAlive())
phClientItems	The list of client handles for the items which have changed.
pvValues	A List of VARIANTS containing the values (in RequestedDataType) for the items which have changed.
pwQualities	A List of Quality values for the items
pftTimeStamps	A list of TimeStamps for the items
pErrors	A list of HRESULTS for the items. If the quality of a data item has changed to UNCERTAIN or BAD, this field allows the server to return additional server specific errors which provide more useful information to the user. See below.

HRESULT Return Codes

Return Code	Description
S_OK	The client must always return S_OK.

'pErrors' Return Codes

Return Code	Description
S_OK	The returned data for this item quality is GOOD.
E_FAIL	The Operation failed for this item.
OPC_E_BADRIGHTS	The item is or has become not readable.
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
OPC_S_DATAQUEUEOVERFLOW	Indicates that not every detected change has been returned. This is an indicator that servers buffer reached its limit and had to purge out the oldest data. Only the most recent data is provided. The server should only remove the oldest data for those items that have newer samples available in the buffer. This will allow single samplings of older items to be returned to the client.
S_xxx, E_xxx	S_xxx - Vendor specific information can be provided if this item quality is other than GOOD. E_xxx - Vendor specific error if this item cannot be accessed. These vendor specific codes can be passed to GetErrorString().

Comments

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

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

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

Callbacks can occur for the following reasons;

- One or more ‘data change’ events. These will happen for active items within an active group where the value or quality of the item has changed. They will happen no faster than the ‘updaterate’ of the group. Deadband is used to determine what items have changed. The TransactionID will be 0 in this case. In general, additional updates are not sent unless there is a change in value or quality.
- Refresh Request made through the AsyncIO2 interface. These will happen for all active items in an active group. They will happen as soon as possible after the refresh request is made. The handle list will contain the handles for all of the active items in the group. The transaction ID will be non-0 in this case.

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

See IOPCItemSamplingMgt::SetItemSamplingRate for more details on server behavior for buffered values.

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

During cleanup after the callback the Server must be sure to do a VariantClear() on each of the value Variants.

4.5.1.2 IOPCDataCallback::OnReadComplete

```
HRESULT OnReadComplete(
  [in] DWORD  dwTransid,
  [in] OPCHANDLE hGroup,
  [in] HRESULT  hrMasterquality,
  [in] HRESULT  hrMastererror,
  [in] DWORD  dwCount,
  [in, sizeis(dwCount)] OPCHANDLE * phClientItems,
  [in, sizeis(dwCount)] VARIANT * pvValues,
  [in, sizeis(dwCount)] WORD  * pwQualities,
  [in, sizeis(dwCount)] FILETIME  * pftTimeStamps,
  [in, sizeis(dwCount)] HRESULT *pErrors
  );
```

Description

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

Parameters	Description
dwTransid	The TransactionID returned to the client when the Read was initiated.
hGroup	The Client handle of the group
hrMasterquality	S_OK if OPC_QUALITY_MASK for all ‘qualities’ are OPC_QUALITY_GOOD, S_FALSE otherwise.
hrMastererror	S_OK if all ‘errors are S_OK, S_FALSE otherwise.
dwCount	The number of items in the client handle, values, qualities, times and errors lists. This may be less than the number of items passed to Read. Items for which errors were detected and returned from Read are not included in the callback.
phClientItems	The list of client handles for the items which were read. This is NOT guaranteed to be in any particular order although it will match the values, qualities, times and errors array.
pvValues	A List of VARIANTS containing the values (in RequestedDataType) for the items.
pwQualities	A List of Quality values for the items
pftTimeStamps	A list of TimeStamps for the items
pErrors	A list of HRESULTS for the items. If the system is unable to return data for an item, this field allows the server to return additional server specific errors which provide more useful information to the user.

HRESULT Return Codes

Return Code	Description
S_OK	The client must always return S_OK

'pErrors' Return Codes

Return Code	Description
S_OK	The returned data for this item quality is GOOD.
E_FAIL	The Read failed for this item
OPC_E_BADRIGHTS	The item is not readable
OPC_E_INVALIDHANDLE	The passed item handle was invalid. (Generally this should already have been tested by AsyncIO2::Read).
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
S_xxx, E_xxx	S_xxx - Vendor specific information can be provided if this item quality is other than GOOD. E_xxx - Vendor specific error if this item cannot be accessed. These vendor specific codes can be passed to GetErrorString().

Comments

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

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

Items for which an error (E_xxx) was returned in the initial AsyncIO2 Read request will NOT be returned here. I.e. the returned list may be 'sparse'. Also the order of the returned list is not specified (it may not match the order of the list passed to read).

This Callback occurs only after an AsyncIO2 Read.

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

4.5.1.3 IOPCDataCallback::OnWriteComplete

```
HRESULT OnWriteComplete(
  [in] DWORD  dwTransid,
  [in] OPCHANDLE   hGroup,
  [in] HRESULT  hrMasterError,
  [in] DWORD  dwCount,
  [in, sizeis(dwCount)] OPCHANDLE * phClientItems,
  [in, sizeis(dwCount)] HRESULT   * pError
);
```

Description

This method is provided by the client to handle notifications from the OPC Group on completion of AsyncIO2 Writes.

Parameters	Description
dwTransid	The TransactionID returned to the client when the Write was initiated.
hGroup	The Client handle of the group
hrMasterError	S_OK if all 'errors are S_OK, S_FALSE otherwise.
dwCount	The number of items in the client handle and errors list. This may be less than the number of items passed to Write. Items for which errors were detected and returned from Write are not included in the callback.
phClientItems	The list of client handles for the items which were written. This is NOT guaranteed to be in any particular order although it must match the 'errors' array.
pErrors	A List of HRESULTs for the items. Note that Servers are allowed to define vendor specific error codes here. These codes can be passed to GetErrorString().

HRESULT Return Codes

Return Code	Description
S_OK	The client must always return S_OK

'pErrors' Return Codes

Return Code	Description
S_OK	The data item was written.
OPC_E_BADRIGHTS	The item is not writable.
OPC_E_INVALIDHANDLE	The passed item handle was invalid. (Generally this should already have been tested by AsyncIO2::Write).
OPC_E_UNKNOWNITEMID	The item is no longer available in the server address space.
S_xxx, E_xxx	S_xxx - the data item was written but there is a vendor specific warning (for example the value

	was clamped). E_xxx - the data item was NOT written and there is a vendor specific error which provides more information (for example the device is offline). These codes can be passed to GetErrorString().

Comments

Items for which an error (E_xxx) was returned in the initial AsyncIO2 Write request will NOT be returned here. I.e. the returned list may be ‘sparse’. Also the order of the returned list is not specified (it may not match the order of the list passed to write).

This Callback occurs only after an AsyncIO2 Write.

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

4.5.1.4 IOPCDataCallback::OnCancelComplete

```
HRESULT OnCancelComplete(
  [in] DWORD  dwTransid,
  [in] OPCHANDLE   hGroup
);
```

Description

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

Parameters	Description
dwTransid	The TransactionID provided by the client when the Read, Write or Refresh was initiated.
hGroup	The Client handle of the group

HRESULT Return Codes

Return Code	Description
S_OK	The client must always return S_OK

Comments

This Callback occurs only after an AsyncIO2 Cancel. Note that if the Cancel Request returned S_OK then the client can expect to receive this callback. If the Cancel request Failed then the client should NOT receive this callback

4.5.2 IOPCShutdown

In order to use this connection point, the client must create an object that supports both the IUnknown and IOPCShutdown Interface. The client would pass a pointer to the IUnknown interface (NOT the IOPCShutdown) to the Advise method of the proper IConnectionPoint in the server (as obtained from IConnectionPointContainer:: FindConnectionPoint or EnumConnectionPoints). The Server will call QueryInterface on the client object to obtain the IOPCShutdown interface. Note that the transaction must be performed in this way in order for the interface marshalling to work properly for Local or Remote servers.

The ShutdownRequest method on this Interface will be called when the server needs to shutdown. The client should release all connections and interfaces for this server.

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

4.5.2.1 IOPCShutdown::ShutdownRequest

```
HRESULT ShutdownRequest (
  [in] LPWSTR  szReason
);
```

Description

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

Parameters	Description
szReason	An optional text string provided by the server indicating the reason for the shutdown. The server may pass a pointer to a NUL string if no reason is provided.

HRESULT Return Codes

Return Code	Description
S_OK	The client must always return S_OK.

Comments

The shutdown connection point is on a ‘per COM object’ basis. That is, it relates to the object created by CoCreate… If a client connects to multiple COM objects then it should monitor each one separately for shutdown requests.

5 Installation Issues

It is assumed that the server vendor will provide a SETUP.EXE to install the needed components for their server. This will not be discussed further. Other than the actual components, the main issue affecting OLE software is management of the Windows Registry and Component Categories. The issues here are (a) what entries need to be made and (b) how they can be made.

Again, certain common installation and registry topics including self-registration, automatic proxy/stub registration and registry reference counting are discussed in the OPC Overview Document.

5.1 Component Categories

The OPC Data Access Interface defines the following Component Categories. Listed below are the CATIDs, Descriptors and Symbolic Equates to be used for Data Access.

"OPC Data Access Servers Version 1.0"

CATID_OPCDAServer10 = {63D5F430-CFE4-11d1-B2C8-0060083BA1FB}

"OPC Data Access Servers Version 2.0"

CATID_OPCDAServer20 = {63D5F432-CFE4-11d1-B2C8-0060083BA1FB}

"OPC Data Access Servers Version 3.0"

CATID_OPCDAServer30 = {CC603642-66D7-48f1-B69A-B625E73652D7}

It is expected that a server will first create any category it uses and then will register for that category. Unregistering a server should cause it to be removed from that category. See the ICatRegister documentation for additional information.

These CATIDs are defined in their associated idl file. For Data Access 1.0, 2.0, 3.0 and XMLDA, the CATIDs are in the opcda.idl.

5.2 Registry Entries for Custom Interface

The following entries are the minimum required to support the Custom Interface for OPC Compliant Servers.

Required by all:

1. HKEY_CLASSES_ROOT*Vendor.Drivername.Version* = ***A Description of your server***

2. HKEY_CLASSES_ROOT*Vendor.Drivername.Version*\CLSID = {***Your Server's unique CLSID***}

3. HKEY_CLASSES_ROOT\CLSID\{***Your Server's unique CLSID***} = ***A Description of your server***

4. HKEY_CLASSES_ROOT\CLSID\{***Your Server's unique CLSID***}\ProgID = *Vendor.Drivername.Version*

One or more of the following lines (inproc and/or local/remote and/or handler)

5. HKEY_CLASSES_ROOT\CLSID\{***Your Server's unique CLSID***}\InprocServer32 = *Full Path to DLL*

6. HKEY_CLASSES_ROOT\CLSID\{***YourServer's unique CLSID***}\LocalServer32 = *Full Path to EXE*

7. HKEY_CLASSES_ROOT\CLSID\{***YourServer's unique CLSID***}\InprocHandler32 = *Full Path to DLL*

1. This entry simply establishes your ProgID as a subkey of the ROOT under which other subkeys can be entered. The description (the 'value' of this key) may be presented to the user as the name of an available OPC server (See example below). It should match the description in line 6.
2. The CLSID line enables the CLSIDFromProgID function to work. I.e. allows the system to open a key given the ProgID and obtain the CLSID as the value of that key. See the example below.
3. The OPC line creates a 'flag' subkey that has no value. This was used for Data Access 1.0 to allow the client to browse for the available OPC servers. As of version 2.0, the preferred approach is for clients and servers to use Component Categories. Version 3.0 approach requires clients and servers to use Component Categories.
4. The Vendor line is optional. It is simply a means of identifying the vendor who supplied the particular OPC server.
5. This line simply establishes your CLSID as a subkey off of ROOT\CLSID under which the other subkeys can be entered. The description (the 'value' of this key) should be a User Friendly description of the server. It should match Item 1 above.
6. The ProgID line enables the ProgIDFromCLSID function to work. I.e. allows the system to open a key given the CLSID and obtain the ProgID as the value of that key. (This function is not commonly used).
7. The InprocServer32 line or LocalServer32 line or InprocHandler32 line allows CoCreateInstance to locate the DLL or EXE given the CLSID. The vendor should define at least one of these.

In general, self registration as described in the Microsoft documentation is recommended for both DLL and EXE servers to simplify installation.

5.3 Registry Entries for the Proxy/Stub DLL

The proxy/stub DLL is used for marshalling interfaces to LOCAL or REMOTE servers. It is generated directly from the IDL code and should be the same for every OPC Server. In general the Proxy/Stub will use self-registration. (Define REGISTER_PROXY_DLL during the build). Since this is completely automatic and transparent it is not discussed further.

Also note that a prebuilt and tested proxy/stub DLL will be provided at the OPC Foundation Web site making it unnecessary for vendors to rebuild this DLL.

Although vendors are allowed to add their own interfaces to OPC objects (as with any COM object) they should NEVER modify the standard OPC IDL files or Proxy/Stub DLLs to include such interfaces. Such interfaces should ALWAYS be defined in a separate vendor specific IDL file and should be marshaled by a separate vendor specific Proxy/Stub DLL.

6 Description of Data Types, Parameters and Structures

Some structures contain ‘reserved’ words. These are generally inserted to pad structures to be 32 bit aligned.

6.1 Item Definition

The ItemID is the fully qualified definition of a data item in the server, commonly referred to as the WHAT. No other information is required to identify the data item for the client to be able to read/write values.

The Item definition (ItemID) used in the OPCITEMDEF and elsewhere is a null-terminated string that uniquely identifies an OPC data item. The syntax of the identifier is server dependent (although it should include only printable UNICODE characters) and it provides a reference or ‘key’ to an ‘item’ in the data source. The item is anything that can be represented by a VARIANT although it is typically a single value such as an analog, digital or string value.

For example, an item such as FIC101 might represent an entire record such as a Fieldbus, Hart Foundation or ProfiBus data structure. Such behavior is specifically allowed but not required by OPC - the return of such structures is considered to be vendor specific behavior. Alternately FIC101.PV might represent one attribute of a record such as the process value. This would probably take the form of a double which could be used by any client.

As an extreme example, since the syntax of the item ID is server specific, additional information such as Counts, Engineering Units Scaling and Signal conditioning information could be embedded in the definition string (although this is not recommended).

Examples:
A server which supports access to an existing DCS might support a simple syntax such as

“TIC101.PV”

A server that supports low level access to a PLC might support syntax such as

“COM1.STATION:42.REG:40001;0,4095,-100.0,+1234.0”

6.2 AccessPath

The AccessPath is intended as a way for the client to provide to the server a suggested data path (e.g. a particular modem or network interface). It indicates HOW to get the data.

The ITEM ID provides all of the information needed to locate and process a data item. The Access Path is an optional piece of information that can be provided by the client. Its use is highly server specific but it is intended to allow the client to provide a 'recommendation' to the server regarding how to get to the data. As an analogy, if the ItemID represented a phone number, the access path might represent a request to route the call via satellite (or transatlantic cable or microwave link). The call will go through regardless of whether you specify an access path and also whether or not the server is able to use that suggested path.

For example, suppose you wanted to access a value in an RTU and had a high speed modem on COM1 and a low speed modem on COM2. You might specify COM1 as the preferred access path. Either one will work, but you would prefer to use COM1 if it is available for better performance.

In any case, the use of access path by both the server and the client is optional. Servers need not provide the function and clients need not use it even if it is provided.

Servers which do not support access paths will completely ignore any passed access path (and will not treat this as an error by the client). Also, when queried, such servers will always return a null access path for all items (i.e. a NUL string).

6.3 Blob

We will discuss why the Blob exists and how it behaves.

The Blob is basically a scratch area for the server to associate with items in order to speed up access to or processing of those items. The exact way in which it is used is server specific.

The idea is that clients refer to items via ASCII strings while internally, to speed up access, the server will probably need to resolve this string into some internal server specific address; a network address, a pointer into a table, a set of indices or files or register numbers, etc. This address resolution could take considerable time and the resulting internal address could take an arbitrary amount of space. This Blob allows the server to return this internal address and allows the client to save it and to provide the Blob back to the server for future references to this item. The server could use the 'Blob' as a 'hint' to help find the item more quickly the next time; "The Blob says that last time I looked for this tag I found it 'here' - so lets see if its still in that location". However, in all cases, the ITEM ID is still the 'key' to the data. Regardless of the contents of the Blob, the server needs to insure that it is in fact referencing the item referred to by the ITEM ID.

The behavior of the Blob is as follows.

Its use by both client and server is optional. Servers which can perform 'AddItems' quickly based just on the item definition should generally not return a Blob. In cases where servers do return a Blob, clients are free to ignore these Blobs (although this will probably affect the performance of that server).

The Blob is passed to AddItems and ValidateItems and is also returned by the server any time an AddItems or ValidateItems or EnumItemAttributes is done. The returned Blob may differ in size and content from the one passed.

Note that the server can update the Blob for an item at any time entirely at the server's discretion (including, for example, whenever the client changes an attribute of an Item).

Proper behavior of a client that wishes to support the Blob is to Enumerate the item attributes to get a fresh copy of the Blobs for each item prior to deleting an item or group and to save that updated copy along with the other application data related to the items.

Comment:

The difference between the server handle and the Blob is that the server handle is fixed in size (DWORD), should not be stored between sessions by the client and that it's implementation is required since it is the only way to identify items after they have been added. The Blob is variable in length, is optional and may be stored by the client between sessions.

6.4 Time Stamps

Time stamps are in the form of a FILETIME as this is more compact than other available standard time structures. There are numerous WIN32 functions for converting between various time formats and time zones. Time stamps are always in UTC, this form is beneficial because it is always increasing and is unambiguous. As discussed earlier in this document, time stamps should reflect the best estimate of the most recent time at which the corresponding value was known to be accurate. If this is not provided by the device itself then it should be provided by the server.

6.5 Variant Data Types for OPC Data Items

Under NT 4.0 and Windows 95 with DCOM support, all VARIANT data types can be marshaled through standard marshalling. Under Automation, types will be coerced to known Automation data types.

NOTE

Real values in the variant (VT_R4, VT_R8) will contain IEEE floating point numbers. Note that the IEEE standard allows certain non-numeric values (called NANs) to be stored in this format. While use of such values is rare, they are specifically allowed. If such a value is returned (in the OPCITEMSTATE) it is required that the QUALITY flag be set to OPC_QUALITY_BAD.

Although the IEEE standard allows NANs to be stored in VT_R4 and VT_R8 format, such values can only be read and written using the exact native format of the target item. They will not be converted to/from other types. When such a value is read (as a native type) the QUALITY flag must be returned by the server as OPC_QUALITY_BAD. If such a value is written (as a native type) then the QUALITY flag provided by the client MUST be OPC_QUALITY_BAD. Whether or not a particular server ever returns or accepts such values is server specific.

6.6 Constants

6.6.1 OPCHANDLE

OPCHANDLEs are used in conjunction with both groups and items within groups. The purpose of handles in OPC is to allow faster access to various objects by both the client and the server.

The exact internal implementation of the server handles is entirely vendor specific. The client should never make any assumptions about the server handles and the server should never make any assumptions about the client handles.

6.6.1.1 Group Handles

OPC groups have both a client and a server handle associated with them.

The server group handle is unique across the server and must be returned when the group is created. The handle is then passed by the client to various methods. The server group handle can be assumed to remain valid until the client Removes the group and free's all of the interfaces.

It should not be persistently stored by the client as it may be different the next time the OPC group is created.

The client group handle is provided by the client to the server. It can be any value and does not need to be unique. It is included in order to help the client identify the source of the data for callbacks.

In practice it is expected that a client will assign a unique value to it's handle if it intends to use any of the asynchronous functions of the OPC interfaces (including IOPCAsyncIO2, and IConnectionPoint interfaces), since this is the only key to the information that the server gives back to the client via the IConnectionPoint interface.

6.6.1.2 Item Handles

OPC items have both a client and a server handle associated with them.

The server item handle is unique within the group and will be returned when the item is created. It is then passed by the client to various methods. The server item handle can be assumed to remain valid until the client Removes the items or Removes the Group containing the items.

It should not be persistently stored by the client as it may be different the next time the OPC Item is created.

The client item handle is provided by the client to the server. It can be any value and does not need to be unique. It is included in the IConnectionPoint callbacks in order to help the client quickly identify which object in the client application is affected by the changed data.

In practice however it is expected that a client will assign unique values to it's handles if it intends to use any of the asynchronous functions of the OPC interfaces (including IOPCAsyncIO2, and IConnectionPoint interfaces), since this is the only key to the information that the server gives back to the client via the IConnectionPoint interface.

6.7 Structures and Masks

6.7.1 OPCITEMSTATE

This structure is used by IOPCSyncIO::Read

```
typedef struct {
        OPCHANDLE    hClient;
        FILETIME     ftTimeStamp;
        WORD         wQuality;
        WORD         wReserved;
        VARIANT      vDataValue;
} OPCITEMSTATE;
```

Member	Description
hClient	The client provided handle for this item
ftTimeStamp	UTC TimeStamp for this item's value. If the device cannot provide a timestamp then the server should provide one.
wQuality	The quality of this item.
vDataValue	The value itself as a variant.

Comments

The Client should call VariantClear() to free any memory associated with the Variant.

Real values in the variant (VT_R4, VT_R8) will contain IEEE floating point numbers. Note that the IEEE standard allows certain non-numeric values (called NANs) to be stored in this format. While use of such values is rare, they are specifically allowed. If such a value is returned it is required that the QUALITY flag be set to OPC_QUALITY_BAD.

Although the IEEE standard allows NANs to be stored in VT_R4 and VT_R8 format, such values can only be read and written using the exact native format of the target item. They will not be converted to/from other types. When such a value is read (as a native type) the QUALITY flag must be returned by the server as OPC_QUALITY_BAD. If such a value is written (as a native type) then the QUALITY flag provided by the client MUST be OPC_QUALITY_BAD. Whether or not a particular server ever returns or accepts such values is server specific.

6.7.2 OPCITEMDEF

```
typedef struct {
        [string]  LPWSTR          szAccessPath;
        [string]  LPWSTR          szItemID;
        BOOL                      bActive ;
        OPCHANDLE                 hClient;
        DWORD                     dwBlobSize;
        [size_is(dwBlobSize)] BYTE * pBlob;
        VARTYPE                   vtRequestedDataType;
        WORD                      wReserved;
} OPCITEMDEF;
```

This structure is used by IOPCItemMgt::AddItems and ValidateItems. The 'used by' column below indicates which of these two functions use each member.

Member	Used by	Description
szAccessPath	both	The access path the server should associate with this item. By convention a pointer to a NUL string specifies that the server should select the access path. Support for access path is optional NOTE: version 1 indicated that a NULL pointer would allow the server to pick the path however passing a NULL pointer will cause a fault in the proxy/stub code and thus is not allowed.
szItemID	both	A null-terminated string that uniquely identifies the OPC data item. See the **Item ID** discussion and the AddItems function for specific information about the contents of this field.
bActive	add	This Boolean value affects the behavior various methods as described elsewhere in this specification.
hClient	add	The handle the client wishes to associate with the item. See the **OPCHANDLE** for more specific information about the contents of this field.
dwBlobSize	both	The size of the pBlob for this item.
pBlob	both	pBlob is a pointer to the Blob.
vtRequestedDataType	both	The data type requested by the client. An error is returned (See Additems or ValidateItems) if the server cannot provide the item in this format. Passing VT_EMPTY means the client will accept the server's canonical datatype.

Comments

Regarding the datatype; often the same value can be returned in more than one format. For example, a numeric value might be returned as text (VT_BSTR) or real (VT_R8). Such conversions are typically handled in the server by VariantChangeType(). Similarly a status (SCAN status, AUTO/MAN, Alarm, etc.) might be returned as an integer (VT_I4) to be used in animation or color selection or as a string (VT_BSTR) to be shown directly to the user. This second case is also known as an enumeration and would be vendor specific. Client vendors should note that this specification does not specify what enumerations exist or how a server maps the values into strings. Server vendors are strongly

encouraged to follow a standard such as FIELDBUS in this area. See IEnumOPCItemAttributes for more information on this topic.

6.7.3 OPCITEMRESULT

```
typedef struct {
        OPCHANDLE           hServer;
        VARTYPE             vtCanonicalDataType;
        WORD                wReserved;
        DWORD               dwAccessRights;
        DWORD               dwBlobSize;
        [size_is(dwBlobSize)] BYTE * pBlob;
} OPCITEMRESULT;
```

This structure is used by IOPCItemMgt::AddItems() and ValidateItems().

Member	Used by	Description
hServer	add	The server handle used to refer to this item.
vtCanonicalDataType	both	The native data type. The type of data maintained within the server for this item. A server that does not know the canonical type of an item (e.g. during startup) must return VT_EMPTY. This is an indication to the client that the datatype is not currently known but will be determined at some future time.
dwAccessRights	both	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. See the Access Rights section below.
dwBlobSize	both	The size of the Blob for this item. Note that this size may be 0 for servers that do not support or require this feature.
pBlob	both	Pointer to the Blob.

Comments

For AddItems pBlob will always be returned by servers which support this feature. For ValidateItems it will only be returned if the dwBlobUpdate parameter to ValidateItems is TRUE.

The client software must free the memory for the Blob before freeing the OPCITEMRESULT structure.

6.7.4 OPCITEMATTRIBUTES

```
typedef struct {
        [string]  LPWSTR          szAccessPath;
        [string]  LPWSTR          szItemID;
        BOOL                      bActive;
        OPCHANDLE                 hClient;
        OPCHANDLE                 hServer;
        DWORD                     dwAccessRights;
        DWORD                     dwBlobSize;
        [size_is(dwBlobSize)] BYTE * pBlob;
        VARTYPE                   vtRequestedDataType;
        VARTYPE                   vtCanonicalDataType;
        OPCEUTYPE                 dwEUType;
        VARIANT                   vEUInfo;
} OPCITEMATTRIBUTES;
```

Member	Description
szAccessPath	The access path specified by the client. A pointer to a NUL string is returned if the server does not support access paths.
szItemID	The unique identifier for this item.
bActive	FALSE if the item is not currently active, TRUE if the item is currently active.
hClient	The handle the client has associated with this item.
hServer	The handle the server uses to reference this item.
dwAccessRights	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. See the Access Rights section below.
dwBlobSize	The size of the pBlob for this item. Note that this size may be 0 for servers that do not support or require this feature.
pBlob	Pointer to the Blob.
vtRequestedDataType	The data type in which the item's value will be returned. Note that if the requested data type was rejected then this field will return the canonical data type.
vtCanonicalDataType	The data type in which the item's value is maintained within the server. A server that does not know the canonical type of an item (e.g. during startup) must return VT_EMPTY. This is an indication to the client that the datatype is not currently known but will be determined at some future time.
dwEUType	Indicate the type of Engineering Units (EU) information (if any) contained in vEUInfo. 0 - No EU information available (vEUInfo will be VT_EMPTY) 1 - Analog - vEUInfo will contain a SAFEARRAY of exactly two doubles (VT_ARRAY | VT_R8) corresponding to the LOW and HI EU range. 2 - Enumerated - vEUInfo 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.)

vEUInfo	The VARIANT containing the EU information. See Comments below.

Comment:

The EU support is optional. Servers which do not support this will always return EUType as 0 and EUInfo as VT_EMPTY. EU information (analog or enumerated) can be returned for any value where the canonical type is any of: VT_I2, I4, R4, R8, BOOL, UI1 although in practice some combinations are clearly more likely than others. Where the item contains an array of values (VT_ARRAY) the EU information will apply to all items in the array (just as the Requested and Canonical Data types apply to all items in the array).

EU information is provided by the server to the client and is essentially Read Only. OPC Does not provide the client with any control over the EU settings.

For analog EU the information returned represents the 'usual' range of the item value. Sensor or instrument failure or deactivation can result in a returned item value which is actually outside this range. Client software must be prepared to deal with this. Similarly a client may attempt to write a value which is outside this range back to the server. The exact behavior (accept, reject, clamp, etc.) in this case is server dependent however in general servers must be prepared to handle this.

For enumerated EU the information returned represents 'string lookup table' corresponding to sequential integer values starting with 0. The number of values represented is determined by the size of the SAFEARRAY. Again, robust clients should be prepared to handle item values outside the range of the list and robust servers should be prepared to handle writes of illegal values.

Servers may optionally support Localization of the enumeration. In this case the server should use the current locale ID of the group. See IOPCServer::AddGroup and IOPCGroupStateMgt::GetState and SetState.

The client is responsible for freeing the VARIANTs in the OPCITEMATTRIBUTES structure including all elements of any SAFEARRAYs.

Client writers may wish to create and use a common function such as FreeOPCITEMATTRIBUTES(ptr) in order to minimize the chance of memory leaks.

6.7.5 OPCSERVERSTATUS

```
typedef struct {
        FILETIME            ftStartTime;
        FILETIME            ftCurrentTime;
        FILETIME            ftLastUpdateTime;
        OPCSERVERSTATE      dwServerState;
        DWORD               dwGroupCount;
        DWORD               dwBandWidth;
        WORD                wMajorVersion;
        WORD                wMinorVersion;
        WORD                wBuildNumber;
        WORD                wReserved;
        [string] LPWSTR     szVendorInfo;
} OPCSERVERSTATUS;
```

This structure used to communicate the status of the server to the client. This information is provided by the server in the IOPCServer::GetStatus() call.

Member	Description
ftStartTime	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.
ftCurrentTime	The current time (UTC) as known by the server.
ftLastUpdateTime	The time (UTC) the server sent the last data value update to this client. This value is maintained on an instance basis.
dwServerState	The current status of the server. Refer to **OPC Server State values below.**
dwGroupCount	The total number of groups being managed by the server instance. This is mainly for diagnostic purposes.
dwBandWidth	The behavior of this field is server specific. A suggested use is that it return 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.
wMajorVersion	The major version of the server software
wMinorVersion	The minor version of the server software
wBuildNumber	The 'build number' of the server software
szVendorInfo	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.

OPCSERVERSTATE Values	Description
OPC_STATUS_RUNNING	The server is running normally. This is the usual state for a server
OPC_STATUS_FAILED	A vendor specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor specific. An error code of E_FAIL should generally be returned from any other server method.
OPC_STATUS_NOCONFIG	The server is running but has no configuration information loaded and thus cannot function normally. Note this state implies that the server needs configuration information in order to function. Servers which do not require configuration information should not return this state.
OPC_STATUS_SUSPENDED	The server has been temporarily suspended via some vendor specific method and is not getting or sending data. Note that Quality will be returned as OPC_QUALITY_OUT_OF_SERVICE.
OPC_STATUS_TEST	The server is in Test Mode. The outputs are disconnected from the real hardware but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.
OPC_STATUS_COMM_FAULT	The server is running properly but is having difficulty accessing data from its data sources. This may be due to communication problems, or some other problem preventing the underlying device, control system, etc. from returning valid data. It may be complete failure, meaning that no data is available, or a partial failure, meaning that some data is still available. It is expected that items affected by the fault will individually return with a BAD quality indication for the items.

6.7.6 Access Rights

This represents the server's ability to access a single OPC data item. Note the low 16 bits of the DWORD are reserved for OPC use and currently include the OPC Access Rights defined in the IDL and described below. The high 16 bits of the DWORD are available for vendor specific use.

The OPC_READABLE and OPC_WRITABLE bits are intended to indicate whether the Item is inherently readable or writable. For example a value representing a physical input would generally be readable but not writable. A value representing a physical output or an adjustable parameter such as a setpoint or alarm limit would generally be readable and writable. It is possible that a value representing a physical output with no readback capability might be marked writable but not readable. It is recommended that Client applications use this information only as something to be viewed by the user. Attempts by the user to read or write a value should always be passed by the client program to the server regardless of the access rights that were returned when the item was added. The Server can return E_BADRIGHTS if needed.

Also, the returned Access Rights value is not related to security issues. It is expected that a server implementing security would validate any reads or writes for the currently logged in user as they occurred and in case of a problem would return an appropriate vendor specific HRESULT in response to that read or write. The server should return both values (OPC_READABLE and OPC_WRITEABLE) to indicate that the access rights are currently “unknown”.

AccessRights Values	Description
OPC_READABLE	The client can read the data item's value.
OPC_WRITEABLE	The client can change the data item's value.

6.7.7 OPCITEMPROPERTY

A pointer to an array of this structure is an element of the structure **OPCITEMPROPERTIES.**.

```
typedef struct tagOPCITEMPROPERTY {
        VARTYPE         vtDataType;
        WORD            wReserved;
        DWORD           dwPropertyID;
        [string] LPWSTR szItemID;
        [string] LPWSTR szDescription;
        VARIANT         vValue;
        HRESULT         hrErrorID;
} OPCITEMPROPERTY;
```

Member	Description
vtDataType	The canonical data type of this property.
dwPropertyID	The Property ID for this property. For a list of defined properties, see the tables in **theItem Properties section.**
szItemID	A fully qualified ItemID that can be used to access this property. If a NULL string is returned, then the property can not be accessed via an ItemID.
szDescription	A non-localized text description of the property.
vValue	The current value of the property. If values were not requested, **vValue** is of type **VT_EMPTY**.
hrErrorID	if FAILED, only **dwPropertyID** will contain a known valid value. All other structure members may contain invalid data.

'Errors' Codes for hrErrorID

Return Code	Description
S_OK	The corresponding Property ID is valid and the structure contains accurate information.
OPC_E_INVALID_PID	The passed Property ID is not defined for this item.

Comment:

The caller must free the **szItemID** and **szDescription** strings and call VariantClear with **vValue**.

6.7.8 OPCITEMPROPERTIES

This structure is used in **IOPCBrowse::GetProperties()** and as an element of the structure **OPCBROWSEELEMENT.**

```
typedef struct tagOPCITEMPROPERTIES {
        HRESULT                                         hrErrorID;
        DWORD                                           dwNumProperties;
        [size_is(dwNumProperties)] OPCITEMPROPERTY *    pItemProperties;
} OPCITEMPROPERTIES;
```

Member	Description
hrErrorID	If FAILED, **dwNumProperties** must be 0 and **pItemProperties** must be NULL.
dwNumProperties	The number of elements in the **pItemProperties** array. 0 if the HRESULT is FAILED
pItemProperties	A pointer to an array of **OPCITEMPROPERTY** structures. NULL if the HRESULT is FAILED

'Errors' Codes for hrErrorID

Return Code	Description
S_OK	The requested properties were returned for the ItemID.
S_FALSE	One ore more requested properties were invalid for this ItemID.
OPC_ E_INVALIDITEMID	The item name does not conform to the server's syntax.
OPC_E_UNKNOWNITEMID	The item is not known in the server address space.

Comment:

The caller must free **pItemProperties**.

6.7.9 OPCBROWSEELEMENT

This structure is returned by **IOPCBrowse::Browse().**

```
typedef struct  tagOPCBROWSEELEMENT {
        [string] LPWSTR         szName;
        [string] LPWSTR         szItemID;
        DWORD                   dwFlagValue;

        OPCITEMPROPERTIES  ItemProperties;
} OPCBROWSEELEMENT;
```

Member	Description
szName	Short user friendly portion of the namespace pointing to the element. This is the string to be used for display purposes in a tree control..
szItemID	The unique identifier for this item.that can be used with AddItems, Browse or GetProperties.
dwFlagValue	The current bits available to be set are: OPC_BROWSE_HASCHILDREN = 0x01 OPC_BROWSE_ISITEM = 0x02 If the first bit is set (OPC_BROWSE_HASCHILDREN), then this indicates that the returned element has children and can be used for a subsequent browse. If it is too time consuming for a server to determine if an element has children, then this value should be set TRUE so that the the client is given the opportunity to attempt to browse for potential children. If the second bit is set (OPC_BROWSE_ISITEM) then the element is an item that can be used to Read, Write, and Subscribe. If the second bit is set and szItemID is a NULL string, then this element is a “hint” versus being a valid item.
ItemProperties	The Item Properties associated with this item.

Comment:

The szItemID used in a Browse() call must be the value of szItemID returned with a browse element from a previous call to browse or an empty string (used to indicate a top level browse).

The szItemID is a fully qualified descriptor for the element that can be used for further browsing and as an item id if the second bit (OPC_BROWSE_ISITEM) of **dwFlagValue** is set. Note, that it is possible for a single element to be both an item and have children (e.g. complex data items).

The caller must free the strings **szName** and **szItemID**.

6.7.10 OPCITEMVQT

This structure is used by IOPCItemIO::WriteVQT

```
typedef struct {
        VARIANT     vDataValue;
        BOOL        bQualitySpecified;
        WORD        wQuality;
        BOOL        bTimeStatmpSpecified;
        FILETIME    ftTimeStamp;
} OPCITEMVQT;
```

Member	Description
vDataValue	The value itself as a variant. If initialized to VT_EMPTY, then there is no value to write.
bQualitySpecified	A value of true indicates there is a valid Quality to write. A value of false indicates no Quality to write.
wQuality	The quality of this item.
bTimeStampSpecified	A value of true indicates there is a valid TimeStamp to write. A value of false indicates no TimeStamp to write.
ftTimeStamp	UTC TimeStamp for this item's value.

Comments

This structure is used to write various combinations of Value, Quality and TimeStamp to the device.

6.8 OPC Quality flags

These flags represent the quality state for a 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 low 8 bits of the Quality flags are currently defined in the form of three bit fields; Quality, Substatus and Limit status. The 8 Quality bits are arranged as follows:

QQSSSSLL

The high 8 bits of the Quality Word are available for vendor specific use. If these bits are used, the standard OPC Quality bits must still be set as accurately as possible to indicate what assumptions the client can make about the returned data. In addition it is the responsibility of any client interpreting vendor specific quality information to insure that the server providing it uses the same 'rules' as the client. The details of such a negotiation are not specified in this standard although a QueryInterface call to the server for a vendor specific interface such as IMyQualityDefinitions is a possible approach.

Details of the OPC standard quality bits follow:

The Quality BitField

QQ	BIT VALUE	DEFINE	DESCRIPTION
0	00SSSSLL	Bad	Value is not useful for reasons indicated by the Substatus.
1	01SSSSLL	Uncertain	The quality of the value is uncertain for reasons indicated by the Substatus.
2	10SSSSLL	N/A	Not used by OPC
3	11SSSSLL	Good	The Quality of the value is Good.

Comment:

A server which supports no quality information must return 3 (Good). It is also acceptable for a server to simply return Bad or Good (0x00 or 0xC0) and to always return 0 for Substatus and limit.

It is recommended that clients minimally check the Quality Bit field of all results (even if they do not check the substatus or limit fields).

Even when a 'BAD' value is indicated, the contents of the value field must still be a well-defined VARIANT even though it does not contain an accurate value. This is to simplify error handling in client applications. For example, clients are always expected to call VariantClear() on the results of a Synchronous Read.

If the server has no known value to return then some reasonable default should be returned such as a NUL string or a 0 numeric value.

The Substatus BitField

The layout of this field depends on the value of the Quality Field.

Substatus for BAD Quality:

SSSS	BIT VALUE	DEFINE	DESCRIPTION
0	000000LL	Non-specific	The value is bad but no specific reason is known.
1	000001LL	Configuration Error	There is some server specific problem with the configuration. For example the item in question has been deleted from the configuration.
2	000010LL	Not Connected	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.
3	000011LL	Device Failure	A device failure has been detected.
4	000100LL	Sensor Failure	A sensor failure had been detected (the 'Limits' field can provide additional diagnostic information in some situations).
5	000101LL	Last Known Value	Communications have failed. However, the last known value is available. Note that the 'age' of the value may be determined from the TIMESTAMP in the OPCITEMSTATE.
6	000110LL	Comm Failure	Communications have failed. There is no last known value is available.
7	000111LL	Out of Service	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.
8	001000LL	Waiting for Initial Data	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.
9-15		N/A	Reserved for future OPC use

Comment

Servers which do not support Substatus should return 0. Note that an 'old' value may be returned with the Quality set to BAD (0) and the Substatus set to 5. This is for consistency with the Fieldbus

Specification. This is the only case in which a client may assume that a 'BAD' value is still usable by the application.

Substatus for UNCERTAIN Quality:

SSSS	BIT VALUE	DEFINE	DESCRIPTION
0	010000LL	Non-specific	There is no specific reason why the value is uncertain.
1	010001LL	Last Usable Value	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 5 (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 TIMESTAMP in OPCITEMSTATE.
2-3		N/A	Not used by OPC
4	010100LL	Sensor Not Accurate	Either the value has 'pegged' at one of the sensor limits (in which case the limit field should be set to 1 or 2) 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 0).
5	010101LL	Engineering Units Exceeded	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.
6	010110LL	Sub-Normal	The value is derived from multiple sources and has less than the required number of Good sources.
7-15		N/A	reserved for future OPC use

Comment

Servers which do not support Substatus should return 0.

Substatus for GOOD Quality:

SSSS	BIT VALUE	DEFINE	DESCRIPTION
0	110000LL	Non-specific	The value is good. There are no special conditions
1-5		N/A	Not used by OPC
6	110110LL	Local Override	The value has been Overridden. Typically this is means the input has been disconnected and a manually entered value has been 'forced'.
7-15		N/A	reserved for future OPC use

Comment

Servers which do not support Substatus should return 0.

The Limit BitField

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

LL	BIT VALUE	DEFINE	DESCRIPTION
0	QQSSSS00	Not Limited	The value is free to move up or down
1	QQSSSS01	Low Limited	The value has 'pegged' at some lower limit
2	QQSSSS10	High Limited	The value has 'pegged' at some high limit.
3	QQSSSS11	Constant	The value is a constant and cannot move.

Comment

Servers which do not support Limit should return 0.

Symbolic Equates are defined for values and masks for these BitFields in the "QUALITY" section of the OPC header files.

7 Summary of OPC Error Codes

We have attempted to minimize the number of unique errors by identifying common generic problems and defining error codes that can be reused in many contexts. An OPC server should only return those OPC errors that are listed for the various methods in this specification or are standard Microsoft errors. Note that OLE itself will frequently return errors (such as RPC errors) in addition to those listed in this specification.

The most important thing for a client is to check FAILED for any error return. Other than that, (the statements above not withstanding) a robust, user-friendly client should assume that the server may return any error code and should call the GetErrorString function to provide user readable information about those errors.

Standard COM errors that are commonly used by OPC Servers	**Description**
E_FAIL	Unspecified error
E_INVALIDARG	The value of one or more parameters was not valid. This is generally used in place of a more specific error where it is expected that problems are unlikely or will be easy to identify (for example when there is only one parameter).
E_NOINTERFACE	No such interface supported
E_NOTIMPL	Not implemented
E_OUTOFMEMORY	Not enough memory to complete the requested operation. This can happen any time the server needs to allocate memory to complete the requested operation.
CONNECT_E_ADVISELIMIT	Advise limit exceeded for this object
OLE_E_NOCONNECTION	Cannot Unadvise - there is no existing connection
DV_E_FORMATETC	Invalid or unregistered Format specified in FORMATETC

OPC Specific Errors	Description
OPC_E_INVALIDHANDLE	The value of the handle is invalid. **Note**: a client should never pass an invalid handle to a server. If this error occurs, it is due to a programming error in the client or possibly in the server.
OPC_E_BADTYPE	The server cannot convert the data between the specified format/ requested data type and the canonical data type.
OPC_E_PUBLIC	The requested operation cannot be done on a public group.
OPC_E_BADRIGHTS	The Items AccessRights do not allow the operation.
OPC_E_UNKNOWNITEMID	The item ID is not defined in the server address space (on add or validate) or no longer exists in the server address space (for read or write).
OPC_E_INVALIDITEMID	The item ID doesn't conform to the server's syntax.
OPC_E_INVALIDFILTER	The filter string was not valid
OPC_E_UNKNOWNPATH	The item's access path is not known to the server.
OPC_E_RANGE	The value was out of range.
OPC_E_DUPLICATENAME	Duplicate name not allowed.
OPC_S_UNSUPPORTEDRATE	The server does not support the requested data rate but will use the closest available rate.
OPC_S_CLAMP	A value passed to WRITE was accepted but the output was clamped.
OPC_S_INUSE	The operation cannot be performed because the object is being referenced.
OPC_E_INVALIDCONFIGFILE	The server's configuration file is an invalid format.
OPC_E_NOTFOUND	The requested object (e.g. a public group) was not found.
OPC_E_INVALID_PID	The passed property ID is not valid for the item.
OPC_E_DEADBANDNOTSET	The item deadband has not been set for this item.
OPC_E_DEADBANDNOTSUPPORTED	The item does not support deadband.
OPC_E_NOBUFFERING	The server does not support buffering of data items that are collected at a faster rate than the group update rate
OPC_E_INVALIDCONTINUATIONPOINT	The continuation point is not valid.
OPC_S_DATAQUEUEOVERFLOW	Indicates that not every detected change has been returned for this item. This is an indicator that servers buffer reached its limit and had to purge out the oldest data. Only the most recent data is provided. The server should only remove the oldest data for those items that have newer samples available in the buffer. This will allow single samplings of older items to be returned to the client.

OPC Specific Errors	Description
OPC_E_INVALIDHANDLE	The value of the handle is invalid. **Note**: a client should never pass an invalid handle to a server. If this error occurs, it is due to a programming error in the client or possibly in the server.
OPC_E_RATENOTSET	Indicates that there is no sampling rate set for a particular item. In this case, the item defaults to the update rate of the group.
OPC_E_NOTSUPPORTED	If a client attempts to write any value, quality, timestamp combination and the server does not support the requested combination (which could be a single quantity such as just timestamp), then the server will not perform any write and will return this error code.

You will see in the appendix that these error codes use ITF_FACILITY. This means that they are context specific (i.e. OPC specific). The calling application should check first with the server providing the error (i.e. call GetErrorString).

Error codes (the low order word of the HRESULT) from 0000 to 0200 are reserved for Microsoft use (although some were inadvertently used for OPC 1.0 errors). Codes from 0200 through 7FFF are reserved for future OPC use. Codes from 8000 through FFFF can be vendor specific.

8 Appendix A - opcerror.h

The following is for information only and the associated text file should be used as the definitive reference, and additionally may be updated separately from the specification. The information should be taken from the associated text file and copied into a file named opcerror.h as opposed from this specification. To ensure that you obtain the associated text file that belongs to a specific version of the specification, the naming convention used is as follows:

opcerror_V.xy.h where V represents the major version and xy represents the minor version number from the cover of the specification. For example, if the specification version was 3.0, the text file name would be opcerror_3.00.h. If the specification version was 3.15, the text file name would be opcerror_3.15.h

```
//============================================================================
// TITLE: operror.h
//
// CONTENTS:
//
// Defines error codes for the Data Access specifications.
//
// (c) Copyright 1997-2003 The OPC Foundation
// ALL RIGHTS RESERVED.
//
// DISCLAIMER:
//  This code is provided by the OPC Foundation solely to assist in
//  understanding and use of the appropriate OPC Specification(s) and
//  may be used as set forth in the License Grant section of the OPC
//  Specification. This code is provided as-is and without warranty or
//  support of any sort and is subject to the Warranty and Liability
//  Disclaimers which appear in the printed OPC Specification.
//
// MODIFICATION LOG:
//
// Date       By    Notes
// ---------- ---   -----
// 1997/05/12 ACC   Removed Unused messages
//                  Added OPC_S_INUSE, OPC_E_INVALIDCONFIGFILE,
//                  OPC_E_NOTFOUND
// 1997/05/12 ACC   Added OPC_E_INVALID_PID
// 2002/08/12 CRT   Added new error codes for DA3.0
// 2003/01/02 RSA   Updated formatting. Added messages to proxy/stub
//                  resource block.
//
#ifndef __OPCERROR_H
#define __OPCERROR_H

#if _MSC_VER >= 1000
#pragma once
#endif

// The 'Facility' is set to the standard for COM interfaces or
// FACILITY_ITF (i.e. 0x004) The 'Code' is set in the range defined OPC
// Commmon for DA (i.e. 0x0400 to 0x04FF)
// Note that for backward compatibility not all existing codes use this
// range.
```

```
//
//  Values are 32 bit values layed out as follows:
//
//   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
//   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
//  +---+-+-+-----------------------+-------------------------------+
//  |Sev|C|R|     Facility          |               Code            |
//  +---+-+-+-----------------------+-------------------------------+
//
//  where
//
//      Sev - is the severity code
//
//          00 - Success
//          01 - Informational
//          10 - Warning
//          11 - Error
//
//      C - is the Customer code flag
//
//      R - is a reserved bit
//
//      Facility - is the facility code
//
//      Code - is the facility's status code
//
//
// Define the facility codes
//


//
// Define the severity codes
//


//
// MessageId: OPC_E_INVALIDHANDLE
//
// MessageText:
//
//  The value of the handle is invalid.
//
#define OPC_E_INVALIDHANDLE              ((HRESULT)0xC0040001L)

//
// MessageId: OPC_E_BADTYPE
//
// MessageText:
//
//  The server cannot convert the data between the specified format
and/or requested data type and the canonical data type.
//
#define OPC_E_BADTYPE                    ((HRESULT)0xC0040004L)

//
// MessageId: OPC_E_PUBLIC
```

```
//
// MessageText:
//
//  The requested operation cannot be done on a public group.
//
#define OPC_E_PUBLIC                     ((HRESULT)0xC0040005L)

//
// MessageId: OPC_E_BADRIGHTS
//
// MessageText:
//
//  The item's access rights do not allow the operation.
//
#define OPC_E_BADRIGHTS                  ((HRESULT)0xC0040006L)

//
// MessageId: OPC_E_UNKNOWNITEMID
//
// MessageText:
//
//  The item ID is not defined in the server address space or no longer
exists in the server address space.
//
#define OPC_E_UNKNOWNITEMID              ((HRESULT)0xC0040007L)

//
// MessageId: OPC_E_INVALIDITEMID
//
// MessageText:
//
//  The item ID does not conform to the server's syntax.
//
#define OPC_E_INVALIDITEMID              ((HRESULT)0xC0040008L)

//
// MessageId: OPC_E_INVALIDFILTER
//
// MessageText:
//
//  The filter string was not valid.
//
#define OPC_E_INVALIDFILTER              ((HRESULT)0xC0040009L)

//
// MessageId: OPC_E_UNKNOWNPATH
//
// MessageText:
//
//  The item's access path is not known to the server.
//
#define OPC_E_UNKNOWNPATH                ((HRESULT)0xC004000AL)

//
// MessageId: OPC_E_RANGE
```

```
//
// MessageText:
//
//  The value was out of range.
//
#define OPC_E_RANGE                      ((HRESULT)0xC004000BL)

//
// MessageId: OPC_E_DUPLICATENAME
//
// MessageText:
//
//  Duplicate name not allowed.
//
#define OPC_E_DUPLICATENAME              ((HRESULT)0xC004000CL)

//
// MessageId: OPC_S_UNSUPPORTEDRATE
//
// MessageText:
//
//  The server does not support the requested data rate but will use
the closest available rate.
//
#define OPC_S_UNSUPPORTEDRATE            ((HRESULT)0x0004000DL)

//
// MessageId: OPC_S_CLAMP
//
// MessageText:
//
//  A value passed to write was accepted but the output was clamped.
//
#define OPC_S_CLAMP                      ((HRESULT)0x0004000EL)

//
// MessageId: OPC_S_INUSE
//
// MessageText:
//
//  The operation cannot be performed because the object is bering
referenced.
//
#define OPC_S_INUSE                      ((HRESULT)0x0004000FL)

//
// MessageId: OPC_E_INVALIDCONFIGFILE
//
// MessageText:
//
//  The server's configuration file is an invalid format.
//
#define OPC_E_INVALIDCONFIGFILE          ((HRESULT)0xC0040010L)

//
```

```
// MessageId: OPC_E_NOTFOUND
//
// MessageText:
//
//  The requested object (e.g. a public group) was not found.
//
#define OPC_E_NOTFOUND                   ((HRESULT)0xC0040011L)

//
// MessageId: OPC_E_INVALID_PID
//
// MessageText:
//
//  The specified property ID is not valid for the item.
//
#define OPC_E_INVALID_PID                ((HRESULT)0xC0040203L)

//
// MessageId: OPC_E_DEADBANDNOTSET
//
// MessageText:
//
//  The item deadband has not been set for this item.
//
#define OPC_E_DEADBANDNOTSET             ((HRESULT)0xC0040400L)

//
// MessageId: OPC_E_DEADBANDNOTSUPPORTED
//
// MessageText:
//
//  The item does not support deadband.
//
#define OPC_E_DEADBANDNOTSUPPORTED       ((HRESULT)0xC0040401L)

//
// MessageId: OPC_E_NOBUFFERING
//
// MessageText:
//
//  The server does not support buffering of data items that are
collected at a faster rate than the group update rate.
//
#define OPC_E_NOBUFFERING                ((HRESULT)0xC0040402L)

//
// MessageId: OPC_E_INVALIDCONTINUATIONPOINT
//
// MessageText:
//
//  The continuation point is not valid.
//
#define OPC_E_INVALIDCONTINUATIONPOINT   ((HRESULT)0xC0040403L)

//
```

```
// MessageId: OPC_S_DATAQUEUEOVERFLOW
//
// MessageText:
//
//  Data Queue Overflow - Some value transitions were lost.
//
#define OPC_S_DATAQUEUEOVERFLOW          ((HRESULT)0x00040404L)

//
// MessageId: OPC_E_RATENOTSET
//
// MessageText:
//
//  Server does not support requested rate.
//
#define OPC_E_RATENOTSET                 ((HRESULT)0xC0040405L)

//
// MessageId: OPC_E_NOTSUPPORTED
//
// MessageText:
//
//  The server does not support writing of quality and/or timestamp.
//
#define OPC_E_NOTSUPPORTED               ((HRESULT)0xC0040406L)

#endif // ifndef __OPCERROR_H
```

9 Appendix B - Data Access IDL Specification

The following is for information only and the associated text file should be used as the definitive reference, and additionally may be updated separately from the specification. The information should be taken from the associated text file and copied into a file named opcda.idl as opposed from this specification. To ensure that you obtain the associated text file that belongs to a specific version of the specification, the naming convention used is as follows:

opcda_V.xy.idl where V represents the major version and xy represents the minor version number from the cover of the specification. For example, if the specification version was 3.0, the text file name would be opcda_3.00.idl. If the specification version was 3.15, the text file name would be opcda_3.15.idl

Use the command line MIDL //Oicf opcda.idl.

The resulting **OPCDA.H** file should be **included** in all clients and servers.

The resulting **OPCDA_I.C** file defines the interface IDs and should be **linked** into all clients and servers.

NOTE: This IDL file and the Proxy/Stub generated from it should NEVER be modified in any way. If you add vendor specific interfaces to your server (which is allowed) you must generate a SEPARATE vendor specific IDL file to describe only those interfaces and a separate vendor specific ProxyStub DLL to marshall only those interfaces.

Note: See the OPC Overview document (OPCOVW.DOC) for a listing and discussion of OPCCOMN.IDL.

```
//============================================================================
// TITLE: opcda.idl
//
// CONTENTS:
//
// Interface declarations for the OPC Data Access specifications.
//
// (c) Copyright 1997-2002 The OPC Foundation
// ALL RIGHTS RESERVED.
//
// DISCLAIMER:
//  This code is provided by the OPC Foundation solely to assist in
//  understanding and use of the appropriate OPC Specification(s) and may be
//  used as set forth in the License Grant section of the OPC Specification.
//  This code is provided as-is and without warranty or support of any sort
//  and is subject to the Warranty and Liability Disclaimers which appear
//  in the printed OPC Specification.
//
// MODIFICATION LOG:
//
// Date       By    Notes
// ---------- ---   -----
// 1997/05/12 ACC   fixed UNCERTAIN bits, add AsyncIO2, OPCDataCallback,
//                  OPCItemProperties,  BROWSE_TO
//
// 1998/06/19 ACC   change V2 uuids prior to final release
//                  to avoid conflict with 'old' OPCDA Automation uuids
//                  Change name of 3 methods on AsyncIO2 to
//                  Cancel2,SetEnable,GetEnable to eliminate conflicts
//
```

```
// 2002/02/10 CRT   Added IOPCItemDeadbandMgt Interface
//                         Added IOPCItemSamplingMgt Interface
//                         Added IOPCBrowse          Interface
//                         Added IOPCServer2         Interface
//                         Added IOPCItemIO          Interface
//                         Added new BAD Quality status mask
//
// 2002/08/09 CRT   Added IOPCITEMVQTstructure
//                         Moved #defines to Library section and "typed" them
//                         Added definition for Category Ids
//
// 2002/08/21 RSA   Added asynchrounous UUIDs. Fixed formatting.
//
// 2002/08/30 CRT   Added IOPCSyncIO2::Read
//                         Added IOPCServer2::RemoveGroupEx
//
// 2002/09/20 CRT   Added bit masks for Browse method
//
// 2002/10/03 CRT   Added IOPCAsyncIO3 Interface
//                         Added IOPCGroupStateMgt2 Interface
//
// 2003/03/03 RSA   Added fields to ensure natural byte alignment for new structures.
//

import "oaidl.idl";
import "ocidl.idl";
import "objidl.idl";

//=============================================================================
// Category ID declarations (defined as interfaces to ensure they show up in the
typelib).

[uuid(63D5F430-CFE4-11d1-B2C8-0060083BA1FB)] interface CATID_OPCDAServer10 : IUnknown {}
[uuid(63D5F432-CFE4-11d1-B2C8-0060083BA1FB)] interface CATID_OPCDAServer20 : IUnknown {}
[uuid(CC603642-66D7-48f1-B69A-B625E73652D7)] interface CATID_OPCDAServer30 : IUnknown {}
[uuid(3098EDA4-A006-48b2-A27F-247453959408)] interface CATID_XMLDAServer10 : IUnknown {}

cpp_quote("#define CATID_OPCDAServer10 IID_CATID_OPCDAServer10")
cpp_quote("#define CATID_OPCDAServer20 IID_CATID_OPCDAServer20")
cpp_quote("#define CATID_OPCDAServer30 IID_CATID_OPCDAServer30")
cpp_quote("#define CATID_XMLDAServer10 IID_CATID_XMLDAServer10")

//=============================================================================
// Structures, Typedefs and Enumerations.

typedef DWORD OPCHANDLE;

typedef enum tagOPCDATASOURCE
{
    OPC_DS_CACHE  = 1,
    OPC_DS_DEVICE
}
OPCDATASOURCE;

typedef enum tagOPCBROWSETYPE
{
    OPC_BRANCH = 1,
    OPC_LEAF,
    OPC_FLAT
}
OPCBROWSETYPE;

typedef enum tagOPCNAMESPACETYPE
{
    OPC_NS_HIERARCHIAL = 1,
    OPC_NS_FLAT
}
OPCNAMESPACETYPE;
```

```
typedef enum tagOPCBROWSEDIRECTION
{
    OPC_BROWSE_UP = 1,
    OPC_BROWSE_DOWN,
    OPC_BROWSE_TO
}
OPCBROWSEDIRECTION;

typedef enum tagOPCEUTYPE
{
    OPC_NOENUM = 0,
    OPC_ANALOG,
    OPC_ENUMERATED
}
OPCEUTYPE;

typedef enum tagOPCSERVERSTATE
{
    OPC_STATUS_RUNNING = 1,
    OPC_STATUS_FAILED,
    OPC_STATUS_NOCONFIG,
    OPC_STATUS_SUSPENDED,
    OPC_STATUS_TEST,
    OPC_STATUS_COMM_FAULT
}
OPCSERVERSTATE;

typedef enum tagOPCENUMSCOPE
{
    OPC_ENUM_PRIVATE_CONNECTIONS = 1,
    OPC_ENUM_PUBLIC_CONNECTIONS,
    OPC_ENUM_ALL_CONNECTIONS,
    OPC_ENUM_PRIVATE,
    OPC_ENUM_PUBLIC,
    OPC_ENUM_ALL
}
OPCENUMSCOPE;

typedef struct tagOPCGROUPHEADER
{
    DWORD       dwSize;
    DWORD       dwItemCount;
    OPCHANDLE   hClientGroup;
    DWORD       dwTransactionID;
    HRESULT     hrStatus;
}
OPCGROUPHEADER;

typedef struct tagOPCITEMHEADER1
{
    OPCHANDLE   hClient;
    DWORD       dwValueOffset;
    WORD        wQuality;
    WORD        wReserved;
    FILETIME    ftTimeStampItem;
}
OPCITEMHEADER1;

typedef struct tagOPCITEMHEADER2
{
    OPCHANDLE   hClient;
    DWORD       dwValueOffset;
    WORD        wQuality;
    WORD        wReserved;
}
OPCITEMHEADER2;

typedef struct tagOPCGROUPHEADERWRITE
{
```

```
    DWORD       dwItemCount;
    OPCHANDLE   hClientGroup;
    DWORD       dwTransactionID;
    HRESULT     hrStatus;
}
OPCGROUPHEADERWRITE;

typedef struct tagOPCITEMHEADERWRITE
{
    OPCHANDLE   hClient;
    HRESULT     dwError;
}
OPCITEMHEADERWRITE;

typedef struct tagOPCITEMSTATE
{
    OPCHANDLE   hClient;
    FILETIME    ftTimeStamp;
    WORD        wQuality;
    WORD        wReserved;
    VARIANT     vDataValue;
}
OPCITEMSTATE;

typedef struct tagOPCSERVERSTATUS
{
           FILETIME       ftStartTime;
           FILETIME       ftCurrentTime;
           FILETIME       ftLastUpdateTime;
           OPCSERVERSTATE dwServerState;
           DWORD          dwGroupCount;
           DWORD          dwBandWidth;
           WORD           wMajorVersion;
           WORD           wMinorVersion;
           WORD           wBuildNumber;
           WORD           wReserved;
  [string] LPWSTR         szVendorInfo;
}
OPCSERVERSTATUS;

typedef struct tagOPCITEMDEF
{
  [string]              LPWSTR     szAccessPath;
  [string]              LPWSTR     szItemID;
                        BOOL       bActive ;
                        OPCHANDLE  hClient;
                        DWORD      dwBlobSize;
  [size_is(dwBlobSize)] BYTE*      pBlob;
                        VARTYPE    vtRequestedDataType;
                        WORD       wReserved;
}
OPCITEMDEF;

typedef struct tagOPCITEMATTRIBUTES
{
  [string]              LPWSTR     szAccessPath;
  [string]              LPWSTR     szItemID;
                        BOOL       bActive;
                        OPCHANDLE  hClient;
                        OPCHANDLE  hServer;
                        DWORD      dwAccessRights;
                        DWORD      dwBlobSize;
  [size_is(dwBlobSize)] BYTE*      pBlob;
                        VARTYPE    vtRequestedDataType;
                        VARTYPE    vtCanonicalDataType;
                        OPCEUTYPE  dwEUType;
                        VARIANT    vEUInfo;
}
OPCITEMATTRIBUTES;
```

```
typedef struct tagOPCITEMRESULT
{
                        OPCHANDLE  hServer;
                        VARTYPE    vtCanonicalDataType;
                        WORD       wReserved;
                        DWORD      dwAccessRights;
                        DWORD      dwBlobSize;
  [size_is(dwBlobSize)] BYTE*      pBlob;
}
OPCITEMRESULT;

typedef struct tagOPCITEMPROPERTY
{
                   VARTYPE     vtDataType;
            WORD     wReserved;
                   DWORD       dwPropertyID;
  [string] LPWSTR      szItemID;
  [string] LPWSTR      szDescription;
               VARIANT vValue;
                   HRESULT     hrErrorID;
                   DWORD    dwReserved;
}
OPCITEMPROPERTY;

typedef struct tagOPCITEMPROPERTIES
{
                                            HRESULT          hrErrorID;
                                            DWORD              dwNumProperties;
  [size_is(dwNumProperties)] OPCITEMPROPERTY* pItemProperties;
                                                        DWORD            dwReserved;
}
OPCITEMPROPERTIES;

typedef struct  tagOPCBROWSEELEMENT
{
  [string]     LPWSTR            szName;
  [string]     LPWSTR            szItemID;
            DWORD             dwFlagValue;
                      DWORD             dwReserved;
            OPCITEMPROPERTIES ItemProperties;
}
OPCBROWSEELEMENT;

typedef struct tagOPCITEMVQT
{
    VARIANT     vDataValue;
    BOOL        bQualitySpecified;
    WORD        wQuality;
    WORD     wReserved;
    BOOL        bTimeStampSpecified;
    DWORD    dwReserved;
    FILETIME ftTimeStamp;
}
OPCITEMVQT;

typedef enum tagOPCBROWSEFILTER
{
        OPC_BROWSE_FILTER_ALL = 1,
        OPC_BROWSE_FILTER_BRANCHES,
        OPC_BROWSE_FILTER_ITEMS,
}
OPCBROWSEFILTER;

//============================================================================
// IOPCServer

[
    object,
```

```
    uuid(39c13a4d-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCServer : IUnknown
{
    HRESULT AddGroup(
        [in, string]        LPCWSTR    szName,
        [in]                BOOL       bActive,
        [in]                DWORD      dwRequestedUpdateRate,
        [in]                OPCHANDLE  hClientGroup,
        [unique, in]        LONG*      pTimeBias,
        [unique, in]        FLOAT*     pPercentDeadband,
        [in]                DWORD      dwLCID,
        [out]               OPCHANDLE* phServerGroup,
        [out]               DWORD*     pRevisedUpdateRate,
        [in]                REFIID     riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
    );

    HRESULT GetErrorString(
        [in]          HRESULT dwError,
        [in]          LCID    dwLocale,
        [out, string] LPWSTR* ppString
    );

    HRESULT GetGroupByName(
        [in, string]        LPCWSTR    szName,
        [in]                REFIID     riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
    );

    HRESULT GetStatus(
        [out] OPCSERVERSTATUS** ppServerStatus
    );

    HRESULT RemoveGroup(
        [in] OPCHANDLE hServerGroup,
        [in] BOOL      bForce
    );

    HRESULT CreateGroupEnumerator(
        [in]                OPCENUMSCOPE dwScope,
        [in]                REFIID       riid,
        [out, iid_is(riid)] LPUNKNOWN*   ppUnk
    );
}

//============================================================================
// IOPCServerPublicGroups

[
    object,
    uuid(39c13a4e-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCServerPublicGroups : IUnknown
{
    HRESULT GetPublicGroupByName(
        [in, string]        LPCWSTR    szName,
        [in]                REFIID     riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
    );

    HRESULT RemovePublicGroup(
        [in] OPCHANDLE hServerGroup,
        [in] BOOL      bForce
    );
}
```

```
//=============================================================================
// IOPCBrowseServerAddressSpace

[
    object,
    uuid(39c13a4f-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCBrowseServerAddressSpace: IUnknown
{
    HRESULT QueryOrganization(
        [out] OPCNAMESPACETYPE* pNameSpaceType
    );

    HRESULT ChangeBrowsePosition(
        [in]          OPCBROWSEDIRECTION dwBrowseDirection,
        [in, string] LPCWSTR             szString
    );

    HRESULT BrowseOPCItemIDs(
        [in]          OPCBROWSETYPE dwBrowseFilterType,
        [in, string] LPCWSTR       szFilterCriteria,
        [in]          VARTYPE       vtDataTypeFilter,
        [in]          DWORD         dwAccessRightsFilter,
        [out]         LPENUMSTRING* ppIEnumString
    );

    HRESULT GetItemID(
        [in]           LPWSTR  szItemDataID,
        [out, string] LPWSTR* szItemID
    );

    HRESULT BrowseAccessPaths(
        [in, string] LPCWSTR       szItemID,
        [out]         LPENUMSTRING* ppIEnumString
    );
}

//=============================================================================
// IOPCGroupStateMgt

[
    object,
    uuid(39c13a50-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCGroupStateMgt : IUnknown
{
    HRESULT GetState(
        [out]          DWORD*     pUpdateRate,
        [out]          BOOL*      pActive,
        [out, string] LPWSTR*    ppName,
        [out]          LONG*      pTimeBias,
        [out]          FLOAT*     pPercentDeadband,
        [out]          DWORD*     pLCID,
        [out]          OPCHANDLE* phClientGroup,
        [out]          OPCHANDLE* phServerGroup
    );

    HRESULT SetState(
        [unique, in]  DWORD*     pRequestedUpdateRate,
        [out]          DWORD*     pRevisedUpdateRate,
        [unique, in]  BOOL*      pActive,
        [unique, in]  LONG*      pTimeBias,
        [unique, in]  FLOAT*     pPercentDeadband,
        [unique, in]  DWORD*     pLCID,
        [unique, in]  OPCHANDLE* phClientGroup
    );
```

```
    HRESULT SetName(
        [in, string] LPCWSTR szName
    );

    HRESULT CloneGroup(
        [in, string]        LPCWSTR    szName,
        [in]                REFIID     riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
    );
}

//============================================================================
// IOPCPublicGroupStateMgt

[
    object,
    uuid(39c13a51-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCPublicGroupStateMgt : IUnknown
{
    HRESULT GetState(
        [out] BOOL* pPublic
    );

    HRESULT MoveToPublic(
        void
    );
}

//============================================================================
// IOPCSyncIO

[
    object,
    uuid(39c13a52-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCSyncIO : IUnknown
{
    HRESULT Read(
        [in]                      OPCDATASOURCE  dwSource,
        [in]                      DWORD          dwCount,
        [in, size_is(dwCount)]    OPCHANDLE*     phServer,
        [out, size_is(,dwCount)] OPCITEMSTATE** ppItemValues,
        [out, size_is(,dwCount)] HRESULT**      ppErrors
    );

    HRESULT Write(
        [in]                      DWORD      dwCount,
        [in, size_is(dwCount)]    OPCHANDLE* phServer,
        [in, size_is(dwCount)]    VARIANT*   pItemValues,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );
}

//============================================================================
// IOPCAsyncIO

[
    object,
    uuid(39c13a53-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCAsyncIO : IUnknown
{
    HRESULT Read(
        [in]                      DWORD         dwConnection,
        [in]                      OPCDATASOURCE dwSource,
```

```
        [in]                    DWORD         dwCount,
        [in, size_is(dwCount)]  OPCHANDLE*    phServer,
        [out]                   DWORD*        pTransactionID,
        [out, size_is(,dwCount)] HRESULT**    ppErrors
    );

    HRESULT Write(
        [in]                    DWORD      dwConnection,
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)]  OPCHANDLE* phServer,
        [in, size_is(dwCount)]  VARIANT*   pItemValues,
        [out]                   DWORD*     pTransactionID,
        [out, size_is(,dwCount)] HRESULT** ppErrors
    );

    HRESULT Refresh(
        [in]  DWORD         dwConnection,
        [in]  OPCDATASOURCE dwSource,
        [out] DWORD*        pTransactionID
    );

    HRESULT Cancel(
        [in] DWORD dwTransactionID
    );
}

//============================================================================
// IOPCItemMgt

[
    object,
    uuid(39c13a54-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCItemMgt: IUnknown
{
    HRESULT AddItems(
        [in]                    DWORD           dwCount,
        [in, size_is(dwCount)]  OPCITEMDEF*     pItemArray,
        [out, size_is(,dwCount)] OPCITEMRESULT** ppAddResults,
        [out, size_is(,dwCount)] HRESULT**      ppErrors
    );

    HRESULT ValidateItems(
        [in]                    DWORD           dwCount,
        [in, size_is(dwCount)]  OPCITEMDEF*     pItemArray,
        [in]                    BOOL            bBlobUpdate,
        [out, size_is(,dwCount)] OPCITEMRESULT** ppValidationResults,
        [out, size_is(,dwCount)] HRESULT**      ppErrors
    );

    HRESULT RemoveItems(
        [in]                       DWORD      dwCount,
        [in, size_is(dwCount)]     OPCHANDLE* phServer,
        [out, size_is(,dwCount)]   HRESULT**  ppErrors
    );

    HRESULT SetActiveState(
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)]  OPCHANDLE* phServer,
        [in]                    BOOL       bActive,
        [out, size_is(,dwCount)] HRESULT** ppErrors
    );

    HRESULT SetClientHandles(
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)]  OPCHANDLE* phServer,
        [in, size_is(dwCount)]  OPCHANDLE* phClient,
        [out, size_is(,dwCount)] HRESULT** ppErrors
```

```
    );

    HRESULT SetDatatypes(
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)]  OPCHANDLE* phServer,
        [in, size_is(dwCount)]  VARTYPE*   pRequestedDatatypes,
        [out, size_is(,dwCount)] HRESULT** ppErrors
    );

    HRESULT CreateEnumerator(
        [in]                REFIID     riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
    );
}

//=============================================================================
// IEnumOPCItemAttributes

[
    object,
    uuid(39c13a55-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IEnumOPCItemAttributes : IUnknown
{
    HRESULT Next(
        [in]                          ULONG                celt,
        [out, size_is(,*pceltFetched)] OPCITEMATTRIBUTES** ppItemArray,
        [out]                         ULONG*               pceltFetched
    );

    HRESULT Skip(
        [in] ULONG celt
    );

    HRESULT Reset(
        void
    );

    HRESULT Clone(
        [out] IEnumOPCItemAttributes** ppEnumItemAttributes
    );
}

//=============================================================================
// IOPCDataCallback

[
    object,
    uuid(39c13a70-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCDataCallback : IUnknown
{
    HRESULT OnDataChange(
        [in]                    DWORD      dwTransid,
        [in]                    OPCHANDLE  hGroup,
        [in]                    HRESULT    hrMasterquality,
        [in]                    HRESULT    hrMastererror,
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)] OPCHANDLE* phClientItems,
        [in, size_is(dwCount)] VARIANT*   pvValues,
        [in, size_is(dwCount)] WORD*      pwQualities,
        [in, size_is(dwCount)] FILETIME*  pftTimeStamps,
        [in, size_is(dwCount)] HRESULT*   pErrors
    );

    HRESULT OnReadComplete(
        [in]                    DWORD      dwTransid,
```

```
        [in]                    OPCHANDLE  hGroup,
        [in]                    HRESULT    hrMasterquality,
        [in]                    HRESULT    hrMastererror,
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)] OPCHANDLE* phClientItems,
        [in, size_is(dwCount)] VARIANT*   pvValues,
        [in, size_is(dwCount)] WORD*      pwQualities,
        [in, size_is(dwCount)] FILETIME*  pftTimeStamps,
        [in, size_is(dwCount)] HRESULT*   pErrors
    );

    HRESULT OnWriteComplete(
        [in]                    DWORD      dwTransid,
        [in]                    OPCHANDLE  hGroup,
        [in]                    HRESULT    hrMastererr,
        [in]                    DWORD      dwCount,
        [in, size_is(dwCount)] OPCHANDLE* pClienthandles,
        [in, size_is(dwCount)] HRESULT*   pErrors
    );

    HRESULT OnCancelComplete(
        [in] DWORD     dwTransid,
        [in] OPCHANDLE hGroup
    );
}

//============================================================================
// IOPCAsyncIO2

[
    object,
    uuid(39c13a71-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCAsyncIO2 : IUnknown
{
    HRESULT Read(
        [in]                       DWORD      dwCount,
        [in, size_is(dwCount)]     OPCHANDLE* phServer,
        [in]                       DWORD      dwTransactionID,
        [out]                      DWORD*     pdwCancelID,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT Write(
        [in]                       DWORD      dwCount,
        [in, size_is(dwCount)]     OPCHANDLE* phServer,
        [in, size_is(dwCount)]     VARIANT*   pItemValues,
        [in]                       DWORD      dwTransactionID,
        [out]                      DWORD*     pdwCancelID,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );


    HRESULT Refresh2(
        [in]  OPCDATASOURCE dwSource,
        [in]  DWORD         dwTransactionID,
        [out] DWORD*        pdwCancelID
    );

    HRESULT Cancel2(
        [in] DWORD dwCancelID
    );

    HRESULT SetEnable(
        [in] BOOL bEnable
    );

    HRESULT GetEnable(
```

```
        [out] BOOL* pbEnable
    );
}

//============================================================================
// IOPCItemProperties

[
    object,
    uuid(39c13a72-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCItemProperties : IUnknown
{
    HRESULT QueryAvailableProperties (
        [in]                       LPWSTR    szItemID,
        [out]                      DWORD*    pdwCount,
        [out, size_is(,*pdwCount)] DWORD**   ppPropertyIDs,
        [out, size_is(,*pdwCount)] LPWSTR**  ppDescriptions,
        [out, size_is(,*pdwCount)] VARTYPE** ppvtDataTypes
    );

    HRESULT GetItemProperties (
        [in]                     LPWSTR    szItemID,
        [in]                     DWORD     dwCount,
        [in, size_is(dwCount)]   DWORD*    pdwPropertyIDs,
        [out, size_is(,dwCount)] VARIANT** ppvData,
        [out, size_is(,dwCount)] HRESULT** ppErrors
    );

    HRESULT LookupItemIDs(
        [in]                              LPWSTR    szItemID,
        [in]                              DWORD     dwCount,
        [in, size_is(dwCount)]            DWORD*    pdwPropertyIDs,
        [out, string, size_is(,dwCount)] LPWSTR**  ppszNewItemIDs,
        [out, size_is(,dwCount)]          HRESULT** ppErrors
    );
}

//============================================================================
// IOPCItemDeadbandMgt

[
    object,
    uuid(5946DA93-8B39-4ec8-AB3D-AA73DF5BC86F),
    pointer_default(unique)
]
interface IOPCItemDeadbandMgt : IUnknown
{
    HRESULT SetItemDeadband(
        [in]                              DWORD      dwCount,
        [in, size_is(dwCount)]         OPCHANDLE* phServer,
        [in, size_is(dwCount)]         FLOAT*     pPercentDeadband,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT GetItemDeadband(
        [in]                              DWORD         dwCount,
        [in, size_is(dwCount)]         OPCHANDLE* phServer,
        [out, size_is(,dwCount)] FLOAT**    ppPercentDeadband,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT ClearItemDeadband(
        [in]                              DWORD      dwCount,
        [in, size_is(dwCount)]         OPCHANDLE* phServer,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );
}
```

```
//============================================================================
// IOPCItemSamplingMgt

[
    object,
    uuid(3E22D313-F08B-41a5-86C8-95E95CB49FFC),
    pointer_default(unique)
]
interface IOPCItemSamplingMgt : IUnknown
{
    HRESULT SetItemSamplingRate (
        [in]                                DWORD      dwCount,
        [in, size_is(dwCount)]   OPCHANDLE* phServer,
        [in, size_is(dwCount)]   DWORD*     pdwRequestedSamplingRate,
        [out, size_is(,dwCount)] DWORD**    ppdwRevisedSamplingRate,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT GetItemSamplingRate (
        [in]                                DWORD      dwCount,
        [in, size_is(dwCount)]   OPCHANDLE* phServer,
        [out, size_is(,dwCount)] DWORD**    ppdwSamplingRate,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT ClearItemSamplingRate(
        [in]                                DWORD      dwCount,
        [in, size_is(dwCount)]         OPCHANDLE* phServer,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT SetItemBufferEnable(
        [in]                        DWORD      dwCount,
        [in, size_is(dwCount)]   OPCHANDLE* phServer,
        [in, size_is(dwCount)]         BOOL*       pbEnable,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT GetItemBufferEnable(
        [in]                        DWORD      dwCount,
        [in, size_is(dwCount)]   OPCHANDLE* phServer,
        [out, size_is(,dwCount)] BOOL**     ppbEnable,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );
}

//============================================================================
// IOPCBrowse

[
    object,
    uuid(39227004-A18F-4b57-8B0A-5235670F4468),
    pointer_default(unique)
]
interface IOPCBrowse : IUnknown
{
    HRESULT GetProperties(
        [in]                                DWORD                   dwItemCount,
        [in, string, size_is(dwItemCount)] LPWSTR*             pszItemIDs,
        [in]                                                BOOL
bReturnPropertyValues,
        [in]                                                DWORD               dwPropertyCount,
        [in, size_is(dwPropertyCount)]     DWORD*              pdwPropertyIDs,
        [out, size_is(,dwItemCount)]       OPCITEMPROPERTIES** ppItemProperties
    );

    HRESULT Browse(
        [in, string]                   LPWSTR             szItemID,
```

```
        [in,out, string]                LPWSTR*             pszContinuationPoint,
        [in]                            DWORD               dwMaxElementsReturned,
               [in]                            OPCBROWSEFILTER     dwBrowseFilter,
        [in, string]                    LPWSTR              szElementNameFilter,
        [in, string]                    LPWSTR              szVendorFilter,
        [in]                            BOOL                bReturnAllProperties,
        [in]                            BOOL                bReturnPropertyValues,
        [in]                            DWORD               dwPropertyCount,
        [in, size_is(dwPropertyCount)] DWORD*              pdwPropertyIDs,
        [out]                           BOOL*               pbMoreElements,
        [out]                           DWORD*              pdwCount,
        [out, size_is(,*pdwCount)]      OPCBROWSEELEMENT** ppBrowseElements
    );
}

//==========================================================================
// IOPCItemIO

[
    object,
    uuid(85C0B427-2893-4cbc-BD78-E5FC5146F08F),
    pointer_default(unique)
]
interface IOPCItemIO: IUnknown
{
    HRESULT Read(
        [in]                        DWORD      dwCount,
        [in, size_is(dwCount)]   LPCWSTR*   pszItemIDs,
        [in, size_is(dwCount)]   DWORD*     pdwMaxAge,
        [out, size_is(,dwCount)] VARIANT**  ppvValues,
        [out, size_is(,dwCount)] WORD**     ppwQualities,
        [out, size_is(,dwCount)] FILETIME** ppftTimeStamps,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT WriteVQT(
        [in]                        DWORD       dwCount,
        [in, size_is(dwCount)]   LPCWSTR*    pszItemIDs,
        [in, size_is(dwCount)]   OPCITEMVQT* pItemVQT,
        [out, size_is(,dwCount)] HRESULT**   ppErrors
    );
}

//==========================================================================
// IOPCSyncIO2

[
    object,
    uuid(730F5F0F-55B1-4c81-9E18-FF8A0904E1FA),
    pointer_default(unique)
]
interface IOPCSyncIO2: IOPCSyncIO
{
    HRESULT ReadMaxAge(
        [in]                        DWORD      dwCount,
        [in, size_is(dwCount)]   OPCHANDLE* phServer,
        [in, size_is(dwCount)]   DWORD*     pdwMaxAge,
        [out, size_is(,dwCount)] VARIANT**  ppvValues,
        [out, size_is(,dwCount)] WORD**     ppwQualities,
        [out, size_is(,dwCount)] FILETIME** ppftTimeStamps,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT WriteVQT (
        [in]  DWORD dwCount,
        [in, size_is(dwCount)]   OPCHANDLE*  phServer,
        [in, size_is(dwCount)]   OPCITEMVQT* pItemVQT,
        [out, size_is(,dwCount)] HRESULT**   ppErrors
    );
```

```
}

//=============================================================================
// IOPCAsyncIO3

[
  object,
  uuid(0967B97B-36EF-423e-B6F8-6BFF1E40D39D),
  pointer_default(unique)
]
interface IOPCAsyncIO3: IOPCAsyncIO2
{
    HRESULT ReadMaxAge(
        [in]                      DWORD      dwCount,
        [in, size_is(dwCount)]    OPCHANDLE* phServer,
        [in, size_is(dwCount)]    DWORD*     pdwMaxAge,
        [in]                      DWORD      dwTransactionID,
        [out]                     DWORD*     pdwCancelID,
        [out, size_is(,dwCount)] HRESULT**  ppErrors
    );

    HRESULT WriteVQT(
        [in]                      DWORD       dwCount,
        [in, size_is(dwCount)]    OPCHANDLE*  phServer,
        [in, size_is(dwCount)]    OPCITEMVQT* pItemVQT,
        [in]                      DWORD       dwTransactionID,
        [out]                     DWORD*      pdwCancelID,
        [out, size_is(,dwCount)] HRESULT**   ppErrors
    );

    HRESULT RefreshMaxAge(
        [in]  DWORD  dwMaxAge,
        [in]  DWORD  dwTransactionID,
        [out] DWORD* pdwCancelID
    );
}

//=============================================================================
// IOPCGroupStateMgt2

[
       uuid(8E368666-D72E-4f78-87ED-647611C61C9F),
       pointer_default(unique)
]
interface IOPCGroupStateMgt2 : IOPCGroupStateMgt
{
    HRESULT SetKeepAlive(
        [in]  DWORD  dwKeepAliveTime,
        [out] DWORD* pdwRevisedKeepAliveTime
    );

    HRESULT GetKeepAlive(
        [out] DWORD* pdwKeepAliveTime
    );
}

//=============================================================================
// Type Library

[
    uuid(3B540B51-0378-4551-ADCC-EA9B104302BF),
    version(3.00),
    helpstring("OPC Data Access 3.00 Type Library")
]
library OPCDA
{
    //=========================================================================
    // Category IDs
```

```
    interface CATID_OPCDAServer10;
    interface CATID_OPCDAServer20;
    interface CATID_OPCDAServer30;
    interface CATID_XMLDAServer10;

    //=============================================================================
    // Constants

        module Constants
        {
                // category description strings.
                const LPCWSTR OPC_CATEGORY_DESCRIPTION_DA10    = L"OPC Data Access Servers
Version 1.0";
                const LPCWSTR OPC_CATEGORY_DESCRIPTION_DA20    = L"OPC Data Access Servers
Version 2.0";
                const LPCWSTR OPC_CATEGORY_DESCRIPTION_DA30    = L"OPC Data Access Servers
Version 3.0";
                const LPCWSTR OPC_CATEGORY_DESCRIPTION_XMLDA10 = L"OPC XML Data Access
Servers Version 1.0";

                // values for access rights mask.
                const DWORD OPC_READABLE           = 0x01;
                const DWORD OPC_WRITEABLE          = 0x02;

                // values for browse element flags.
                const DWORD OPC_BROWSE_HASCHILDREN = 0x01;
                const DWORD OPC_BROWSE_ISITEM      = 0x02;
        }

        //=============================================================================
    // Qualities

        module Qualities
        {
                // Values for fields in the quality word
         const WORD OPC_QUALITY_MASK                     = 0xC0;
         const WORD OPC_STATUS_MASK                      = 0xFC;
                const WORD OPC_LIMIT_MASK                        = 0x03;

                // Values for QUALITY_MASK bit field
                const WORD OPC_QUALITY_BAD                       = 0x00;
                const WORD OPC_QUALITY_UNCERTAIN                 = 0x40;
                const WORD OPC_QUALITY_GOOD                      = 0xC0;

                // STATUS_MASK Values for Quality = BAD
                const WORD OPC_QUALITY_CONFIG_ERROR                      = 0x04;
                const WORD OPC_QUALITY_NOT_CONNECTED                     = 0x08;
                const WORD OPC_QUALITY_DEVICE_FAILURE                    = 0x0c;
                const WORD OPC_QUALITY_SENSOR_FAILURE                    = 0x10;
                const WORD OPC_QUALITY_LAST_KNOWN                        = 0x14;
                const WORD OPC_QUALITY_COMM_FAILURE                      = 0x18;
                const WORD OPC_QUALITY_OUT_OF_SERVICE                    = 0x1C;
                const WORD OPC_QUALITY_WAITING_FOR_INITIAL_DATA = 0x20;

                // STATUS_MASK Values for Quality = UNCERTAIN
                const WORD OPC_QUALITY_LAST_USABLE               = 0x44;
                const WORD OPC_QUALITY_SENSOR_CAL                = 0x50;
                const WORD OPC_QUALITY_EGU_EXCEEDED              = 0x54;
                const WORD OPC_QUALITY_SUB_NORMAL                = 0x58;

                // STATUS_MASK Values for Quality = GOOD
                const WORD OPC_QUALITY_LOCAL_OVERRIDE            = 0xD8;

                // Values for Limit Bitfield
                const WORD OPC_LIMIT_OK                          = 0x00;
                const WORD OPC_LIMIT_LOW                         = 0x01;
                const WORD OPC_LIMIT_HIGH                        = 0x02;
                const WORD OPC_LIMIT_CONST                       = 0x03;
        }
```

```
        //============================================================================
    // Properties

        module Properties
        {
                // property ids.
                const DWORD OPC_PROPERTY_DATATYPE                   = 1;
                const DWORD OPC_PROPERTY_VALUE                      = 2;
                const DWORD OPC_PROPERTY_QUALITY                    = 3;
                const DWORD OPC_PROPERTY_TIMESTAMP                  = 4;
                const DWORD OPC_PROPERTY_ACCESS_RIGHTS              = 5;
                const DWORD OPC_PROPERTY_SCAN_RATE                  = 6;
                const DWORD OPC_PROPERTY_EU_TYPE                    = 7;
                const DWORD OPC_PROPERTY_EU_INFO                    = 8;
                const DWORD OPC_PROPERTY_EU_UNITS                   = 100;
                const DWORD OPC_PROPERTY_DESCRIPTION                = 101;
                const DWORD OPC_PROPERTY_HIGH_EU                    = 102;
                const DWORD OPC_PROPERTY_LOW_EU                     = 103;
                const DWORD OPC_PROPERTY_HIGH_IR                    = 104;
                const DWORD OPC_PROPERTY_LOW_IR                     = 105;
                const DWORD OPC_PROPERTY_CLOSE_LABEL                = 106;
                const DWORD OPC_PROPERTY_OPEN_LABEL                 = 107;
                const DWORD OPC_PROPERTY_TIMEZONE                   = 108;
                const DWORD OPC_PROPERTY_CONDITION_STATUS           = 300;
                const DWORD OPC_PROPERTY_ALARM_QUICK_HELP           = 301;
                const DWORD OPC_PROPERTY_ALARM_AREA_LIST            = 302;
                const DWORD OPC_PROPERTY_PRIMARY_ALARM_AREA         = 303;
                const DWORD OPC_PROPERTY_CONDITION_LOGIC            = 304;
                const DWORD OPC_PROPERTY_LIMIT_EXCEEDED             = 305;
                const DWORD OPC_PROPERTY_DEADBAND                   = 306;
                const DWORD OPC_PROPERTY_HIHI_LIMIT                 = 307;
                const DWORD OPC_PROPERTY_HI_LIMIT                   = 308;
                const DWORD OPC_PROPERTY_LO_LIMIT                   = 309;
                const DWORD OPC_PROPERTY_LOLO_LIMIT                 = 310;
                const DWORD OPC_PROPERTY_CHANGE_RATE_LIMIT          = 311;
                const DWORD OPC_PROPERTY_DEVIATION_LIMIT            = 312;
                const DWORD OPC_PROPERTY_SOUND_FILE                 = 313;

                // property descriptions.
                const LPCWSTR OPC_PROPERTY_DESC_DATATYPE            = L"Item Canonical Data
Type";
                const LPCWSTR OPC_PROPERTY_DESC_VALUE               = L"Item Value";
                const LPCWSTR OPC_PROPERTY_DESC_QUALITY             = L"Item Quality";
                const LPCWSTR OPC_PROPERTY_DESC_TIMESTAMP           = L"Item Timestamp";
                const LPCWSTR OPC_PROPERTY_DESC_ACCESS_RIGHTS       = L"Item Access
Rights";
                const LPCWSTR OPC_PROPERTY_DESC_SCAN_RATE           = L"Server Scan Rate";
                const LPCWSTR OPC_PROPERTY_DESC_EU_TYPE             = L"Item EU Type";
                const LPCWSTR OPC_PROPERTY_DESC_EU_INFO             = L"Item EU Info";
                const LPCWSTR OPC_PROPERTY_DESC_EU_UNITS            = L"EU Units";
                const LPCWSTR OPC_PROPERTY_DESC_DESCRIPTION         = L"Item Description";
                const LPCWSTR OPC_PROPERTY_DESC_HIGH_EU             = L"High EU";
                const LPCWSTR OPC_PROPERTY_DESC_LOW_EU              = L"Low EU";
                const LPCWSTR OPC_PROPERTY_DESC_HIGH_IR             = L"High Instrument
Range";
                const LPCWSTR OPC_PROPERTY_DESC_LOW_IR              = L"Low Instrument
Range";
                const LPCWSTR OPC_PROPERTY_DESC_CLOSE_LABEL         = L"Contact Close
Label";
                const LPCWSTR OPC_PROPERTY_DESC_OPEN_LABEL          = L"Contact Open
Label";
                const LPCWSTR OPC_PROPERTY_DESC_TIMEZONE            = L"Item Timezone";
                const LPCWSTR OPC_PROPERTY_DESC_CONDITION_STATUS    = L"Condition Status";
                const LPCWSTR OPC_PROPERTY_DESC_ALARM_QUICK_HELP    = L"Alarm Quick Help";
                const LPCWSTR OPC_PROPERTY_DESC_ALARM_AREA_LIST     = L"Alarm Area List";
                const LPCWSTR OPC_PROPERTY_DESC_PRIMARY_ALARM_AREA = L"Primary Alarm
Area";
                const LPCWSTR OPC_PROPERTY_DESC_CONDITION_LOGIC     = L"Condition Logic";
```

```
                const LPCWSTR OPC_PROPERTY_DESC_LIMIT_EXCEEDED     = L"Limit Exceeded";
                const LPCWSTR OPC_PROPERTY_DESC_DEADBAND           = L"Deadband";
                const LPCWSTR OPC_PROPERTY_DESC_HIHI_LIMIT         = L"HiHi Limit";
                const LPCWSTR OPC_PROPERTY_DESC_HI_LIMIT           = L"Hi Limit";
                const LPCWSTR OPC_PROPERTY_DESC_LO_LIMIT           = L"Lo Limit";
                const LPCWSTR OPC_PROPERTY_DESC_LOLO_LIMIT         = L"LoLo Limit";
                const LPCWSTR OPC_PROPERTY_DESC_CHANGE_RATE_LIMIT  = L"Rate of Change
Limit";
                const LPCWSTR OPC_PROPERTY_DESC_DEVIATION_LIMIT    = L"Deviation Limit";
                const LPCWSTR OPC_PROPERTY_DESC_SOUND_FILE         = L"Sound File";
        }

    //============================================================================
    // Synchronous Interfaces

    interface IOPCServer;
    interface IOPCServerPublicGroups;
    interface IOPCBrowseServerAddressSpace;
    interface IOPCGroupStateMgt;
    interface IOPCPublicGroupStateMgt;
    interface IOPCSyncIO;
    interface IOPCAsyncIO;
        interface IOPCDataCallback;
    interface IOPCItemMgt;
    interface IEnumOPCItemAttributes;
    interface IOPCAsyncIO2;
    interface IOPCItemProperties;
    interface IOPCItemDeadbandMgt;
    interface IOPCItemSamplingMgt;
    interface IOPCBrowse;
    interface IOPCItemIO;
    interface IOPCSyncIO2;
    interface IOPCAsyncIO3;
    interface IOPCGroupStateMgt2;
};
```