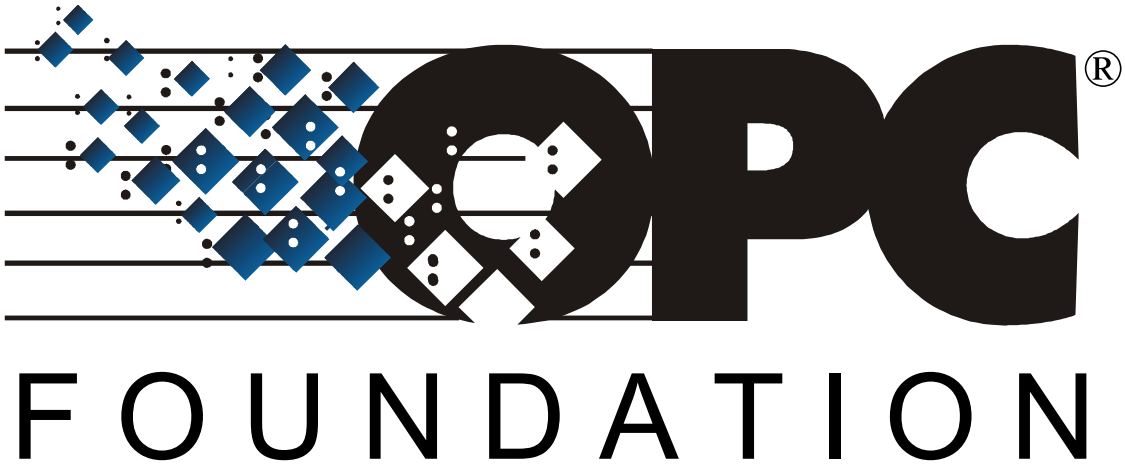

OPC XML-DA Specification

Version 1.01

Status: Released

December 18, 2004

Specification Type	Industry Standard Specification		
Title:	**OPC XML-DA Specification**	Date:	**December 18, 2004**
Version:	1.01	Soft Source:	MS-Word OPC XMLDA 1.01 Specification.doc
Author:	OPC Foundation	Status:	**Released**

Synopsis:

This document is targeted at developers and is the specification of the services to be exposed by XML-DA servers and used by XML-DA clients.. 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:

Minimally, any operating system that is capable of parsing XML messages and can support 64-bit integers and 64-bit floating point types. Practically, the runtime environment should be a 32-bit operating system with a Web server, an XML parser and a SOAP API of some sort.

NON-EXCLUSIVE LICENSE AGREEMENT

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

The OPC specifications define standard interfaces, objects, methods, and properties for servers of real-time information like distributed process systems, programmable logic controllers, smart field devices and analyzers. The OPC Foundation distributes specifications, prototype software examples and related documentation (collectively, the "OPC Materials") to its members in order to facilitate the development of OPC compliant applications.

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.

All OPC Materials, unless explicitly designated otherwise, are only available to currently registered members of the OPC Foundation (an "Active Member"). If the User is not an employee or agent of an Active Member then the User is prohibited from using the OPC Materials and all copies (in all formats) of such materials in User’s possession must either be destroyed or returned to the OPC Foundation.

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. User may not distribute OPC Materials outside of the Active Member organization to which User belongs unless the OPC Foundation has explicitly designated the OPC Material for public use.

All copies of the OPC Materials made and/or distributed by User must include all copyright and other proprietary rights notices included 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.

The following additional restrictions apply to all OPC Materials that are software source code, libraries or executables:

1. User is requested to acknowledge the use of the OPC Materials and provide a link to the OPC Foundation home page www.opcfoundation.org from the About box of the User's or Active Member's application(s).

2. User may include the source code, modified source code, built binaries or modified built binaries within User's own applications for either personal or commercial use except for:

   a) The OPC Foundation software source code or binaries cannot be sold as is, either individually or together.

   b) The OPC Foundation software source code or binaries cannot be modified and then sold as a library component, either individually or together.

In other words, User may use OPC Foundation software to enhance the User's applications and to ensure compliance with the various OPC specifications. User is prohibited from gaining commercially from the OPC software itself.

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 the relevant OPC specifications. 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, 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ 85260-1830, USA.

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 1.01 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:

- Added link to OPC Forum for OPC XML-DA errata (1.6).
- Added clarifications for data type conversions (2.7.4).
- Modified rule for the availability of the Element DiagnosticInfo in ItemValue if no DiagnosticInfo is available (3.1.5).
- Added comments to ReturnItemName and ReturnItemPath in the RequestOptions (3.1.6).
- Added E_BADTYPE to the summary list of OPCError (3.1.9).
- Added comment to MaxAge description for Read (3.3.1).
- Added requirement that servers must not allow conversions from strings during writes (3.4.1).
- Removed error code E_NOSUBSCRIPTION from the list of possible errors for SubscriptionPolledRefreshResponse (3.6.2).
- Added error E_NOSUBSCRIPTION to the list of possible errors for SubscriptionCancelResponse (3.7.2).
- Added comment to SubscriptionPolledRefresh (3.6.1) to clarify the behaviour for the first SubscriptionPolledRefresh call after Subscribe.
- Clarified the use of the value element in a response to a write only item (3.3.2) (3.4.1) (3.5.2).
- Added suggestion that clients always specify a non-zero SubscriptionPingRate (3.5.1).

Table of Contents

1. INTRODUCTION 9

1.1 BACKGROUND 9
1.2 PURPOSE 9
1.3 RELATIONSHIP TO OTHER OPC SPECIFICATIONS 9
1.4 DELIVERABLES 9
1.5 PREREQUISITES 9
1.6 XML-DA ERRATA 10

2. FUNDAMENTAL CONCEPTS 11

2.1 SOAP 11
2.2 NAME SPACE 11
2.3 OPC-XML-DA SERVER DETECTION 11
2.4 LOCALE IDS 11
2.5 SUBSCRIPTION ARCHITECTURE 12
2.5.1 Basic Polled Refresh Approach 13
2.5.2 Advanced Polled Refresh Approach 14
2.5.3 Data Management Optimization 16
2.5.4 Buffered Data 18
2.5.5 Timestamps 20
2.6 FAULTS AND RESULT CODES 23
2.7 DATA TYPES FOR ITEM VALUES 25
2.7.1 Simple 25
2.7.2 Enumeration 26
2.7.3 Array 26
2.7.4 Data Range and Precision 27
2.7.5 Data Types and Localization 28
2.7.6 Data Type Conversions 29
2.8 SECURITY 29
2.9 COMPLIANCE 30

3. OPC XML-DA SCHEMA REFERENCE 31

3.1 BASE SCHEMAS 31
3.1.1 Hierarchical Parameters 31
3.1.2 Null Parameters 32
3.1.3 RequestList 33
3.1.4 RequestItem 33
3.1.5 ItemValue 34
3.1.6 RequestOptions 38
3.1.7 ServerState 40
3.1.8 ReplyBase 40
3.1.9 OPCError 41
3.1.10 ItemProperty 42
3.2 GETSTATUS 47
3.2.1 GetStatus 47
3.2.2 GetStatusResponse 48
3.3 READ 50
3.3.1 Read 50
3.3.2 ReadResponse 53
3.4 WRITE 56
3.4.1 Write 56
3.4.2 WriteResponse 59

3.5 SUBSCRIBE 61
3.5.1 Subscribe 61
3.5.2 SubscribeResponse 64
3.6 SUBSCRIPTIONPOLLEDREFRESH 67
3.6.1 SubscriptionPolledRefresh 67
3.6.2 SubscriptionPolledRefreshResponse 69
3.7 SUBSCRIPTIONCANCEL 72
3.7.1 SubscriptionCancel 72
3.7.2 SubscriptionCancelResponse 73
3.8 BROWSE 74
3.8.1 Browse 74
3.8.2 BrowseResponse 77
3.9 GETPROPERTIES 81
3.9.1 GetProperties 81
3.9.2 GetPropertiesResponse 83

4. TRANSPORTS 85

5. APPENDIX A - PATENT ISSUES 87

6. APPENDIX B - FORMAL SCHEMAS (WSDL) 89

1. Introduction

1.1 Background

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

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

1.2 Purpose

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

The goal for this document is to provide:

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

1.3 Relationship to Other OPC Specifications

This specification is analogous to an Automation Specification, which is a companion document to a Custom Specification. The Custom Specification provides the base concepts and capabilities. OPC then specifies how these base concepts and capabilities are exposed as Automation interfaces in the Automation Specification. The XML-DA Specification is a companion to the OPC Data Access 3.0 Specification.

XML-DA servers may stand alone, or may be developed to wrap COM based 3.0, and even 2.0x servers.

1.4 Deliverables

This document covers the analysis and design for an XML based interface to exchange Data Access 2.0x and 3.0 type data. This document will also minimally specify transport specific interoperability requirements.

Sample code, or reference implementations will supplement this document to help vendors understand and leverage this technology.

1.5 Prerequisites

Readers are expected to be familiar with the applicable OPC Specifications.

These following document titles and others can be found at the following Web address:
http://www.opcfoundation.org/

OPC Data Access Custom Interface Standard 2.05

OPC Data Access Custom Interface Standard 3.0

OPC Common Definitions and Interfaces 1.0

OPC Security Custom Interface Standard

Readers should be familiar with XML.

Information regarding XML and various links to related sites, white papers, specs, etc, can be found at the following Web address: http://www.w3.org/XML/

1.6 XML-DA Errata

Any errors, omissions or corrections to this OPC XML-DA Specification will be posted to the OPCXML-DA Errata topic of the OPC foundation forums:

http://www.opcfoundation.org/forum/viewtopic.php?t=1113

2. Fundamental Concepts

2.1 SOAP

OPC XML-DA is being developed in a manner that leverages concepts from Simple Object Access Protocol (SOAP) 1.1 found at http://www.w3.org/TR/SOAP/.

OPC XML-DA is modeled in a manner which allows its structured information to be delivered in a SOAP message as a SOAP Body.

2.2 Name Space

The OPC XML-DA Specification is consistent with SOAP Notational Conventions. The following is borrowed from the SOAP Spec for the reader's convenience:

> "The namespace prefixes "SOAP-ENV" and "SOAP-ENC" used in this document are associated with the SOAP namespaces "http://schemas.xmlsoap.org/soap/envelope/" and "http://schemas.xmlsoap.org/soap/encoding/" respectively.
>
> Throughout this document the namespace prefix "xsi" is associated with the URI "http://www.w3.org/2001/XMLSchema-instance" which is defined in the XML Schemas specification [11].
>
> Similarly, the namespace prefix "xsd" is associated with the URI "http://www.w3.org/2001/XMLSchema" which is defined in [10]. The namespace prefix "tns" is used to indicate whatever is the target namespace of the current document. All other namespace prefixes are samples only."

OPC XML-DA addresses OPC Items via ItemPath, and ItemName. These concepts are described later in the document.

2.3 OPC-XML-DA Server Detection

Currently, OPC has not defined a mechanism to detect nodes with OPC-XML-DA Servers or to detect OPC-XML-DA Servers on a specific node. The Universal Description, Discovery and Integration (UDDI) protocol (see http://www.uddi.org) is a widely used standard for web services and it will be the likely basis for any future OPC specification for web service discovery. Until then, an OPC-XML-DA client needs to know the URL of any OPC-XML-DA server it wants to use.

Note that web service implementations never need to know their URLs since the person deploying and maintaining the web service on a particular machine always assigns them. In addition, these URLs (as defined by RC 1788) allow for an optional port number, as a result, OPC-XML-DA web services are not required to use the standard HTTP Port 80 (provided the web server used supports configurable port numbers).

2.4 Locale IDs

Some data in the response of a server is subject to localization. These are:

- Verbose Error information (see the Text element in OPCError)
- Values of type 'string' (this is completely server-specific)

Unlike previous OPC data access specifications, the OPC XML-DA specification describes data exchange in an environment that assumes no persistent connection between the client and the server. Although the server may maintain some state for some specific services, locale information needs to be requested in each call ("LocaleID" - see RequestOptions).

In XML-DA the LocaleID is specified as a string with the following format:

```
<language>[-<country>]
```

Where <language> is the two letter ISO 639 code for a language and <country> is the two-letter ISO 3166 code for the country. This format is a subset of the format specified by RFC 3066. The following table lists several sample Windows LCIDs and the corresponding XML-DA LocaleID.

Locale	Windows LCID	XML-DA LocaleID
Neutral	0x0000	No equivalent.
Invariant	0x007F	Empty string ("").
System Default	0x0400	No equivalent.
User Default	0x0800	No equivalent.
English	0x0009	en
English (US)	0x0409	en-US
German	0x0007	de
German (Germany)	0x0407	de-DE
German (Austrian)	0x0C07	de-AT

The invariant locale is neither language nor country specific. It is a third type of locale that is culture-insensitive. It is associated with the English language but not with a country or region. It primarily is used internally by applications for locale independent operations such as system calls or file serialization. The invariant locale is *not* the default locale since the default locale for a system or a user always specifies a specific language and country. XML-DA clients that wish to use the default locale for the server must not specify any value for the LocaleID. Servers must return its default locale as the RevisedLocaleID in this case. Clients that specify the invariant locale (i.e. an empty string) are requesting that results be returned in a culture insensitive format. A server may or may not choose to support the invariant locale and should respond to the client accordingly.

The neutral, system default and user default LCIDs have no equivalent in XML-DA. Clients are required to explicitly request a locale or allow the server to choose the locale by not specifying any value for the LocaleID.

In the event that the server does not support the requested locale, it is expected that the server select the best matching locale by ignoring the country component of the locale. If the server does not support the country neutral locale for a specific language then the server should select its default locale. In all cases the server should set the RevisedLocaleID in the ReplyBase object to indicate what locale was actually used if it differs from the requested locale.

The clients may alternatively determine the server's full set of supported LocaleIDs by querying the server via a *GetStatus*. This gives the client the option to either select a supported LocaleID or use data returned by the server based on the RevisedLocaleID. In this case the client is still free to make requests but with the understanding that localized strings will be returned in the language specified by the value of the RevisedLocaleID.

2.5 Subscription architecture

The design of the OPC-XML-DA subscription employs a "polled-pull" style of subscription. The client thus enters into a loose contract with the server. The client application initiates the subscription

and agrees to issue periodic refresh requests. In order to better simulate the callback capabilities of the original COM-based OPC DA design a "Controlled Response" mechanism is also included in the design. This mechanism can be used to reduce the latency time of reporting a value change to a client and minimize the number of round trips between the client and server.

XML-DA supports the following subscription based services: *Subscribe*, *SubscriptionPolledRefresh,* and *SubscriptionCancel. Subscribe* is used to initiate a subscription contract with a server. *SubscriptionPolledRefresh* is called periodically to acquire the latest item value changes. *SubscriptionCancel* is used to terminate the subscription contract with the server.

2.5.1 Basic Polled Refresh Approach

The basic polled subscription interaction between client and server is outlined below in Figure 2.1. The client initiates the subscription and the server returns a subscription handle in response to the request. The server will also return any initial values (value, quality, and timestamp) that are readily available if the ReturnValuesOnReply option is set to true. The client then enters a periodic polling cycle and continues to poll periodically by issuing subscription refresh requests passing in the subscription handle each time. The server responds immediately returning all value and/or quality changes since the previous poll. This process continues until the client no longer wishes to maintain the subscription at which point it issues a subscription cancel request to the server. The server cleans up allocated resources and performs any other actions necessary to end the subscription contract it held with the client.

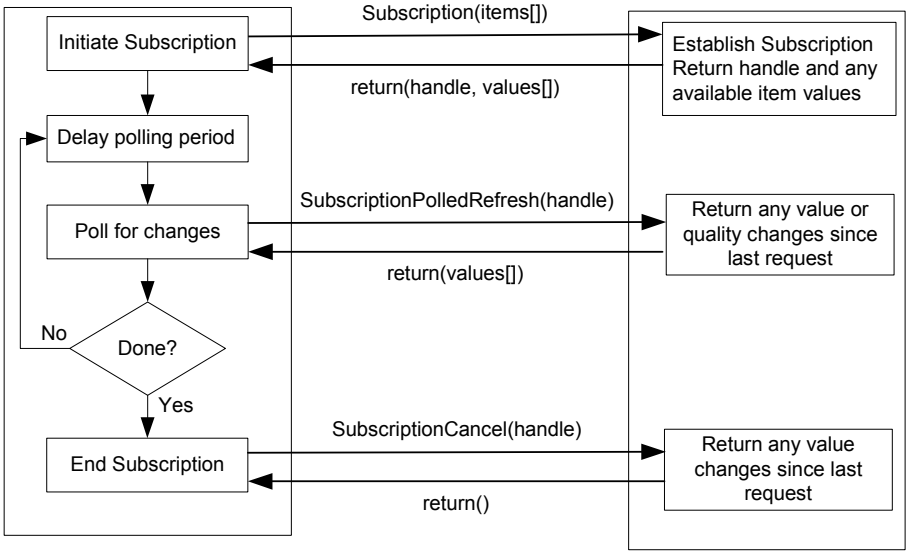

Figure 2.1 Basic polled subscription interaction diagram

The client application must be prepared to handle error conditions from the server. The type of error returned will dictate what action needs to be taken. Figure 2.2 below illustrates the logic flow of a client when dealing with critical subscription errors. If the subscription operation fails then the client should retry if the error is such that the condition could be cleared over time (i.e., communications errors). It is expected that a well-designed client will interpret the error codes and take appropriate actions. This includes handling timeouts, both on the initial *Subscription* call, and the subsequent *SubscriptionPolledRefresh* calls.

It is important to note that in the case of fatal errors or timeouts on *SubscriptionPolledRefresh* calls the client needs to determine the appropriate error response; reissue the same or a revised

SubscriptionPolledRefresh call, or cleanup the existing subscription and start over by creating a new subscription, and then cancel the old subscription. The servers may take steps to optimize this subscription "re-creation" process.

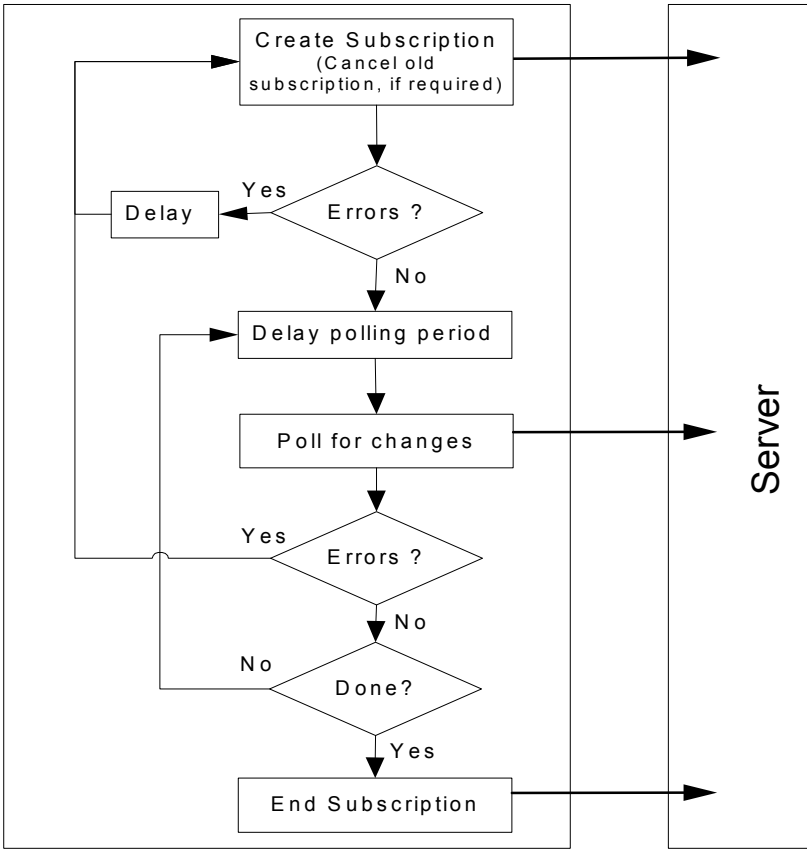

Figure 2.2 Error handling of polled subscriptions

2.5.2 Advanced Polled Refresh Approach

An advanced client application may elect to use the more sophisticated polling approach in order to optimize the behavior of the server in its response to client requests for data and to more closely simulate the traditional asynchronous callback provided with the OPC COM based interface. This approach makes use of two Subscription Refresh parameters.

- Holdtime - instructs the server to hold off returning from the *SubscriptionPolledRefresh* call until the specified absolute server time is reached.
- Waittime -instructs the server to wait the specified duration (number of milliseconds) after the Holdtime is reached before returning if there are no changes to report. A change in one of the subscribed items, during this wait period, will result in the server returning immediately rather than completing the wait time.

Using this approach the client application does not perform a delayed poll but rather delegates the waiting to the server. The client may issue a *SubscriptionPolledRefresh* call as soon as it has processed

the returned changes provided by the previous call. By shifting the polling delays to the server side the worst-case latency time to deliver a change to the client is minimized. Figure 2.3 and 2.4 below illustrates the relative effects of the two parameters.

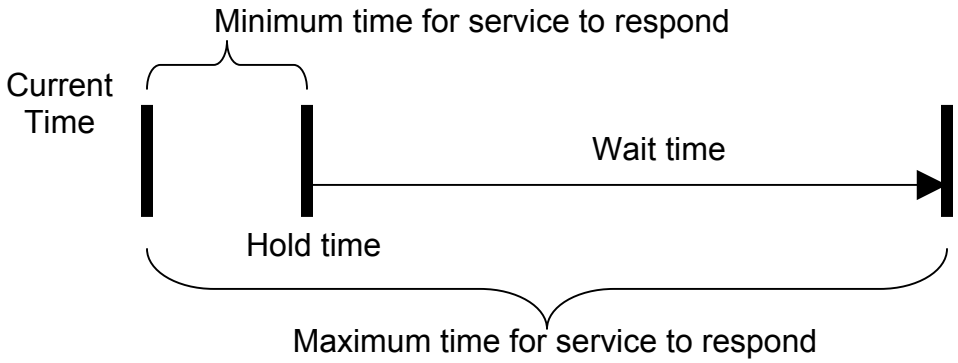

Figure 2.3 Minimum and Maximum response times

The client application sets the Holdtime parameter based on the maximum update rate needed (as specified as an absolute time value). This parameter is analogous to the 'maximum allowable latency' - the smaller this delay the smaller the latency will be in reporting changes. That is to say, if there are changes to report they will be returned only after Holdtime has passed. The side effect of reducing the Holdtime is the number of client-to-server round trips will increase resulting in more network traffic and more client and server processing. The client application sets the Waittime parameter to a value that balances the need for a reasonably short time period to quickly detect server failures with the need for a reasonably long time period to minimize client to server round trips. If there are no changes to report, the server will not respond to the client refresh request until the sum of Holdtime and Waittime has elapsed. If at the end of the Holdtime and Waittime and there are still no values which have changed, the server will still respond with a response although the response will not have any Item Values.

The client application should manage a timeout period that is greater than the sum of Holdtime, Waittime and network round trip time. The client should always be aware that specifying values that are too large may result in other transport based errors.

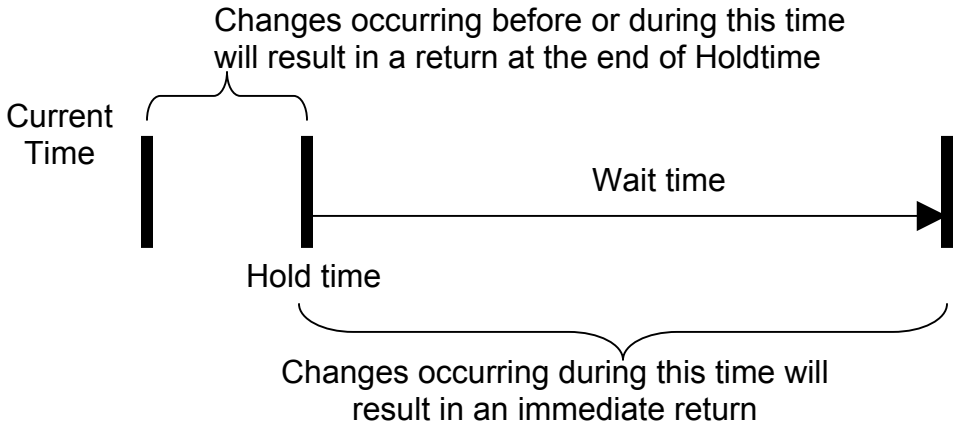

Figure 2.4 Response timing

A subscription contract will be considered abandoned if the client application fails to issue *SubscriptionPolledRefresh* calls. The server is free to terminate the subscription any time after the period specified by SubscriptionPingRate has elapsed since the last response to a refresh service call.

A server is free to return from the *SubscriptionPolledRefresh* call sooner than the requested "Holdtime" in the case an error has occurred that needs to be reported back to the client.

A client application electing to make use of the simulated callback polling service outlined here must take into account the extended time period during which it will be waiting on the server to respond. Either asynchronous subscription refresh calls should be made or a dedicated refresh polling thread should be used.

2.5.3 Data Management Optimization

By subscribing to data items the client identifies to the server that it is interested in those specific data items. By entering into this loose contract with the server via the *Subscribe* service the client may provide the server with suggestions on the time and data change characteristics of the data of interest. These suggestions are meant to further optimize the relationship between client and server and the server's ability to manage its data. The suggestions are provided via the attributes:

- RequestedSamplingRate
- EnableBuffering
- DeadBand

These attributes may or may not be useful to the server depending on how the server maintains the data that is within its domain.

A typical scenario is a server which front ends some device, and is only able to periodically poll the device to update the server's data cache. The server may be limited by the device as to the periodicity of the data polling: the fastest practical rate, the specific periods, and the total bandwidth of polling requests. The server may arbitrarily balance these constraints, or it might be responsive to client suggestions.

The client may specify a RequestedSamplingRate at the List level (described later in document) which indicates the fastest rate at which the server should poll the underlying device for data changes. Polling at rates faster than this rate is acceptable, but not necessary to meet the needs of the client. The client may also specify 0 for RequestedSamplingRate which indicates that the server should use the fastest practical rate.

How the server deals with the sampling rate and how often it actually polls the hardware internally is a server implementation detail.

The client may also specify a RequestedSamplingRate at the Item level (described later in document). By specifying a RequestedSamplingRate different than List level RequestedSamplingRate the client is requesting that the server override the List level SamplingRate for purposes as the suggestion for the server's underlying device poll rate.

In many cases the RequestedSamplingRate(s) are giving the Server an indication of the fastest that a client will be submitting *SubscriptionPolledRefresh* requests. The RevisedSamplingRate is in return giving the client an indication of the fastest that a client should be submitting *SubscriptionPolledRefresh* requests.

By specifying EnableBuffering = True, the server will save all value changes detected at the specified rate in a buffer for return to the client at the next *SubscriptionPolledRefresh* request.

The following table indicates the expected behavior for a server that front ends a device, which must be polled for data and is able to respond to the client's requests. The attributes SamplingRate (List), and SamplingRate (Item) in the table represent the values requested by the client, and may not be exactly equivalent to those that are returned by the server as "Revised" values. The client is describing the desired behavior via the requested attributes. The server will attempt to honor the client's desired behavior.

SamplingRate (List)	SamplingRate (Item)	Values	Expected behavior
Missing	Missing	LCV	Server will attempt to poll underlying device at some server default rate and return the most accurate data available.
0 (Fastest)	Missing	LCV	Server will attempt to poll underlying device at fastest practical rate and return the most accurate data available.
> 0	Missing	LCV	Server will attempt to poll underlying device at the List level SamplingRate and return the most accurate data available.
N/A	0 (Fastest)	LCV	Server will attempt to poll underlying device at fastest practical rate and return the most accurate data available.
N/A	> 0	LCV	Server will attempt to poll underlying device at the Item level SamplingRate and return the most accurate data available.

Note:

LCV=Latest Changed Value
N/A=Not Applicable

As an example: RequestedSamplingRate=500 msec. The server can only support 1 second update rates, so it returns RevisedSamplingRate=1 second.

In all cases above, if EnableBuffering is True, then the server will poll the underlying device at the SamplingRate and return to the client all (or as many as resources allow) changed values (i.e., Latest Changed Value and any buffered values) from the last *SubscriptionPolledRefresh* request. Although the server may be sampling at a faster rate than the SamplingRate to support other clients, the client should only expect values at the negotiated SamplingRate.

The following table uses some pseudo VB code to describe the behavior of the server in responding to "Requested" SamplingRate at the List (RqtSR-List) and Item (RqtSR-Item) level for polling type data.

If RqtSR-List Missing and RqtSR-Item Missing Then
RevisedSamplingRate-List = Minimum (Fastest Supported Item Sampling Rates) or Default
RevisedSamplingRate-Item = Minimum (Fastest Supported Item Sampling Rate) or Default
EndIf

If RqtSR-List > Minimum (Fastest Supported Item Sampling Rates) Then
RevisedSamplingRate-List = RqtSR-List
Else
RevisedSamplingRate-List = Minimum (Fastest Supported Item Sampling Rates)
EndIf

If RqtSR-Item Missing and RqtSR-List NOT Missing Then
RqtSR-Item = RqtSR-List
EndIf
If RqtSR-Item > Fastest Supported Item Sampling Rate Then
RevisedSamplingRate- Item = RqtSR- Item
Else
RevisedSamplingRate-Item = Fastest Supported Item Sampling Rate
EndIf

The server may be supporting data which is collected based on a sampling model, or generated based on an exception based model. The Fastest Supported Item Sampling Rate may be equal to 0 which indicates that the data item is exception based versus being sampled at some period. If the client passes in a 0 for Requested SamplingRate, then the server may respond with either 0 or some value which represents the fastest practical rate that would be of interest to the client. The client may use the RevisedSamplingRate values as a hint for how often to initiate a *SubscriptionPolledRefresh* request. Not all value changes are guaranteed to be returned if the client passes in Requested SamplingRate > 0, and the data is exception based.

DeadBand specifies the percentage of full engineering unit range of an item's value that must change prior to the value being of interest to the client. "Uninteresting" value changes are those which are less than the Deadband and are not maintained or saved by the server for return to the client in a *SubscriptionPolledRefresh* as described below.

2.5.4 Buffered Data

As mentioned above, if EnableBuffering = True the server maintains a buffer of the changed values detected in between *SubscriptionPolledRefresh* requests in addition to the cached value for an item. The server may then return more than 1 value to a client for a buffered item in a *SubscriptionPolledRefreshResponse*.

When buffering is OFF the server must deliver at most 1 value per item per *SubscriptionPolledRefreshResponse* and that will be the latest value obtained (LCV). Where buffering is ON the server may return the maximum number of values as determined by the associated sampling

rate. The expected behavior for servers is that they deliver values in which the interval between timestamps is as close to the SamplingRate as possible and where any missing values are the result of deadband logic. The server may deliver fewer values than dictated by the sampling rate based on EnableBuffering, Deadband and implementation constraints. The server will neither buffer nor deliver values for a particular item if there have not been any changes related to that item in between the *SubscriptionPolledRefresh* requests, except and only if ReturnAllItems is True.

Because the server may need to buffer an unknown amount of data, the server is allowed to constrain the buffer to a fixed maximum amount of data. If the server determines that its maximum buffer capacity has been reached, then it will push out the older data, keeping the newest data in the buffer. The server is expected to maintain at least the most current value (LCV) for each item that it is tracking, i.e., its cache value.

The following is an example of a server receiving changed values for Items 1-4, and how it would keep these values in its data cache, and its data buffer. The notation used for the values are:

Item, Timestamp, (Value index), Buffer/Cache.

Item 1, 00:01, (1), Buffer

Item 1, 00:02, (2), Buffer

Item 4, 00:01, (3), Cache

Item 2, 00:01, (4), Buffer

Item 1, 00:03, (5), Buffer

Item 2, 00:02, (6), Buffer

Item 1, 00:04, (7), Cache

Item 2, 00:03, (8), Buffer

Item 2, 00:04, (9), Cache

Item 3, 00:03, (10), Cache

If the buffer can only hold 6, and the following items come in

Item 3, 00:04, (11)

Item 3, 00:05, (12)

Item 3, 00:06, (13)

The Server would flush (1), (2), and (4). The Server would maintain (3) because it is the latest changed (cache) value for Item 4.

The server is required to return values for any particular item in chronological order. Values from different items may be interspersed but values for any specific item must be ordered by time for that item. There is no requirement for values from different items to be in any order. The values which are returned will always return at least the current value for a changed item. A sequence of values which would meet these requirements are that all of the buffered items are returned followed by the most current items which have changed in the period since the last polled refresh. The example presented could return:

Item 1, 00:03, (5), Buffer

Item 2, 00:02, (6), Buffer

Item 2, 00:03, (8), Buffer

Item 3, 00:03, (10), Buffer

Item 3, 00:04, (11), Buffer

Item 3, 00:05, (12), Buffer

Item 1, 00:04, (7), Cache

Item 2, 00:04, (9), Cache

Item 3, 00:06, (13), Cache

Item 4, 00:01, (3), Cache

Buffering of data may result in unexpected behavior when using a deadband limit and the server encounters a resource limitation on the number of values that can be maintained. It is realistically possible that multiple samples of a value exceed the item's deadband limit, and the server thus buffers one or more of these samples. It is also then possible that all of the samples that were buffered are flushed out due to memory limitations. The current value could be either identical to the previous value sent to the client or within the deadband limit. The client will thus get the current value which does not exceed the deadband limit of the previous value it received, but will not get the transient buffered values.

Refer to the OPC DA Custom Specification for additional details on this topic.

2.5.5 Timestamps

The servers will provide the most accurate timestamp to associate with a value(s). 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.

If the Item Value is an array, then there is a single Timestamp which will be associated with all array elements. The server is responsible for determining/returning the most accurate timestamp.

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.

For *SubscriptionPolledRefresh* requests the changing exception based data will exhibit a Timestamp which will correspond to the time that the value changed, and not likely correspond with any requested sampling rate period. The caveat for this behavior is when the client has specified ReturnAllItems = True. For those items which the value has not changed, the server will include the last exception generated value, but use the current Timestamp. This behavior is identical to that expected if a Read had been done for those non changed items at that time.

The following tables present examples of the associated Timestamp for sampled, and exception based values. The behavior of a Read is included as a comparison.

The following table depicts the values for 2 items over time: t0 to t10.

- F101 is a sampled data item (SamplingRate = 2 seconds)
- B101 is an exception based data item

Time	F101	B101
0	2	1
1		
2	2	
3		3
4	5	
5		6
6	10	
7		9
8	10	
9		10
10	11	5

Subscribe done at t0 with ReturnValuesOnReply set

Sampling Rate at 2 seconds for F101, 0 for B101

Reads or Polled Refreshes at 2 sec

Time	Read (F101)	Read (B101)	Subscribe / Polled Refresh (F101)	Subscribe / Polled Refresh (B101)	Subscribe / Polled Refresh (BE) (F101)	Subscribe / Polled Refresh (BE) (B101)
0	2, t0	1, t0	2, t0	1, t0	2, t0	1, t0
2	2, t2	1, t2				
4	5, t4	3, t4	5, t4	3, **t3**	5, t4	3, **t3**
6	10, t6	6, t6	10, t6	6, **t5**	10, t6	6, **t5**
8	10, t8	9, t8		9, **t7**		9, **t7**
10	11, t10	5, t10	11, t10	5, t10	11, t10	10, **t9*** 5, t10

Note: BE = Buffering Enabled

*If Sampling Rate for B101=2, then this value would not be returned

The following scenarios have ReturnAllItems set.

Time	Read (F101)	Read (B101)	Subscribe / Polled Refresh (F101)	Subscribe / Polled Refresh (B101)	Subscribe / Polled Refresh (BE) (F101)	Subscribe / Polled Refresh (BE) (B101)
0	2, t0	1, t0	2, t0	1, t0	2, t0	1, t0
2	2, t2	1, t2	2, t2	1, **t2**	2, t2	1, **t2**
4	5, t4	3, t4	5, t4	3, **t3**	5, t4	3, **t3**
6	10, t6	6, t6	10, t6	6, **t5**	10, t6	6, **t5**
8	10, t8	9, t8	10, t8	9, **t7**	10, t8	9, **t7**
10	11, t10	5, t10	11, t10	5, t10	11, t10	10, **t9*** 5, t10

Note: BE = Buffering Enabled

*If Sampling Rate for B101=2, then this value would not be returned

2.6 Faults and Result Codes

The OPC-XML-DA specification describes a set of behaviors that web services as exposed by the server must implement and that web based client applications use. When abnormal conditions arise, the services must be able to communicate exception information to applications on a per-operation and a per-item basis.

When a given operation fails entirely, the web service must return a SOAP fault, as defined by the SOAP specification. This would occur, for instance, in response to a badly formatted request (e.g., missing required attributes). No other results (e.g., item values) are returned. E_FAIL is for cases where execution of the request fails due to unknown reasons, and the server is in a state that should support that request. The following XML document is an example of a response returned to the client containing a SOAP fault:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
  <soap:Body>
    <soap:Fault>
      <faultcode xmlns:q0=http://opcfoundation.org/webservices/XMLDA/1.0/>
         q0:E_SERVERSTATE
      </faultcode>
      <faultstring>
        The operation could not complete due to an abnormal server state.
      </faultstring>
      <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

In this example the SOAP fault code is set to one of the error codes defined in this specification. An XML-DA server must ensure that, whenever possible, the SOAP fault code is set properly and must not rely on the default behavior for their development environment. The XML-DA server should also return the error text for the LocaleID specified in the request.

The server state is made available via the ReplyBase. If the server state is “failed”, then the server should reject all calls except *GetStatus* with a SOAP fault. If the server state is "suspended", or “noConfig” the server should reject any data related calls (*Read*, *Write*, and *Subscribe*) with a SOAP fault.

However, individual items may encounter critical or non-critical exceptions even when an operation as a whole succeeds. OPC-XML-DA employs a mechanism similar to SOAP faults to express these item-level errors.

Each item value in the result may have a ResultID attribute. The value of this attribute is an XML qualified name. Similar to the HRESULT in the COM-based OPC Interfaces it may specify a critical error or a non-critical exception (a so-called success code). The qualified names for critical errors have the prefix “E_” (e.g., E_OUTOFMEMORY); the qualified names for non-critical exceptions have a prefix of “S_” (like S_UNSUPPORTEDRATE).

In case of a critical error the returned value may not be useful. For non-critical exceptions the returned value is useful, although the client may need to react to an abnormal condition. If the attribute is not present, then the item has encountered no abnormal conditions and the value is useful.

While the result codes are very useful for applications to recognize and deal with error conditions, they are not intended to be "human-readable". Client applications may request localized, human-readable text for result codes from the web service with each transaction. The text appears in a series of non-

duplicated Errors elements in the body of the response. This allows the response to contain multiple instances of the same Error ID that map to a single verbose OPC Error. See the example below which has the same error associated with multiple items, and each of the ResultIDs points to one verbose representation of the error. The server should use the LocaleID in determining the specific verbose description to be returned.

```
<soap:Body>
  <ReadResponse xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <ReadResult RcvTime="2003-05-26T15:55:14.0250000-07:00"
ReplyTime="2003-05-26T15:55:14.1250000-07:00"
ServerState="running" />
    <RItemList>
      <Items ItemName="Simple Types/UInt1" ResultID="E_UNKNOWNITEMNAME">
        <Quality QualityField="bad" />
      </Items>
      <Items ItemName="Simple Types/UInt2" ResultID="E_UNKNOWNITEMNAME">
        <Quality QualityField="bad" />
      </Items>
    </RItemList>
    <Errors ID="E_UNKNOWNITEMNAME">
      <Text>The item name is no longer available in the server address
space.</Text>
    </Errors>
  </ReadResponse>
</soap:Body>
```

OPC-XML-DA defines a series of standard result codes (Success or Error) that have specific applications in data access operations. These codes are always qualified with the namespace http://opcfoundation.org/webservices/XMLDA/1.0/.

Vendors may choose to create their own custom result codes, but these must be qualified with a vendor-specific namespace (i.e. "http://company.com/etc"). Please refer to the W3C XML 1.0 specification for more information about namespaces and qualified names. Note that vendors are still required to use the standard codes where specifically mentioned. Vendor result codes also have to use the convention that success codes begin with “S_” and error codes begin with “E_”.

Success codes and error codes for each service are listed as part of the response messages in the sections that describe those services.

2.7 Data Types for Item Values

2.7.1 Simple

The supported data types are a subset of those defined in "XML Schema Part 2: Datatypes" http://www.w3.org/TR/xmlschema-2/, and are consistent with those provided by the OPC Data Access Custom Interface.

The data types supported by OPC XML-DA Servers are presented in the table below.

Data Type	Description	Variant Data Type
string	A sequence of UNICODE characters.	VT_BSTR
boolean	A binary logic value (true or false).	VT_BOOL
float	An IEEE single-precision 32-bit floating point value.	VT_R4
double	An IEEE double-precision 64-bit floating point value.	VT_R8
decimal	A fixed-point decimal value with arbitrary precision.	VT_CY
long	A 64-bit signed integer value.	VT_I8
int	A 32-bit signed integer value.	VT_I4
short	A 16-bit signed integer value.	VT_I2
byte	An 8-bit **signed** integer value. Note this differs from the definition of 'byte' used in most programming languages.	VT_I1
unsignedLong	A 64-bit unsigned integer value.	VT_UI8
unsignedInt	A 32-bit unsigned integer value.	VT_UI4
unsignedShort	A 16-bit unsigned integer value.	VT_UI2
unsignedByte	An 8-bit unsigned integer value.	VT_UI1
base64Binary	A sequence of 8-bit values represented in XML with Base-64 Encoding.	VT_UI1 | VT_ARRAY
dateTime	A specific instance in time.	VT_DATE
time	An instant of time that recurs every day. See: W3C "XML Schema Part 2: Datatypes"	VT_DATE
date	A Gregorian calendar date. See: W3C "XML Schema Part 2: Datatypes"	VT_DATE
duration	A duration of time as specified by Gregorian year, month, day, hour, minute, and second components. See: W3C "XML Schema Part 2: Datatypes"	VT_BSTR
QName	An XML qualified name comprising of a name and a namespace.	No equivalent

	The name must be a valid XML element name and the namespace must be a valid URI. QNames are equal only if the name and the namespace are equal.	
anyType	A value that has its type explicitly specified in the XML document with the 'type' attribute. This type only has relevance when used as an element in an array (See 2.7.3).	VT_VARIANT

Note: The type attribute associated with the Item's value element identifies the type of the value.

time, **date**, and **duration** are not supported fully by the .NET tools. **time**, and **date** are transmitted as **dateTime** while **duration** is transmitted as a **string**. A **ValueTypeQualifier** attribute is included when values of this type are transmitted between client and server. The ValueTypeQualifier attribute uniquely identifies the intended value type of the value versus the type as transmitted across the wire.

Servers may support values of a type other than those specified above, but there may be compatibility issues with clients which do not understand those types.

2.7.2 Enumeration

"XML Schema Part 2: Datatypes" found at http://www.w3.org/TR/xmlschema-2/ defines enumerations, and SOAP directly adopts the defined mechanism. Enumeration as defined is a data type constraining facet which means that all data types except Boolean may have associated enumerated values. OPC recommends against these defined enumerations as item values, but instead recommends the use of the enumeration methodology as described in the OPC DA Specification.

Servers may return either the string representation or the integer representation of the enumeration value. The type of returned value will be based on the client's requested type, with the default being the string representation of the enumeration.

The OPC Enumeration methodology provides two Item Properties: euType, and euInfo to let the client be aware of whether the values are enumerated and if so, then euInfo would provide an array of strings which represents the textual representation of the elements of the enumeration. See the ItemProperty description in Section 3.1.10 for further details.

2.7.3 Array

OPC defines the following arrays for a subset of the simple types listed above:

Type Name	Element Type Name
ArrayOfByte	byte
ArrayOfShort	short
ArrayOfUnsignedShort	unsignedShort
ArrayOfInt	int
ArrayOfUnsignedInt	unsignedInt
ArrayOfLong	long

ArrayOfUnsignedLong	unsignedLong
ArrayOfFloat	float
ArrayOfDecimal	decimal
ArrayOfDouble	double
ArrayOfBoolean	boolean
ArrayOfString	string
ArrayOfDateTime	dateTime
ArrayOfAnyType	anyType

ArrayOfUnsignedByte is not supported because it is more efficient to transport these arrays as the XML type base64Binary. The XML binary type uses base64 encoding to transmit the array as a single XML element instead of one element for each byte.

ArrayOfAnyType allows each array element to be either a different simple type or another array. The following XML fragment illustrates how this type appears in an XML document.

```
<Value xsi:type="ArrayOfAnyType">
  <anyType xsi:type="xsd:byte">127</anyType>
  <anyType xsi:type="xsd:unsignedByte">255</anyType>
  <anyType xsi:type="xsd:string">Hello<>World</anyType>
  <anyType xsi:type="ArrayOfInt">
    <int>-2147483648</int>
    <int>0</int>
    <int>2147483647</int>
  </anyType>
  <anyType xsi:type="ArrayOfAnyType">
    <anyType xsi:type="xsd:byte">127</anyType>
    <anyType xsi:type="xsd:unsignedByte">255</anyType>
    <anyType xsi:type="xsd:string">Hello<>World</anyType>
    <anyType xsi:type="ArrayOfInt">
      <int>-2147483648</int>
      <int>0</int>
      <int>2147483647</int>
    </anyType>
  </anyType>
</Value>
```

2.7.4 Data Range and Precision

Most implementers of this specification will need to represent the data types required by this specification in a binary format that is appropriate for their development environment. This binary representation will always impose restrictions on the range and precision of a value that are not defined by the XML data type itself.

For example, consider a Windows based XML-DA client that wishes to write a 64-bit FILETIME representation of a dateTime value to a UNIX based XML-DA server that uses a 32-bit UNIX time_t representation for a dateTime values. In this situation, the server will not be able to accept valid XML dateTime values (such as 1601-01-01) written by the client because they exceed the capacity of its internal representation.

For this reason, this specification defines three item properties (minimum value, maximum value and value precision) that allow an XML-DA server to publish any limitations on an item value imposed by its internal representation of the item's data type. A server must implement these properties whenever

it cannot support at least the default range and precision for an XML data type defined by this specification.

Note that these properties may only be used for canonical data types that do not have an explicit range and precision defined in the XML Schema specification.

The following table summarizes the default range and precision for each data type that does not have an explicit range and precision defined by the XML Schema specification:

Data Type	Minimum Value	Maximum Value	Value Precision
decimal	0.0001	922337203685477.5807	4 (digits) [1]
dateTime	1900-01-01 00:00:00.000	2100-12-31 23:59:59.999	1000000 (ns) [2]
time	00:00:00.000	23:59:59.999	1000000 (ns) [2]
date	1900-01-01	2100-12-31	Not applicable
Duration	Not applicable	Not applicable	1000000 (ns) [2]

[1] The precision for decimal types indicates the maximum number of digits after the decimal place.

[2] The precision for date/time types indicates the minimum time difference in nanoseconds.

A server that supports at least the range specified in the table does not need to implement the range properties. In this situation, a client must be prepared to accept values that are outside these ranges. In addition, a server must accept any value during a write that it returned during a read. A server may return the error E_RANGE if the client writes a value outside the default range.

A server that explicitly specifies the range properties must only accept and return values within the range. It must return the error E_RANGE if the client writes a value outside the range.

Note that the ranges for dateTime values only apply to items which have a canonical data type of dateTime. Timestamps are intended to be values ‘close’ (e.g. ±1 year) to the current time. XML-DA clients must be prepared to deal with errors if they write timestamps that are outside of this loosely defined range.

The precision property is an approximation that is intended to provide guidance to a client. A server is expected to silently round any value with more precision that it supports. This implies that a client may encounter cases where the value read back from a server differs from the value that it wrote to the server. This difference should be no more than the difference suggested by the value precision property.

Note that the precision property only applies to writes and reads using the canonical data type for an item. XML-DA clients should be prepared for round off errors whenever one data type is converted to another.

2.7.5 Data Types and Localization

An XML-DA server must support conversions from any supported scalar data type to a string.

When converting a scalar value to a string, an XML-DA server must use either the XML string representation for the data type or a localized string representation. If a server uses a localized string representation it must attempt to use the locale specified by the client in the request. If it cannot use that locale it may use another locale, however, the actual locale used should be returned in the response to the client. This implies that the same locale should be used for all values returned within a single request.

An XML-DA server may also support conversions from any supported array data type and to a string using a vendor-defined syntax.

An XML-DA client cannot make any assumption about the syntax of the string value returned from a server. More specifically, it cannot know whether the server used a localized representation or the XML Schema representation for a given string value. If a client wishes to use the value returned for any purpose other than display to the user then it should request the value in its canonical data type.

2.7.6 Data Type Conversions

An XML-DA server must support all of the Data Type conversions defined for the OPC DA Custom Specification except as noted below.

String Values

An XML-DA server must not convert string values to any other data type during a write since differences between the string representations of values in different locales may result in the incorrect value being written. During a read, the server must support conversions from strings to any scalar data type.

DateTime, Date, Time and Duration Values

An XML-DA Server must support conversions to and from strings for the dateTime, date, time and duration data types (except during writes as noted above). An XML-DA server may support vendor specific conversions to and from numeric values and these data types.

Boolean Values

An XML-DA Server must support conversions to and from boolean values for string and numeric data types (except during writes as noted above).

For conversions from numeric values to boolean values, an XML-DA server must convert a zero value to “false” and any non-zero value to “true”.

For conversions from boolean values to numeric values, an XML-DA server must convert a false value to “0” and a true value to any non-zero value.

For conversion from a boolean value to a numeric value, an XML-DA must convert a true to a non-zero value and a false to a zero value.

`Decimal` Values

An XML-DA Server must support conversions to and from decimal values for string and numeric data types (except during writes as noted above).

Even if an XML-DA server does not have any items that are decimal type, it still must be able to parse the XML message and determine if a specific decimal value can be converted to a numeric type that it does support.

`QName` Values

An XML-DA Server is not required to support any data type conversions for the QName type.

2.8 Security

The assumption that OPC XML-DA makes is, that the transport will handle security, e.g., HTTPS

The OPC specifications define interfaces that provide open access to various forms of process control information. Such information can be of great importance to the operations of an enterprise and should therefore be protected. Vendors and end-users must work together to ensure that sensitive information

is guarded against unauthorized access. Unauthorized access can include both data espionage and sabotage of critical control parameters.

In the past, many companies have simply chosen to adopt a "wide-open" security policy for DCOM OPC servers and have relied on firewalls to protect from intruders. With the advent of web service technology, process control information is no longer restricted to the confines of a LAN. Web services are frequently deployed outside the firewall, potentially exposing important information to any person connected to the Internet.

End-users (network and site administrators) are responsible for enabling and properly configuring the security features of their selected web server components (for example, enabling the SSL capabilities of Microsoft IIS). This may include restricting access to web services to authorized users.

Vendors may also provide additional mechanisms to allow finer control over the types of operations that specific users are permitted to carry out on specific items (for example, using the Microsoft .NET security classes).

It is highly recommended that, as a minimum, vendors provide a means to globally disable the Server's "write" capabilities, putting it into a "read-only" mode.

If a vendor does choose to provide custom mechanisms, then that vendor must be certain that they do not compromise existing security mechanisms already in use. Custom mechanisms must be well integrated with existing security mechanisms. For example, client authentication and identification must be based on facilities supplied by the operating system (where available), rather than vendor-specific approaches.

End-users are still responsible for configuring vendor-specific security mechanisms correctly. Vendors should provide assistance with configuration as necessary.

The OPC Foundation is not responsible for any damage relating to compromised security. Vendors and end-users must choose for themselves the security measures needed to ensure the safety of data exposed via OPC.

Please refer to OPC Security Custom Interface Standard for additional insight into security concepts.

2.9 Compliance

OPC compliance tools have been developed to validate compliance of OPC Data Access (OPC-DA) servers and OPC Alarm & Events (OPC-AE) servers. OPC compliance test suites have not been developed for the OPC clients. The OPC Foundation will develop a compliance test suite for OPC XML-DA servers to facilitate compliance to the OPC XML-DA specification. The compliance test will be available within 6 months of release of the XML-DA specification.

3. OPC XML-DA Schema Reference

This section includes a reference for the OPC XML-DA Services. The structure, parameters, and behavior of each are defined.

The types of services to be supported are:

- **Status:** *GetStatus*, *GetStatusResponse*
- **Read:** *Read*, *ReadResponse*
- **Write:** *Write*, *WriteResponse*
- **Subscription:** *Subscribe*, *SubscribeResponse*
- **Subscription Polled Refresh:** *SubscriptionPolledRefresh*, *SubscriptionPolledRefreshResponse*
- **Subscription Cancel:** *SubscriptionCancel*, *SubscriptionCancelResponse*
- **Browse:** *Browse*, *BrowseResponse*
- **Get Properties:** *GetProperties*, *GetPropertiesResponse*

The section is to be used as a quick reference for the various OPC XML-DA services. Refer to the Appendices for formal schema definitions. The pseudo schemas in this section are not intended to be valid XML Schemas.

As a general convention the WSDL excerpts include the shorthand prefixes “s0:” and “s:” which define the OPC XML-DA namespace and the XML schema namespace respectively.

All attributes are optional unless explicitly specified as required. The description of the services will describe the expected behavior for attributes which are not included.

3.1 Base Schemas

OPC XML-DA defines base schemas which are contained by the other schemas to describe the messages to be transported.

NOTE:

The WSDL fragments shown in the document are to facilitate understanding of the specification.

Implementers must use the published WSDL that accompanies this document when building web applications that comply with this specification.

3.1.1 Hierarchical Parameters

The OPC XML-DA schemas are based on a hierarchical nature of some of the information. Information (attributes) may be specified at the Request, List, or Item level. Information specified at a lower level overrides information at a higher level. Omitted lower level attributes always imply that the higher level attributes are to be used. The client may selectively override information. Not all requests support all hierarchical levels.

As an example, a client provides MaxAge at the List level, and for certain items provides an overriding value, yet the List level value will be used for the other items in the request.

3.1.2 Null Parameters

The OPC XML-DA schemas accommodate clients and servers passing null parameters. This is the basis for supporting Hierarchical Parameters, and for providing responses which are subsets of the Item List or Items in the request. Servers and clients should support null parameters by intelligently ignoring them.

As an example, ItemPath = "" is not missing - which means that an ItemPath of "" at the item level overrides the ItemPath at the list level. However, this distinction is only necessary where a missing attribute has some meaning. In most cases, a null string or an empty string has the same meaning (i.e. continuation point). The specification defines the interpretation of a missing attribute where it has a different meaning from the default value for a type. By default, null and "" have the same meaning for all string attributes.

3.1.3 RequestList

Description

RequestList is a conceptual type that includes attributes that are part of the request list types used in read, write and subscribe requests. These attributes are described separately here for convenience.

```
<s:complexType name="RequestList">
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ReqType" type="s:QName" />
</s:complexType>
```

Tag Name	Description
ItemPath	A portion of the namespace pointing to the data. The ItemPath is server specific and the client should not make any implications about its content, structure, or consistency across servers. ItemPath may or may not imply node, or server. If an XML-DA server was front ending a DA based server, then an example could be: \\NODE\OPC.DAServer.2. ItemPath is a hierarchical parameter. If **ItemPath** is Blank or missing at all levels of the hierarchy, then the ItemName is expected to be a fully qualified name.
ReqType	Specifies the client's requested type for the Item's value to be returned by the server for a *Read* request. A Blank or missing or "anyType" ReqType will indicate to the server to use the canonical data type. If the client specifies a type, and the server is unable to respond to the request, then an error will be returned in the Response. See "Data Types for Item Values" section. Also see the corresponding section in the OPC DA Custom Specification to see which conversions are supported. ReqType is a hierarchical parameter.

Comments:

As described above, the RequestList attributes are applied hierarchically to requests. Values that appear at the list level are the defaults for items that do not explicitly specify a value for the same attribute.

3.1.4 RequestItem

Description

RequestItem is a conceptual type that includes attributes that are part of the request item types used in read and subscribe requests. These attributes are described separately here for convenience.

```
<s:complexType name="RequestItem">
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ReqType" type="s:QName" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="ClientItemHandle" type="s:string" />
</s:complexType>
```

Tag Name	Description
ItemPath	Same as attribute described in Section 3.1.3.
ReqType	Same as attribute described in Section 3.1.3.
ItemName	Identifier of the Data. It is free format (as in the OPC COM server). Required attribute.
ClientItemHandle	A string that can be passed by the client and be returned along with the data. If the Client includes the ClientItemHandle attribute, then Server must return it to the Client.

3.1.5 ItemValue

Description

ItemValue is the container of information that transports Item Values.

```
<s:complexType name="ItemValue">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="DiagnosticInfo"
type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Value" />
    <s:element minOccurs="0" maxOccurs="1" name="Quality"
type="s0:OPCQuality" />
  </s:sequence>
  <s:attribute name="ValueTypeQualifier" type="s:QName" use="optional" />
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="ClientItemHandle" type="s:string" />
  <s:attribute name="Timestamp" type="s:dateTime" />
  <s:attribute name="ResultID" type="s:Qname" />
</s:complexType>

<s:complexType name="OPCQuality">
  <s:attribute default="good" name="QualityField" type="s0:qualityBits" />
  <s:attribute default="none" name="LimitField" type="s0:limitBits" />
  <s:attribute default="0" name="VendorField" type="s:unsignedByte" />
</s:complexType>

<s:simpleType name="qualityBits">
  <s:restriction base="s:string">
    <s:enumeration value="bad" />
    <s:enumeration value="badConfigurationError" />
    <s:enumeration value="badNotConnected" />
    <s:enumeration value="badDeviceFailure" />
    <s:enumeration value="badSensorFailure" />
    <s:enumeration value="badLastKnownValue" />
    <s:enumeration value="badCommFailure" />
    <s:enumeration value="badOutOfService" />
    <s:enumeration value="badWaitingForInitialData" />
    <s:enumeration value="uncertain" />
    <s:enumeration value="uncertainLastUsableValue" />
    <s:enumeration value="uncertainSensorNotAccurate" />
    <s:enumeration value="uncertainEUExceeded" />
    <s:enumeration value="uncertainSubNormal" />
    <s:enumeration value="good" />
```

```
    <s:enumeration value="goodLocalOverride" />
  </s:restriction>
</s:simpleType>

<s:simpleType name="limitBits">
  <s:restriction base="s:string">
    <s:enumeration value="none" />
    <s:enumeration value="low" />
    <s:enumeration value="high" />
    <s:enumeration value="constant" />
  </s:restriction>
</s:simpleType>
```

Tag Name	Description
DiagnosticInfo	Verbose server specific diagnostic information that provides additional information relative to errors. If the client requests this information, and if there is an ItemValue structure being returned, then the server is required to return item specific diagnostic information. If no diagnostic information is available, the server must not return the element in the response. The server can also provide diagnostic information if the request succeeded for the item e.g if the quality is not good.
ItemPath	A portion of the namespace pointing to the data. The ItemPath is server specific and the client should not make any implications about its content, structure, or consistency across servers. ItemPath may or may not imply node, server, or group. If an XML-DA server was front-ending a DA based server, then an example could be: \\NODE\OPC.DAServer.2. ItemPath is a hierarchical parameter. If **ItemPath** is Blank or missing at all levels of the hierarchy, then the ItemName is expected to be a fully qualified name.
ItemName	Identifier of the Data. It is free format (as in the OPC COM server).
ClientItemHandle	A String that can be passed by the client and be returned along with the data. If the Client includes the ClientItemHandle attribute, then Server must return it to the Client whenever the corresponding ItemValue structure is returned.
ResultID	If an error or a non-critical exception (minor problem) occurred this ID will contain a qualified name of an OPCError. If the server also returns verbose error messages the associated OPCError element will be located elsewhere in the message. This allows multiple instances of an Error ID to map to the same verbose OPC Error. See section 2.6 for more details about result codes. For pre-defined ResultIDs see section 3.1.9 on OPCError.
ValueTypeQualifier	A ValueTypeQualifier attribute is included when values of type time, date, and duration are tranmitted between client and server. The ValueTypeQualifier attribute uniquely identifies the intended value type of the value versus the type as transmitted across the wire.

	For all values of types other than time, date and duration, this attribute will be missing or if not missing, then ignored.
Value	A value with its type specified with the XML Schema 'type' attribute. The type may be any of the simple or array types defined in Section 2.7.
Timestamp	As in OPC COM, the Timestamp is the most accurate time the server is able to associate with a value. 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. If the **Value** is an array, then there is a single Timestamp that will be associated with all array elements. The server is responsible for determining/returning the most accurate timestamp. See discussion in Fundamental Concepts section for more detail on Timestamps. Timestamps are only returned if ReturnItemTime = True. Clients may specify a Timestamp when performing a *Write*.
Quality	Equivalent to the data contained within the DA Quality Word.
QualityField	A qualified name matching the OPC Quality Status and Substatus (i.e., the Quality BitField, and the Substatus BitField of the DA Quality Word). A "Good" quality may result in no QualityField attribute being returned. If "Bad", or "Uncertain" then a QualityField attribute will be returned. The server will NOT return a value if the quality is Bad, except as described in table below. The server is required to return a "reasonable" value when the quality is uncertain. Clients may specify a QualityField when performing a *Write*.
LimitField	A qualified name matching the OPC Limit Bit Field. A LimitField attribute will be returned for any Limit Status other than "none" irrespective of the value of QualityField. Clients may specify a LimitField when performing a *Write*.
VendorField	A numeric value matching the OPC Vendor Bit Field. VendorField attribute may be returned at Vendor's discretion. Clients may specify a VendorField when performing a *Write*.

Comments:

Some of the returned items are optional – See RequestOptions for further detail.

The following table summarizes the interactions of quality, and the value related items which are returned:

Item	Good	Bad	Uncertain
Value	"Good" value	If available return "Last Known Value" else NO value returned	"Reasonable" value
QualityField	Not returned	Variation of Bad If "Last Known Value" is available QualityField = "badLastKnownValue"	Variation of Uncertain
VendorField	May be returned at Vendor's discretion	May be returned at Vendor's discretion	May be returned at Vendor's discretion
LimitField	Will be returned for any Limit Status other than "none"	Will be returned for any Limit Status other than "none"	Will be returned for any Limit Status other than "none"
Timestamp	If **ReturnItemTime**, the time corresponding to the returned value	If **ReturnItemTime**, and "Last Known Value" is available, the time corresponding to the returned value	If **ReturnItemTime**, the time corresponding to the returned value

3.1.6 RequestOptions

Description

RequestOptions is the container of information that represents the options available to clients in most of the XML-DA requests.

```
<s:complexType name="RequestOptions">
  <s:attribute default="true" name="ReturnErrorText" type="s:boolean" />
  <s:attribute default="false" name="ReturnDiagnosticInfo" type="s:boolean"
/>
  <s:attribute default="false" name="ReturnItemTime" type="s:boolean" />
  <s:attribute default="false" name="ReturnItemPath" type="s:boolean" />
  <s:attribute default="false" name="ReturnItemName" type="s:boolean" />
  <s:attribute name="RequestDeadline" type="s:dateTime" use="optional" />
  <s:attribute name="ClientRequestHandle" type="s:string" />
  <s:attribute name="LocaleID" type="s:string" />
</s:complexType>
```

Tag Name	Description
ReturnErrorText	If TRUE (default) the server will return verbose error description. See also **LocaleID**, below and the **Error** type.
ReturnDiagnosticInfo	If TRUE the server will return verbose server specific diagnostic information to provide additional information relative to item specific errors. The server is required to return specific diagnostic information or a blank string if diagnostic information is not available.
ReturnItemTime	Indicates whether to return the timestamp for each item. Default is False which means item time will not be returned. Item values and quality are not client options and are returned according to the value's quality – see description in ItemValue section.
ReturnItemName	Indicates whether to return **ItemName** for each item. Default is False which means **ItemName** will not be returned. If the value is true, the passed ItemName must be also returned if it was unkown or invalid.
ReturnItemPath	Indicates whether to return **ItemPath** for each item. Default is False which means **ItemPath** will not be returned. If the value is true, the passed ItemPath must be also returned if it was unkown or invalid
RequestDeadline	Indicates the specific absolute time (in UTC) that the client wants to wait for the Server to process a response by either returning whatever data it might have or confirm that there was some error condition which prevents a successful response. Data for items, which is not available by that time, should be returned as errors. If the **RequestDeadline** is earlier than the current time of the server (**RcvTime**) then the whole request fails. If (**RequestDeadline - RcvTime**) <= 0 Then E TIMEDOUT fault

	If (**RequestDeadline** - current time) <= 0 Then E_TIMEDOUT errors for unprocessed items The request may timeout independently of **RequestDeadline** based on other system parameters. In the case of the system based timeout, the whole request will fail. The client can thus expect that the request will come to some closure based on the lesser of the **RequestDeadline** and the system based timeout. If omitted then the server will use some server specific period to process the response. The expectation is that the client and server are reasonably time synched which is necessary for this service to work properly. The client may gain some insight in calculating this attribute by reviewing **RcvTime**, and **ReplyTime** in the **ReplyBase** for responses. The server specific timeout might be determined by something equivalent to a TCP/IP timeout or perhaps by the time required to make a connection to a remote RTU. It is expected that the server specific maximum time will generally be no more than a minute or two although this depends on the details of the underlying system.
`ClientRequestHandle`	An optional value supplied by the client that will be returned with the response. In larger and more complex systems it helps the client to associate the replies with the proper requests.
`LocaleID`	An optional value supplied by the client that specifies the language for certain return data (see Section 2.4).

Comments:

3.1.7 ServerState

Description

ServerState is the container of information that represents the possible states of an OPC-XML-DA server. The meaning of these states is the same as in the *OPC Data Access Custom Interface Specification Version 3.0*.

```
<s:simpleType name="serverState">
  <s:restriction base="s:string">
    <s:enumeration value="running" />
    <s:enumeration value="failed" />
    <s:enumeration value="noConfig" />
    <s:enumeration value="suspended" />
    <s:enumeration value="test" />
    <s:enumeration value="commFault" />
  </s:restriction>
</s:simpleType>
```

3.1.8 ReplyBase

Description

ReplyBase is the container of information that represents the basic information for most responses.

```
<s:complexType name="ReplyBase">
  <s:attribute name="RcvTime" type="s:dateTime" use="required" />
  <s:attribute name="ReplyTime" type="s:dateTime" use="required" />
  <s:attribute name="ClientRequestHandle" type="s:string" />
  <s:attribute name="RevisedLocaleID" type="s:string" />
  <s:attribute name="ServerState" type="s0:serverState" use="required" />
</s:complexType>
```

Tag Name	Description
RcvTime	The time that the server received the request. Required attribute
ReplyTime	The time that the server returns the Response. Required attribute.
ClientRequestHandle	If supplied by the client in the request then this value is echoed back in the response.
RevisedLocaleID	If the client requested a LocaleID not supported by the server then the server will return its default locale id in this attribute. It is at the server's discretion as to whether the LocaleID is returned when the server is able to support the requested LocaleID.
ServerState	This attribute is used to communicate the current state and will always be returned. Required attribute

Comments:

3.1.9 OPCError

Description

OPCError is the container of information that represents the definition of OPC-XML-DA Errors.

```
<s:complexType name="OPCError">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Text" type="s:string" />
  </s:sequence>
  <s:attribute name="ID" type="s:QName" use="required" />
</s:complexType>
```

Tag Name	Description
ID	Contains the qualified name (ResultID) of the OPCError.
Text	The Textual representation of the Error. The encoding of the string is based on LocaleID. For each OPCError there will be a Text element.

OPC-XML-DA defines a series of standard result codes that have specific applications in data access operations. These codes are always qualified with the namespace http://opcfoundation.org/webservices/XMLDA/1.0/. The standard result codes are as follows.

Success codes

S_CLAMP	The value written was accepted but the output was clamped.
S_DATAQUEUEOVERFLOW	Not every detected change has been returned since the server's buffer reached its limit and had to purge out the oldest data.
S_UNSUPPORTEDRATE	The server does not support the requested rate but will use the closest available rate.

Error codes:

E_ACCESS_DENIED	The server denies access (read and/or write) to the specified item. This error is typically caused by Web Service security (e.g. globally disabled write capabilities).
E_BUSY	The server is currenly processing another polled refresh for one or more of the subscriptions.
E_FAIL	Unspecified error.
E_INVALIDCONTINUATIONPOINT	The continuation point is not valid.
E_INVALIDFILTER	The filter string is not valid.
E_INVALIDHOLDTIME	The hold time is too long (determined by server).
E_INVALIDITEMNAME	The item name does not conform the server's syntax.
E_INVALIDITEMPATH	The item path does not conform the server's syntax.
E_INVALIDPID	The property id is not valid for the item.

E_NOSUBSCRIPTION	An invalid subscription handle was passed to the request.
E_NOTSUPPORTED	The server does not support writing to the quality and/or timestamp.
E_OUTOFMEMORY	Ran out of memory.
E_RANGE	The value was out of range.
E_BADTYPE	The passed data type cannot be accepted for this item.
E_READONLY	The value is read only and may not be written to.
E_SERVERSTATE	The operation could not complete due to an abnormal server state.
E_TIMEDOUT	The operation took too long to complete (determined by server).
E_UNKNOWNITEMNAME	The item name is no longer available in the server address space.
E_UNKNOWNITEMPATH	The item path is no longer available in the server address space.
E_WRITEONLY	The value is write-only and may not be read from or returned as part of a write response.

Vendors may choose to create their own custom result codes, but these must be qualified with a vendor-specific namespace (i.e. http://company.com/etc). Please refer to the W3C XML 1.0 specification for more information about namespaces and qualified names. Note that vendors are still required to use the standard codes where specifically mentioned. In addition, the vendor specific codes should also follow the convention where critical errors are prefixed with 'E_' and none critical errors are prefixed with 'S_'.

Comments:

The OPCError elements will not be returned if the client is not interested in textual representations of the error (RequestOptions.ReturnErrorText = FALSE).

The server, if requested by the client, will return additional diagnostic information.

3.1.10 ItemProperty

Description

`ItemProperty` is the container of information that represents the properties that are accessed via the *Browse* and *GetProperties* services.

```
<s:complexType name="ItemProperty">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Value" />
  </s:sequence>
  <s:attribute name="Name" type="s:QName" use="required" />
  <s:attribute name="Description" type="s:string" />
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="ResultID" type="s:QName" />
</s:complexType>
```

Tag Name	Description
`Name`	Contains the Qualified name of the Property.
`Description`	Contains the Description of the Property.
`ItemPath`	If this Item Property can be read, written or subscribed to then ItemPath and ItemName together uniquely identify this property in the server's browse space. If ItemPath is empty, then ItemName by itself is a fully qualified name that uniquely identifies this element. If ItemPath and ItemName are both blank or missing, then this Item Property cannot be read, written or subscribed to.. See the corresponding section in the OPC DA Custom Specification which references **IOPCItemProperties::LookupItemIDs()**.
`ItemName`	See ItemPath.
`Value`	The current value of the property
`ResultID`	If an error or a non-critical exception (minor problem) occurred this ID will contain a qualified name. If the server also returns verbose error messages the respective OPCError elements will be located elsewhere in the message. This allows multiple instances of a Result ID to map to the same verbose OPC Error. See section 2.6 for more details about Resultcodes. For pre-defined ResultIds see section 3.1.9 on OPCError.

Comments:

ItemProperty is analogous to the data returned from **IOPCItemProperties::GetItemProperties()** in the OPC DA Custom Specification. That specification uses DWORD IDs to identify a property versus the use of qualified names above. Vendors may choose to create their own custom item properties, but these must be qualified with a vendor-specific namespace (i.e. "http://company.com/etc"). Please refer to the W3C XML 1.0 specification for more information about namespaces and qualified names. The IDs used in the DA specification map to the qualified names as follows:

ID	OPC-XML-DA Qualified Name	STANDARD DESCRIPTION	DATA TYPE
1	dataType	"Item Canonical DataType"	QName
2	value	"Item Value"	anyType
3	quality	"Item Quality"	OPCQuality
4	timestamp	"Item Timestamp"	dateTime

5	accessRights	"Item Access Rights"	string – one of the following valid values must be used:
			`"unknown"`
			`"readable"`
			`"writable"`
			`"readWritable"`

6	scanRate	"Server Scan Rate" This represents the fastest rate (in milliseconds) 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' or fastest RequestedSamplingRate which could be used if this item were subscribed to. The accuracy of this value (the ability of the server to attain 'best case' performance) may be greatly affected by system load and other factors.	float

7	euType	"Item EU Type"	string – one of the following valid values must be used:
			`"noEnum"`
			`"analog"`
			`"enumerated"`

8	euInfo	"Item EUInfo" If item 7 "Item EU Type" is "Enumerated" then EUInfo will contain an array of strings which correspond to sequential numeric values (0, 1, 2, etc.) (Example: <string>OPEN</string> <string>CLOSE</string> <string>IN TRANSIT</string> etc.)	ArrayOfString

9-99		Reserved for future OPC use	
100	engineeringUnits	"EU Units" e.g. “DEGC” or “GALLONS”	string
101	description	"Item Description" e.g. “Evaporator 6 Coolant Temp”	string
102	highEU	"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	double
103	lowEU	"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	double
104	highIR	"High Instrument Range" Present only for ‘analog’ data. This represents the highest value that can be returned by the instrument. e.g. 9999.9	double
105	lowIR	"Low Instrument Range" Present only for ‘analog’ data. This represents the lowest value that can be returned by the instrument. e.g. -9999.9	double
106	closeLabel	"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.	string
107	openLabel	"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	string

		e.g. "STOP", "OPEN", "DISABLE", "UNSAFE" ,etc.	
108	timeZone	"Item Timezone" The time difference (in minutes) between the item's 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.	unsignedInt
109	minimumValue	"Minimum Value" The smallest positive value that can be stored in the item. See Section 2.7.4 for a complete explanation.	Same as the data type for the item.
110	maximumValue	"Maximum Value" The largest positive value that can be stored in the item. See Section 2.7.4 for a complete explanation.	Same as the data type for the item.
111	valuePrecision	"Value Precision" The maximum precision that can be stored in the item. See Section 2.7.4 for a complete explanation.	double
112-199		Reserved for future OPC use. Additional IDs may be added without impacting the SupportedInterfaceVersions.	
		IDs 300 to 399 are reserved for use by OPC Alarms and Events. **See the OPC Alarm and Events specification for additional information.**	

3.2 GetStatus

3.2.1 GetStatus

Description

`GetStatus` is the container of information that represents the *GetStatus* request.

The purpose of the *GetStatus* service is:

1. It provides a common mechanism for checking the status of the server - whether it is operational or in need of maintenance.

2. It provides a common mechanism for obtaining vendor-specific information about the server that is not available through the other OPC services (version number, etc).

3. Provides insight for clients as to the relative time synchronization between the client and server. As an example, this information is useful for *Read* requests.

```
<s:element name="GetStatus">
  <s:complexType>
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="ClientRequestHandle" type="s:string" />
  </s:complexType>
</s:element>
```

Tag Name	Description
`LocaleID`	An optional value supplied by the client that specifies the language for textual status data.
`ClientRequestHandle`	An optional value supplied by the client that will be returned with the response. In larger and more complex systems it helps the client to associate the replies with the proper requests.

Comments:

Example

```
<soap:Body>
  <GetStatus
    LocaleID="de-AT"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
  />
</soap:Body>
```

3.2.2 GetStatusResponse

Description

GetStatusResponse is the container of information that represents the *GetStatus* response. The server is required to return valid values for all of the items as described below.

```
<s:element name="GetStatusResponse">
  <s:complexType>
    <s:sequence>
      <s:element
        minOccurs="0" maxOccurs="1"
        name="GetStatusResult"
        type="s0:ReplyBase"
      />
      <s:element minOccurs="0" maxOccurs="1" name="Status"
type="s0:ServerStatus" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ServerStatus">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="StatusInfo" type="s:string"
/>
    <s:element minOccurs="0" maxOccurs="1" name="VendorInfo" type="s:string"
/>
    <s:element
      minOccurs="0" maxOccurs="unbounded"
      name="SupportedLocaleIDs"
      type="s:string"
    />
    <s:element
      minOccurs="0" maxOccurs="unbounded"
      name="SupportedInterfaceVersions"
      type="s0:interfaceVersion"
    />
  </s:sequence>
  <s:attribute name="StartTime" type="s:dateTime" use="required" />
  <s:attribute name="ProductVersion" type="s:string" />
</s:complexType>

<s:simpleType name="interfaceVersion">
  <s:restriction base="s:string">
    <s:enumeration value="XML_DA_Version_1_0" />
  </s:restriction>
</s:simpleType>
```

Tag Name	Description
GetStatusResult	For a detailed description of ReplyBase see the separate section, above. Required Element.
StatusInfo	String providing additional information about the server state. This may be locale-specific.

VendorInfo	Vendor specific string providing additional information about the server. It is recommended that this mention the name of the company and the type of device(s) supported. This string may be locale-specific.
SupportedLocaleIDs	1 or more Locale IDs supported by the server. Required element.
SupportedInterfaceVersions	Array of Strings, containing the versions of the XML-DA Specification that this server supports. (Required to provide at least 1) The text associated with this Specification is: "XML_DA_Version_1_0"
StartTime	Time (UTC) the server was started. This is constant for the server instance and is not reset when the server changes states. Each instance of a server should keep the time when the process started.
ProductVersion	Version String, containing "Major", "Minor" and "Build" number.

Comments:

Abnormal Result Codes:

Faults:

The server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.

Example

```
<soap:Body>
  <GetStatusResponse
xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <GetStatusResult
      RcvTime="2003-05-26T20:17:42.4781250-07:00"
      ReplyTime="2003-05-26T20:17:42.5781250-07:00"
      RevisedLocaleID="de"
      ServerState="running"
    />
    <Status
      StartTime="2003-05-26T20:16:45.0937500-07:00"
      ProductVersion="1.00.1.00"
    >
      <VendorInfo>OPC XML Data Access 1.00 Sample Server</VendorInfo>
      <SupportedLocaleIDs>en</SupportedLocaleIDs>
      <SupportedLocaleIDs>en-US</SupportedLocaleIDs>
      <SupportedLocaleIDs>de</SupportedLocaleIDs>

<SupportedInterfaceVersions>XML_DA_Version_1_0</SupportedInterfaceVersions>
    </Status>
  </GetStatusResponse>
</soap:Body>
```

3.3 Read

3.3.1 Read

Description

Read is the container of information that represents the *Read* request.

This service provides the ability to read the value and quality for one or more items. Other attributes, such as timestamp, can optionally be requested for items. Other information can also be optionally requested such as the inclusion of verbose error messages in the response.

The items to be read are contained in an Item List. The client specifies request specific attributes that allow the server to more appropriately respond to the client's data needs. Certain of these attributes are "hierarchical" in nature and are described elsewhere in the document.

The *Read* request runs to completion before the response is returned. The server obtains the data for an item, or it determines that the data cannot be read. It places either the data or an error code for each requested item into the *ReadResponse*, according to the structure and order of the Items in the request.

The client may request for the server to return the subset of verbose error messages that correspond to the unique error codes encountered in the list of values. The verbose error messages follow the list of values/error codes.

The data can be read from a server's cache, in which case, it should be accurate to within the optional MaxAge attribute specified for the item in the request. Alternatively, the server may be front-ending a device, and certain data requests will cause a read from the underlying physical device. The cache values that met the MaxAge attribute will not have to be reevaluated after the device read is performed. The exact implementation of cache and device reads is not defined by this specification.

In the WSDL extract below, the attribute minOccurs is set to 0 for Items (in ReadRequestItemList) to be compatible with code generation tools. However, at least one Item is required in the ReadRequestItemList, else an E_FAIL will be returned.

```
<s:element name="Read">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="Options"
        type="s0:RequestOptions" />
      <s:element minOccurs="0" maxOccurs="1"
        name="ItemList"
        type="s0:ReadRequestItemList" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ReadRequestItemList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Items"
      type="s0:ReadRequestItem" />
  </s:sequence>
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ReqType" type="s:QName"/>
  <s:attribute name="MaxAge" type="s:int" />
</s:complexType>

<s:complexType name="ReadRequestItem">
```

```
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ReqType" type="s:QName" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="ClientItemHandle" type="s:string" />
  <s:attribute name="MaxAge" type="s:int" />
</s:complexType>
```

Tag Name	Description
Options	For a detailed description of these options see the separate section (RequestOptions), above.
ItemList	The container for the individual Items.
Items	A container tag of the item information that follows. It is expected that there are one or more Items per ItemList. For a detailed description of RequestItem see the separate section, above.
Hierarchical Parameters: The following parameters are hierarchical parameter, i.e., it can occur either on the list or on the item level. A value specified for an item will override the value on list level.	
ItemPath	Same as attribute described in Section 3.1.3.
ReqType	Same as attribute described in Section 3.1.3.
MaxAge	Indicates the requested freshness of the data in number of milliseconds. The data should be no older than this value. If omitted or if the value is 0 at all levels of the hierarchy then the server should return the most accurate data available (which is analogous to a "DEVICE" read in the COM server – reference the corresponding section of the OPC DA Specification to get further clarification on this concept.). The server must not return an error or bad quality if the most accurate data available has an older time stamp than MaxAge.

Comments:

The Server will maintain the order of Items within the ItemList.

Read requests are expected to be one-shot operations, and there will not be any assumed relationship (contract) between client and server. *Read* requests are also synchronous in nature, i.e., the server will return all requested values or errors for those values which it cannot retrieve.

If data is needed on a regular basis, clients should use the Subscription services.

Example

```
<soap:Body>
  <Read xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <Options
      ReturnErrorText="false"
      ReturnItemTime="true"
      ReturnItemName="true"
      LocaleID="en" />
    <ItemList>
      <Items ItemName="Simple Types/UInt" />
      <Items ItemName="Simple Types/Int" />
      <Items ItemName="Simple Types/Float" />
    </ItemList>
  </Read>
```

```
</soap:Body>
```

3.3.2 ReadResponse

Description

ReadResponse is the container of information that represents the *Read* response.

In the WSDL extract below, the attribute minOccurs is set to 0 for Items (in ReplyItemList) to be compatible with code generation tools. However, the number of Items must match the corresponding number in the request.

```
<s:element name="ReadResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="ReadResult"
        type="s0:ReplyBase" />
      <s:element minOccurs="0" maxOccurs="1"
        name="RItemList"
        type="s0:ReplyItemList" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="Errors"
        type="s0:OPCError" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ReplyItemList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Items"
      type="s0:ItemValue" />
  </s:sequence>
  <s:attribute name="Reserved" type="s:string" />
</s:complexType>
```

Tag Name	Description
ReadResult	For a detailed description of ReplyBase see the separate section, above. Required Element.
RItemList	A container for the individual Item elements. Note that the "Reserved" attribute in the ReplyItemList type exists in order to prevent WSDL based code generation tools from representing the returned list as an array of ItemValues.
Items	A container of the item elements and their value information. It is expected that there are one or more Item elements per RItemList. The *ReadResponse* will maintain the Item order in the *Read* request. If an item is write-only the server will return E_WRITEONLY in the response for this item. The response will provide no value element for the affected item.
Errors	An array of OPCError elements that is appropriate for this Response. Errors are only present if Items contain result codes.

Comments:

The Server will maintain the order of Items within RItemList.

Abnormal Result Codes:

One of the following codes can be part of any of the values.

E_ACCESS_DENIED	See description in Section 3.1.9.
E_BADTYPE	See description in Section 3.1.9.
E_INVALIDITEMNAME	See description in Section 3.1.9.
E_INVALIDITEMPATH	See description in Section 3.1.9.
E_RANGE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9
E_UNKNOWNITEMNAME	See description in Section 3.1.9.
E_UNKNOWNITEMPATH	See description in Section 3.1.9.
E_WRITEONLY	See description in Section 3.1.9.
E_XXX, S_XXX	Vendor-specific result code.

Faults:

The server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.

Example

```
<soap:Body>
  <ReadResponse xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <ReadResult
      RcvTime="2003-05-27T00:15:36.6400000-07:00"
      ReplyTime="2003-05-27T00:15:36.7500000-07:00"
      ServerState="running"
    />
    <RItemList>
      <Items
        ItemName="Simple Types/UInt"
        Timestamp="2003-05-27T00:15:36.7343750-07:00">
        <Value xsi:type="xsd:unsignedInt">4294967295</Value>
      </Items>
      <Items
        ItemName="Simple Types/Int"
        Timestamp="2003-05-27T00:15:36.7343750-07:00">
        <Value xsi:type="xsd:int">2147483647</Value>
      </Items>
      <Items
        ItemName="Simple Types/Float"
        Timestamp="2003-05-27T00:15:36.7343750-07:00">
       <Value xsi:type="xsd:float">3.402823E+38</Value>
```

```
      </Items>
    </RItemList>
  </ReadResponse>
</soap:Body>
```

3.4 Write

3.4.1 Write

Description

Write is the container of information that represents the *Write* request.

This service writes the value for one or more items. Optionally, the Time, QualityField, LimitField, VendorField attributes of the value also can be written.

The client may tailor the information to be returned in the corresponding *WriteResponse* and as such may optionally request the inclusion of verbose error messages in the response.

The values to be written are contained in an ItemList. The client specifies request specific attributes which allow the server to respond to the client's data write needs. Certain of these attributes are "hierarchical" in nature and described elsewhere in the document. The client also may request a subsequent read of the items, allowing it to obtain the results of the writes.

The service runs to completion before the response is returned. The server writes the data for each item, or it determines that the data cannot be written.

If requested, after all writes complete, the server performs a read of the items. The server places either the data or an error code (write or read) for each requested item into the *WriteResponse*, matching the structure and order of the request.

The server and the scope of the data that it represents will determine the data destination, i.e., cache, or underlying device. The exact implementation of a cache or device is not defined by this specification.

In the WSDL extract below, the attribute minOccurs is set to 0 for Items (in WriteRequestItemList) to be compatible with code generation tools. However, at least one Item is required in the ItemList, else an E_FAIL will be returned.

```
<s:element name="Write">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="Options"
        type="s0:RequestOptions" />
      <s:element minOccurs="0" maxOccurs="1"
        name="ItemList"
        type="s0:WriteRequestItemList" />
    </s:sequence>
    <s:attribute name="ReturnValuesOnReply" type="s:boolean" use="required"
/>
  </s:complexType>
</s:element>

<s:complexType name="WriteRequestItemList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Items"
      type="s0:ItemValue" />
  </s:sequence>
  <s:attribute name="ItemPath" type="s:string" />
</s:complexType>
```

Tag Name	Description

Options	For a detailed description of these options see the separate section (RequestOptions), above.
ReturnValuesOnReply	Indicates whether to return the Value for each item (i.e., the Value of the ItemValue structure). The returned value is the value "accepted" by the server, or device. If a Value is returned, then the associated QualityField will also be returned, and as appropriate the LimitField, and VendorField. The returned value is equivalent to the value which would be returned by the server to the client if the client had performed a read request directly after the write request. Values are never returned if the Write fails. If an item is write-only the server will return E_WRITEONLY in the response for this item. The response will provide no value element for the affected item.
ItemList	The container for the individual Items.
Items	A container of the item information that follows. It is expected that there are one or more Items in the ItemList. The typical information to be transported is the ItemName, and a Value. The other tags are optional and are valid only if the Server supports the ability to write other than value.
Item.Value	One or more **Value(s)** are always supplied. The *Write* request allows the client to specify one or more of the following attributes of a value (Time, QualityField, LimitField, and VendorField). The servers must either support writing none of these qualifying attributes, or they must support writing all on a per item basis. Writes of only Value, or Value/Quality/Time/Bits are atomic writes. Either they all are written, or none are written. Examples of clients which might leverage this capability are simulation clients, or advanced control type clients. In both cases the client is generating the Item Value, and would benefit from the ability to write one or more of: Time, QualityField, LimitField, or VendorField. If the client specifies a subset of the attributes, then the server will only attempt to write the supplied subset. 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 the E_NOTSUPPORTED error code. In the case of a quality other than Good (and if the server supports the writing of QualityField), the value should be some client specific default item value (typically 0 or a Null string), and will be ignored by the server.
Hierarchical Parameter: The following parameters are hierarchical parameter, i.e., it can occur either on the list or on the item level. A value specified for an item will override the value on list level.	

ItemPath	Same as attribute described in Section 3.1.3.

Comments:

The Server will maintain the order of Items within the ItemList.

The Server should maintain the order of the write request as much as possible in performing the actual writes, but the client should not count on any expectation of that order. The server may reorder the actual writes based on server, and device constraints such as performance, device availability, etc.

The server will attempt to convert the client's supplied value to the server's canonical representation of the data else return an E_BADTYPE error.

A server **must not** allow conversions from a string to any other data type. This restriction is necessary because different representations of the same value in different locales can be ambiguous and lead to unexpected behavior. For example, the decimal number 1.234 is represented in a German locale as the string "1,234". The server must return an E_BADTYPE error if a client attempts to write a string value to any item with a canonical data type that is not a string.

Example:

```
<soap:Body>
  <Write xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <Options
      ReturnErrorText="false"
      ReturnItemName="true"
      LocaleID="en"
    />
    <ItemList>
      <Items ItemName="Simple Types/UInt">
        <Value xsi:type="xsd:unsignedInt">4294967295</Value>
      </Items>
      <Items ItemName="Simple Types/Int">
        <Value xsi:type="xsd:int">2147483647</Value>
      </Items>
      <Items ItemName="Simple Types/Float">
        <Value xsi:type="xsd:float">3.402823E+38</Value>
      </Items>
    </ItemList>
  </Write>
</soap:Body>
```

3.4.2 WriteResponse

Description

Write is the container of information that represents the *Write* response.

In the WSDL extract below, the attribute minOccurs is set to 0 for Items (in ReplyItemList) to be compatible with code generation tools. However, the number of Items must match the corresponding request.

```
<s:element name="WriteResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="WriteResult"
        type="s0:ReplyBase" />
      <s:element minOccurs="0" maxOccurs="1"
        name="RItemList"
        type="s0:ReplyItemList" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="Errors"
        type="s0:OPCError" />
    </s:sequence>
  </s:complexType>
</s:element>
```

Tag Name	Description
WriteResult	For a detailed description of ReplyBase see the separate section, above. Required Element.
RItemList	The container for the individual Item elements
Items	A container of the item information that follows. It is expected that there are one or more Item elements in the list. The *WriteResponse* will maintain the order of Items in the *Write* request. The data type of the returned value(s) must match the data type of the value(s) being supplied in the *Write* request. The Value element is only present if ReturnValuesOnReply was True. Timestamp is only present if ReturnItemTime was True.
Errors	An array of OPCError elements that is appropriate for this Response. Error elements are only present if Item elements contain result codes.

Comments:

The Server will maintain the order of Items within the RItemList.

Abnormal Result Codes:

One of the following codes can be part of any of the values.

E_ACCESS_DENIED	See description in Section 3.1.9.
E_BADTYPE	See description in Section 3.1.9.
E_INVALIDITEMID	See description in Section 3.1.9.

E_INVALIDITEMNAME	See description in Section 3.1.9.
E_INVALIDITEMPATH	See description in Section 3.1.9.
E_NOTSUPPORTED	See description in Section 3.1.9.
E_RANGE	See description in Section 3.1.9.
E_READONLY	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9
E_UNKNOWNITEMNAME	See description in Section 3.1.9.
E_UNKNOWNITEMPATH	See description in Section 3.1.9.
E_WRITEONLY	See description in Section 3.1.9.
S_CLAMP	See description in Section 3.1.9.
E_XXX, S_XXX	Vendor-specific result code.

Faults:

The server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.

Example:

```
<soap:Body>
  <WriteResponse xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
  <WriteResult
    RcvTime="2003-05-27T05:19:26.3687500-07:00"
    ReplyTime="2003-05-27T05:19:26.4687500-07:00"
    ServerState="running" />
  <RItemList>
    <Items ItemName="Simple Types/UInt" />
    <Items ItemName="Simple Types/Int" />
    <Items ItemName="Simple Types/Float" />
  </RItemList>
  </WriteResponse>
</soap:Body>
```

3.5 Subscribe

For a detailed description of the OPC-XML-DA Subscription mechanism see the section under ***Fundamental Concepts***.

3.5.1 Subscribe

Description

`Subscribe` is the container of information that represents the *Subscribe* request.

In the WSDL extract below, the attribute minOccurs is set to 0 for Items (in SubscribeRequestItemList) to be compatible with code generation tools. However, at least one Item is required in the list, else an E_FAIL will be returned.

```
<s:element name="Subscribe">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="Options"
        type="s0:RequestOptions" />
      <s:element minOccurs="0" maxOccurs="1"
        name="ItemList"
        type="s0:SubscribeRequestItemList" />
    </s:sequence>
    <s:attribute name="ReturnValuesOnReply" type="s:boolean" use="required"
/>
    <s:attribute default="0" name="SubscriptionPingRate" type="s:int" />
  </s:complexType>
</s:element>

<s:complexType name="SubscribeRequestItemList">
  <s:sequence>
    <s:element
      minOccurs="0" maxOccurs="unbounded"
      name="Items"
      type="s0:SubscribeRequestItem" />
  </s:sequence>
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ReqType" type="s:QName" />
  <s:attribute name="Deadband" type="s:float" />
  <s:attribute name="RequestedSamplingRate" type="s:int" />
  <s:attribute name="EnableBuffering" type="s:boolean" />
</s:complexType>

<s:complexType name="SubscribeRequestItem">
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ReqType" type="s:QName" use="required" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="ClientItemHandle" type="s:string" />
  <s:attribute name="Deadband" type="s:float" />
  <s:attribute name="RequestedSamplingRate" type="s:int" />
  <s:attribute name="EnableBuffering" type="s:boolean" />
</s:complexType>
```

Name	Description
`Options`	For a detailed description of these options see the separate

	section (RequestOptions), above.
ReturnValuesOnReply	If TRUE the server will return item values which are readily available for inclusion in the *SubscribeResponse*. Depending on when the *SubscriptionPolledRefresh* is requested, these items may or may not be updated in the first *SubscriptionPolledRefresh*. If FALSE the server must not deliver any item values in the *SubscribeResponse*.
SubscriptionPingRate	This is a required attribute. The SubscriptionPingRate is the requested rate in milliseconds that the server should reevaluate the existence of the client. If the client has not had any communication in the specified period, then the Server is free to clean up all resources associated with that client for this Subscription. The server should attempt to honor the client's request, but it may reevaluate the existence of the client at a rate faster than the SubscriptionPingRate based on its own implementation, and resource constraints. If the SubscriptionPingRate is 0, then the server will use its own algorithm to reevaluate the existence of the client. It is highly recommended that clients always specify a non-zero ping rate since specifying zero will allow the server to choose a ping rate that the client will not have knowledge of and may be inappropriate.
ItemList	The container tag for the individual Items.
Items	A container tag of the item information. It is expected that there are one or more Item elements in the list.
Hierarchical Parameters: The following parameters are hierarchical parameters, i.e., they can occur either on the list or on the item level. A value specified for an item will override the value on list level.	
Deadband	Specifies the percentage of full engineering unit range of an item's value that must change prior to being returned in a *SubscriptionPolledRefresh* response. The deadband value shall be in the range 0-100 percent and only applies to analog (integer or float) types. The deadband will also apply to array types. The entire array is returned if any array element exceeds the deadband threshold. Server default is 0. See the OPC DA Custom Specification for further detail.
RequestedSamplingRate	The client specifies the rate in milliseconds at which the server should check for value changes. If no item-specific sampling rate is specified, sampling will be based on the rate of the item list. See the section on Data Management Optimization in this

	document and OPC DA Custom Specification for further detail.
EnableBuffering	If True, the client is requesting that the server use the RequestedSamplingRate to check for value changes and save all changes in a buffer for return to the client at the next SubscriptionPolledRefresh request. See the section on Data Management Optimization in this document and OPC DA Custom Specification for further detail.

Comments:

The Server will maintain the order of Items within the list in the Response.

Responses to subscribe or poll requests usually return only a subset of all subscribed items. To be able to identify them the client has to assign unique values to the handles of items (ClientItemHandle).

Example:

```
<soap:Body>
  <Subscribe
    ReturnValuesOnReply="true"
    SubscriptionPingRate="10000"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
  >
  <Options
    ReturnErrorText="false"
    ReturnItemTime="true"
    ReturnItemName="true"
    LocaleID="en"
  />
  <ItemList RequestedSamplingRate="1000">
    <Items
      ItemName="Analog Types/Double"
      ClientItemHandle="e035d707-e27a-4b06-b103-fea125ce5ca4" />
    <Items
      ItemName="Analog Types/Int"
      ClientItemHandle="fdce6f30-b8d4-4eeb-becf-6deeacdc7f36" />
  </ItemList>
  </Subscribe>
</soap:Body>
```

3.5.2 SubscribeResponse

Description

SubscribeResponse is the container of information that represents the *Subscribe* response.

In the WSDL extract below, the attribute minOccurs is set to 0 for Items (in SubscribeReplyItemList) to be compatible with code generation tools. See text below for the scenarios on when and how these are included in the response.

```
<s:element name="SubscribeResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="SubscribeResult"
        type="s0:ReplyBase" />
      <s:element minOccurs="0" maxOccurs="1"
        name="RItemList"
        type="s0:SubscribeReplyItemList" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="Errors"
        type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="ServerSubHandle" type="s:string" />
  </s:complexType>
</s:element>

<s:complexType name="SubscribeReplyItemList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Items"
      type="s0:SubscribeItemValue" />
  </s:sequence>
  <s:attribute name="RevisedSamplingRate" type="s:int" />
</s:complexType>

<s:complexType name="SubscribeItemValue">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      name="ItemValue"
      type="s0:ItemValue" />
  </s:sequence>
  <s:attribute name="RevisedSamplingRate" type="s:int" />
</s:complexType>
```

Name	Description
SubscribeResult	For a detailed description of *ReplyBase* see the separate section, above. Required Element.
ServerSubHandle	Supplied by the Server. It must be used for *SubscriptionPolledRefresh* and *SubscriptionCancel*. ServerSubHandle is specific to the client making the request.
RevisedSamplingRate	The server responds to the client with the actual update rate that it can support.

	Refer to the section on “Data Management Optimization” for further detail.
RItemList	The RItemList structure is the container for Item elements which carry error or value information. The readily available item values are sent back via Item elements in RItemList if and only if the client requested them with “ReturnValuesOnReply” (see *Subscribe*). If the server does not have a value for some of the items at the time of *Subscribe*, the response will provide no value element for the affected item. If error conditions (like invalid item name or unsupported rate) are detected by the server, then Item Elements will be returned to communicate the error conditions. If ReturnValuesOnReply is “false” and no errors are found, RItemList will be empty. .
Items	No additional comments.
RevisedSamplingRate	The server responds to the client with the actual sampling rate that it can support. Refer to the section on “Data Management Optimization” for further detail.
Errors	An array of OPCError elements that is appropriate for this Response. OPCError elements are only present if Item elements contain result codes.

Comments:

SubscribeResponse is the server’s response to the *Subscribe* request.

A subscription will be created if at least one of the specified items in the passed item list is valid.

If all items are rejected the server will still return Item elements with the error codes. However, no subscription will be created and an empty string will be returned as “ServerSubHandle”.

Abnormal Result Codes:

One of the following codes can be part of any of the values.

E_ACCESS_DENIED	See description in Section 3.1.9.
E_BADTYPE	See description in Section 3.1.9.
E_INVALIDITEMNAME	See description in Section 3.1.9.
E_INVALIDITEMPATH	See description in Section 3.1.9.
E_RANGE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.
E_UNKNOWNITEMNAME	See description in Section 3.1.9.
E_UNKNOWNITEMPATH	See description in Section 3.1.9.
E_WRITEONLY	See description in Section 3.1.9.
S_UNSUPPORTEDRATE	See description in Section 3.1.9.

E_XXX, S_XXX	Vendor-specific result code.

Faults:

The Server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.

Example:

```
<soap:Body>
  <SubscribeResponse
    ServerSubHandle="f6f7900f-3962-4965-abba-31607ce5246b"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <SubscribeResult
      RcvTime="2003-05-27T06:16:23.7750000-07:00"
      ReplyTime="2003-05-27T06:16:23.8750000-07:00"
      RevisedLocaleID=""
      ServerState="running"
    />
    <RItemList>
      <Items>
        <ItemValue
          ItemName="Analog Types/Double"
          ClientItemHandle="e035d707-e27a-4b06-b103-fea125ce5ca4"
          Timestamp="2003-05-27T06:15:56.0625000-07:00"
        >
          <Value xsi:type="xsd:double">3.8060233744357421</Value>
          <Quality />
        </ItemValue>
      </Items>
      <Items>
        <ItemValue
          ItemName="Analog Types/Int"
          ClientItemHandle="fdce6f30-b8d4-4eeb-becf-6deeacdc7f36"
          Timestamp="2003-05-27T06:15:56.0625000-07:00"
        >
          <Value xsi:type="xsd:int">500</Value>
        <Quality />
        </ItemValue>
      </Items>
    </RItemList>
  </SubscribeResponse>
</soap:Body>
```

3.6 SubscriptionPolledRefresh

Refreshes the data items from the last *SubscriptionPolledRefresh*. For a detailed description of the OPC-XML-DA Subscription mechanism see the section under ***Fundamental Concepts***.

3.6.1 SubscriptionPolledRefresh

Description

`SubscriptionPolledRefresh` is the container of information that represents the *SubscriptionPolledRefresh* request.

In the WSDL extract below, the attribute minOccurs is set to 0 for ServerSubHandles to be compatible with code generation tools. However, at least one handle is required.

```
<s:element name="SubscriptionPolledRefresh">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="Options"
        type="s0:RequestOptions" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="ServerSubHandles"
        type="s:string" />
    </s:sequence>
    <s:attribute name="HoldTime" type="s:dateTime" use="optional" />
    <s:attribute default="0" name="WaitTime" type="s:int" use="required" />
    <s:attribute default="false" name="ReturnAllItems" type="s:boolean" />
  </s:complexType>
</s:element>
```

Name	Description
`Options`	For a detailed description of these options see the separate section (RequestOptions) above.
`RequestDeadline`	**RequestDeadline** is only applicable for the condition of **RcvTime** being after the **RequestDeadline**. For all other cases, **HoldTime** and **WaitTime** control the server behavior after the initial receipt of the *SubscriptionPolledRefresh* request.
`ServerSubHandles`	Supplied by the Server in the *SubscribeResponse, i*t is used by the server to identify the Subscription to be polled. Multiple **ServerSubHandles** may be supplied. The server will respond with the changes in data associated with all supplied **ServerSubHandles**. The Server will maintain the order of Items within each polled subscription list, and for subscriptions in the response (relative to the ServerSubHandles) – even if some subscriptions or some items in the subscriptions are missing.
`HoldTime`	Instructs the server to hold off returning from the refresh service call until the absolute time of the server is equal or greater than this value. This attribute is optional. If **HoldTime** is missing, then **WaitTime** is ignored.

WaitTime	Instructs the server to wait the specified number of milliseconds after **HoldTime** before returning if there are no changes to report. A change in one of the subscribed items, during this wait period, will result in the service returning immediately rather than completing the wait time.
ReturnAllItems	If set to FALSE, then the server will return only the changed Items between this *SubscriptionPolledRefresh* request and the previous request. If TRUE the server will return all Items specified by the original *Subscribe*. The server will wait the **HoldTime** but then return with all current values (and any buffered values if **EnableBuffering**) ignoring the change status of the items. That is the **WaitTime** is not considered under this condition

Comments:

Please note that the server may have to initiate parallel processing of multiple subscriptions in order to respond to the *SubscriptionPolledRefresh* request. This behavior is necessitated based on the hold time and wait time parameters being only applied once.

The first SubscriptionPolledRefresh after the Subscribe must return only Items with changed values if the Subscribe returned the values (ReturnValuesOnReply = true). The first SubscriptionPolledRefresh after the Subscribe must return all items if the Subscribe did not return the values (ReturnValuesOnReply = false).

Example:

```
<soap:Body>
  <SubscriptionPolledRefresh
    Holdtime="2003-05-27T06:16:31.8750000-07:00"
    Waittime="0"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
  >
    <Options
      ReturnErrorText="false"
      ReturnItemTime="true"
      ReturnItemName="true"
      LocaleID="en"
    />
    <ServerSubHandles>f6f7900f-3962-4965-abba-
31607ce5246b</ServerSubHandles>
  </SubscriptionPolledRefresh>
</soap:Body>
```

3.6.2 SubscriptionPolledRefreshResponse

Description

SubscriptionPolledRefreshResponse is the container of information that represents the *SubscriptionPolledRefresh* response.

```
<s:element name="SubscriptionPolledRefreshResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="SubscriptionPolledRefreshResult"
        type="s0:ReplyBase" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="InvalidServerSubHandles"
        type="s:string" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="RItemList"
        type="s0:SubscribePolledRefreshReplyItemList" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="Errors"
        type="s0:OPCError" />
    </s:sequence>
    <s:attribute default="false" name="DataBufferOverflow" type="s:boolean"
/>
  </s:complexType>
</s:element>

<s:complexType name="SubscribePolledRefreshReplyItemList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="Items"
type="s0:ItemValue" />
  </s:sequence>
  <s:attribute name="SubscriptionHandle" type="s:string" />
</s:complexType>
```

Name	Description
SubscriptionPolledRefreshResult	For a detailed description of ReplyBase see the separate section, above. Required Element.
InvalidServerSubHandles	The server will identify 0 or more ServerSubHandles that were invalid.
RItemList	One RItemList for each subscription of which items have to be returned. A RItemList for each polled (and valid) subscription handle is sent if the client requested them with “ReturnAllItems”. If “ReturnAllItems” is FALSE, the server only returns Items which had changed. Each RItemList contains the SubscriptionHandle. Within each list the Items will be returned in a relative order based on their relative order in the original *Subscribe* even if some of the Items are

	missing because the values have not changed. If there are no values which have changed, the server will respond with a response without any RItemList. If EnableBuffering = False then the server will send only the latest value that it is maintaining for those changed items. If EnableBuffering = True then the server will send all value changes (Last Changed Value and any buffered values) for those changed items since the last *SubscriptionPolledRefresh*.
`DataBufferOverflow`	This is an indicator that several changes for individual items occurred, but not all of these changes could be buffered due to resource limitations. The server is required to provide at least the most recent change for each item that changed since the last update. The individual items will indicate whether they were impacted by this resource limitation. For more details on buffering see the section on Buffered Data and the OPC DA Custom Specification for additional details on this topic..
`Errors`	An array of OPCError elements that is appropriate for this Response. OPCError elements are only present if Item Elements contain result codes or if 1 or more ServerSubHandles were invalid.

Comments:

There is no implied ordering of the data returned based on the ServerSubHandles.

Abnormal Result Codes:

One of the following codes can be part of any of the values.

E_ACCESS_DENIED	See description in Section 3.1.9.
E_BADTYPE	See description in Section 3.1.9.
E_RANGE	See description in Section 3.1.9.
E_UNKNOWNITEMNAME	See description in Section 3.1.9.
E_UNKNOWNITEMPATH	See description in Section 3.1.9.
S_DATAQUEUEOVERFLOW	See description in Section 3.1.9.
E_XXX, S_XXX	Vendor-specific result code.

Faults:

The server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_BUSY	See description in Section 3.1.9.
E_FAIL	See description in Section 3.1.9.
E_INVALIDHOLDTIME	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.

Example:

```
<soap:Body>
  <SubscriptionPolledRefreshResponse
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <SubscriptionPolledRefreshResult
      RcvTime="2003-05-27T06:16:31.8218750-07:00"
      ReplyTime="2003-05-27T06:16:31.9218750-07:00"
      RevisedLocaleID=""
      ServerState="running"
    />
    <RitemList SubscriptionHandle="f6f7900f-3962-4965-abba-31607ce5246b">
      <Items
        ItemName="Analog Types/Int"
        ClientItemHandle="fdce6f30-b8d4-4eeb-becf-6deeacdc7f36"
        Timestamp="2003-05-27T06:16:31.6718750-07:00"
      >
        <Value xsi:type="xsd:int">0</Value>
        <Quality />
      </Items>
      <Items
        ItemName="Analog Types/Double"
        ClientItemHandle="e035d707-e27a-4b06-b103-fea125ce5ca4"
        Timestamp="2003-05-27T06:16:31.6718750-07:00"
      >
        <Value xsi:type="xsd:double">14.644660940672427</Value>
        <Quality />
      </Items>
    </RItemList>
  </SubscriptionPolledRefreshResponse>
</soap:Body>
```

3.7 SubscriptionCancel

3.7.1 SubscriptionCancel

Description

SubscriptionCancel is the container of information that represents the *SubscriptionCancel* request.

The server will cancel a subscription (ServerSubHandle) and allow the server to clean up any resources associated with the subscription. The server will cancel any processing in progress associated with the specified subscription. The ServerSubHandle will also be invalid for any further *SubscriptionPolledRefresh* requests. If the subscription was part of a *SubscriptionPolledRefresh* which specified multiple subscriptions, then only the specified subscription will be cancelled and any others still active will continue to be processed. If the subscription associated with ServerSubHandle was the last subscription still active, then the *SubscriptionPolledRefresh* will return immediately. In all cases, the server will identify the invalid (canceled) ServerSubHandles.

```
<s:element name="SubscriptionCancel">
  <s:complexType>
    <s:attribute name="ServerSubHandle" type="s:string" />
    <s:attribute name="ClientRequestHandle" type="s:string" />
  </s:complexType>
</s:element>
```

Name	Description
ServerSubHandle	An identifier which had been supplied by the Server in the response to the *Subscribe* request.
ClientRequestHandle	An optional attribute supplied by the client that will be returned with the response. In larger and more complex systems it helps the client to associate the replies with the proper requests.

Comments:

Example:

```
<soap:Body>
  <SubscriptionCancel
      ServerSubHandle="67409acb-f926-4106-9d8c-69bb85859ebb"
      xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
    />
</soap:Body>
```

3.7.2 SubscriptionCancelResponse

Description

`SubscriptionCancelResponse` is the container of information that represents the *SubscriptionCancel* response.

```
<s:element name="SubscriptionCancelResponse">
  <s:complexType>
     <s:attribute name="ClientRequestHandle" type="s:string" />
  </s:complexType>
</s:element>
```

Name	Description
`ClientRequestHandle`	If supplied by the client in the request then this value is echoed back in the response.

Comments:

Faults:

The Server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_NOSUBSCRIPTION	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.

Example:

```
<soap:Body>
  <SubscriptionCancelResponse
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
  />
</soap:Body>
```

3.8 Browse

3.8.1 Browse

Description

Browse is the container of information that represents the *Browse* request

```
<s:element name="Browse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="PropertyNames"
        type="s:QName" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="ClientRequestHandle" type="s:string" />
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ItemName" type="s:string" />
    <s:attribute name="ContinuationPoint" type="s:string" />
    <s:attribute default="0" name="MaxElementsReturned" type="s:int" />
    <s:attribute default="all" name="BrowseFilter" type="s0:browseFilter" />
    <s:attribute name="ElementNameFilter" type="s:string" />
    <s:attribute name="VendorFilter" type="s:string" />
    <s:attribute default="false" name="ReturnAllProperties" type="s:boolean"
/>
    <s:attribute default="false" name="ReturnPropertyValues"
type="s:boolean" />
    <s:attribute default="false" name="ReturnErrorText" type="s:boolean" />
  </s:complexType>
</s:element>

<s:simpleType name="browseFilter">
  <s:restriction base="s:string">
    <s:enumeration value="all" />
    <s:enumeration value="branch" />
    <s:enumeration value="item" />
  </s:restriction>
</s:simpleType>
```

Name	Description
PropertyName	A sequence of qualified property names to be returned with each element. If ReturnAllProperties is true, PropertyName is ignored and all properties are returned.
LocaleID	An optional value supplied by the client that specifies the language for certain return data (see section LocaleID, above).
ClientRequestHandle	An optional value supplied by the client that will be returned with the response. In larger and more complex systems it helps the client to associate the replies with the proper requests.
ItemPath	The ItemPath for the starting item. If this is a secondary Browse request, this must be identical to the value supplied in the initial request.

ItemName	The ItemName for the starting item. If this is a secondary Browse request, this must be identical to the value supplied in the initial request.
ContinuationPoint	If this is a secondary Browse request, the BrowseResponse might have returned a Continuation Point where the server can restart the browse operation. This will not be provided in the initial Browse request. This is an Opaque value that the server creates. A Continuation Point will be returned in the response if a Server does support Continuation Point, and the response is larger than MaxItemsReturned. The Continuation Point will allow the Client to resume the Browse request from the previous completion point. When using continuation point, clients must pass the same mask and filter for all subsequent Browse calls. Failing to do so will return error E_INVALIDCONTINUATIONPOINT.
MaxElementsReturned	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 MaxElementsReturned. The server will set MoreElements to True if there are more elements than MaxItemsReturned. If MaxElementsReturned is missing or 0 then there is no client side restriction on the number of returned elements.
BrowseFilter	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 BrowseElement are returned for each value of BrowseFilter.
ElementNameFilter	An expression that is identical to the format as defined in DA 2.0, and DA 3.0 will be used to filter Element names, i.e., the user readable Name field.
VendorFilter	A Vendor specific expression that will be used to filter Vendor specific information. Impact to results of the ElementNameFilter is undefined.
ReturnAllProperties	Server must return all properties which are available for each of the returned elements. If ReturnAllProperties is True, PropertyName is ignored. If ReturnAllProperties is False, or missing, PropertyName will be used.
ReturnPropertyValues	Server must return the property values in addition to the property names.
ReturnErrorText	If TRUE the server will return verbose error description.

Comments:

If no filters are specified, then an ALL is assumed.

The *Browse* service only supports a single level of browsing. If the user wishes to recursively browse a hierachy, then the user will use the returned ItemPaths of elements with children for that purpose.

If the client specifies a null string for ItemPath and ItemName, then the server will do a *Browse* beginning at the top level.

The server will do a *Browse* from the level specified by the combination of ItemPath and ItemName.

The following table describes the relationship between the possible values of HasChildren and IsItem supplied in the response.

BrowseElement in Response		Description	BrowseFilter in Request		
IsItem	**HasChildren**		**All**	**Branch**	**Item**
false	false	An empty branch	●	●	
false	true	A branch that has children, or possibly has children.	●	●	
true	false	An item that is not a branch	●		●
true	true	A branch that has children, or possibly has children, that is also an item	●	●	●

Example:

```
<soap:Body>
  <Browse
    ClientRequestHandle=""
    ItemName="Enumerated Types"
    ReturnAllProperties="true"
    ReturnPropertyValues="true"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
  >
    <PropertyNames>accessRights</PropertyNames>
    <PropertyNames>euType</PropertyNames>
    <PropertyNames>euInfo</PropertyNames>
  </Browse>
</soap:Body>
```

3.8.2 BrowseResponse

Description

BrowseResponse is the container of information that represents the *Browse* response.

```
<s:element name="BrowseResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
          name="BrowseResult"
          type="s0:ReplyBase" />
      <s:element minOccurs="0" maxOccurs="unbounded"
          name="Elements"
          type="s0:BrowseElement" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="Errors"
        type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="ContinuationPoint" type="s:string" />
    <s:attribute default="false" name="MoreElements" type="s:boolean" />
  </s:complexType>
</s:element>

<s:complexType name="BrowseElement">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Properties"
      type="s0:ItemProperty" />
  </s:sequence>
  <s:attribute name="Name" type="s:string" />
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="IsItem" type="s:boolean" use="required" />
  <s:attribute name="HasChildren" type="s:boolean" use="required" />
</s:complexType>
```

Name	Description
BrowseResult	For a detailed description of ReplyBase see the separate section, above. Required Element.
ContinuationPoint	If this is a secondary *Browse* request, the *BrowseResponse* might have returned a Continuation Point where the *Browse* request the server can restart the browse operation. This is an Opaque value that the server creates. The Server may support a Continuation Point. A Continuation Point is desirable for requests that are larger than MaxItemsReturned. When using continuation point, clients must pass the same mask and filter for all subsequent Browse calls. Failing to do so will return error E_INVALIDCONTINUATIONPOINT.
MoreElements	The server will set MoreElements to True if there are more elements than MaxItemsReturned.

	This attribute is always returned.
Errors	An embedded Error structure associated with the Properties of the elements.
Elements	An arbitrary number of elements which match the request.

BrowseElement:

Name	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.
ItemPath	ItemPath and ItemName together uniquely identify this element in the server's browse space. They are used together in subsequent calls to *Browse*, *Read*, *Write*, *Subscribe*, and *GetProperties*. If ItemPath is empty, then ItemName by itself is a fully qualified name that uniquely identifies this element. In general, the client should use ItemPath and ItemName as-is for subsequent calls to services.
ItemName	See ItemPath
IsItem	If IsItem is set then the element is an item that can be used to *Read*, *Write*, and *Subscribe*. If ItemPath and ItemName are missing and IsItem is True then this element is a "hint" versus being a valid item. Refer to the OPC DA Custom Specification for detail on items, or hints.
HasChildren	If HasChidren is set, 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.
Properties	An array of ItemProperty elements as requested.

Comments:

HasChildren and IsItem are useful for a UI presentation of the hierarchy using a tree control without doing "browse ahead". The following truth table indicates the desired UI representation:

HasChildren	IsItem	UI Representation
false	false	An empty folder icon with no expand (+) symbol
false	true	An icon indicating an item
true	false	A folder icon with the expand (+) symbol
true	true	A folder/item icon with the expand (+) symbol

If the level specified by the combination of ItemPath and ItemName is valid, but does not have any children, then the *Browse* will succeed, but the result will be an empty result.

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

Abnormal Result Codes:

One of the following codes can be part of any of the property elements.

E_INVALIDPID	See description in Section 3.1.9.
E_WRITEONLY	See description in Section 3.1.9.
E_XXX, S_XXX	Vendor-specific result code.

Faults:

The server should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_INVALIDCONTINUATIONPOINT	See description in Section 3.1.9.
E_INVALIDFILTER	See description in Section 3.1.9.
E_INVALIDITEMNAME	See description in Section 3.1.9.
E_INVALIDITEMPATH	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.
E_UNKNOWNITEMNAME	See description in Section 3.1.9.
E_UNKNOWNITEMPATH	See description in Section 3.1.9.

Example:

```
<soap:Body>
  <BrowseResponse xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <BrowseResult
      RcvTime="2003-05-27T07:19:27.4625000-07:00"
      ReplyTime="2003-05-27T07:19:27.5625000-07:00"
      ClientRequestHandle=""
      ServerState="running"
    />
      <Elements
        Name="Fellowship"
        ItemName="Enumerated Types/Fellowship"
        IsItem="true"
        HasChildren="false"
      >
        <Properties Name="accessRights" Description="Item Access Rights">
          <Value xsi:type="xsd:string">readWritable</Value>
        </Properties>
      </Elements>
      <Elements
        Name="Gems"
```

```
        ItemName="Enumerated Types/Gems"
        IsItem="true"
        HasChildren="false"
      >
      <Properties Name="accessRights" Description="Item Access Rights">
        <Value xsi:type="xsd:string">readWritable</Value>
      </Properties>
    </Elements>
  </BrowseResponse>
</soap:Body>
```

3.9 GetProperties

3.9.1 GetProperties

Description

GetProperties is the container of information that represents the *GetProperties* request.

```
<s:element name="GetProperties">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="ItemIDs"
        type="s0:ItemIdentifier" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="PropertyNames"
        type="s:QName" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="ClientRequestHandle" type="s:string" />
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute default="false" name="ReturnAllProperties" type="s:boolean"
/>
    <s:attribute default="false" name="ReturnPropertyValues"
type="s:boolean" />
    <s:attribute default="false" name="ReturnErrorText" type="s:boolean" />
  </s:complexType>
</s:element>

<s:complexType name="ItemIdentifier">
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ItemName" type="s:string" />
</s:complexType>
```

Name	Description
ItemIDs	A list of Items for which to get properties. ItemPath and ItemName together uniquely identify this element in the server's browse space. If ItemPath is empty, then ItemName by itself is a fully qualified name that uniquely identifies this element.
PropertyNames	A sequence of qualified property names to be returned. If ReturnAllProperties is true, PropertyName is ignored and all properties are returned.
LocaleID	An optional value supplied by the client that specifies the language for certain return data (see section LocaleID, above).
ClientRequestHandle	An optional value supplied by the client that will be returned with the response. In larger and more complex systems it helps the client to associate the replies with the proper requests.
ItemPath	ItemPath is a hierarchical parameter. It can be overridden by the ItemPath in the ItemIdentifier.

	If **ItemPath** is Blank or missing at all levels of the hierarchy, then the ItemName is expected to be a fully qualified name.
ReturnAllProperties	Server must return all properties which are available. If ReturnAllProperties is true, PropertyName is ignored.
ReturnPropertyValues	Server must return the property values in addition to the property names.
ReturnErrorText	If TRUE the server will return verbose error description.

Comments:

Example:

```
<soap:Body>
  <GetProperties
    ReturnPropertyValues="true"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
  >
    <ItemIDs ItemName="Enumerated Types/Fellowship" />
    <PropertyNames>accessRights</PropertyNames>
    <PropertyNames>euType</PropertyNames>
  </GetProperties>
</soap:Body>
```

3.9.2 GetPropertiesResponse

Description

GetPropertiesResponse is the container of information that represents the *GetProperties* response.

```
<s:element name="GetPropertiesResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="GetPropertiesResult"
        type="s0:ReplyBase" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="PropertyLists"
        type="s0:PropertyReplyList" />
      <s:element minOccurs="0" maxOccurs="unbounded"
        name="Errors"
        type="s0:OPCError" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="PropertyReplyList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Properties"
      type="s0:ItemProperty" />
  </s:sequence>
  <s:attribute name="ItemPath" type="s:string" />
  <s:attribute name="ItemName" type="s:string" />
  <s:attribute name="ResultID" type="s:QName" />
</s:complexType>
```

Name	Description
GetPropertiesResult	For a detailed description of ReplyBase see the separate section, above. Required Element.
PropertyList	One of these elements is returned for each requested item. ItemName and ItemPath are returned for convenience. If unknown or invalid an error is returned in ResultID. Otherwise, Property contains the list of requested properties. For a detailed description of ItemProperty see the separate section above.
Errors	An array of Errors that is appropriate for this Response.

Abnormal Result Codes:

One of the following codes can be part of any of the property elements.

E_FAIL	See description in Section 3.1.9.

E_INVALIDITEMNAME	See description in Section 3.1.9.
E_INVALIDITEMPATH	See description in Section 3.1.9.
E_INVALIDPID	See description in Section 3.1.9.
E_UNKNOWNITEMPATH	See description in Section 3.1.9.
E_UNKNOWNITEMNAME	See description in Section 3.1.9.
E_WRITEONLY	See description in Section 3.1.9.
E_XXX, S_XXX	Vendor-specific result code.

Faults:

The sServer should use the following fault codes. Additional faults may occur due to protocol or parsing errors.

E_FAIL	See description in Section 3.1.9.
E_OUTOFMEMORY	See description in Section 3.1.9.
E_SERVERSTATE	See description in Section 3.1.9.
E_TIMEDOUT	See description in Section 3.1.9.

Example:

```
<soap:Body>
  <GetPropertiesResponse
xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
    <GetPropertiesResult
      RcvTime="2003-05-27T07:29:33.1875000-07:00"
      ReplyTime="2003-05-27T07:39:33.1875000-07:00"
      ServerState="running"
    />
      <PropertyLists>
        <Properties Name="accessRights" Description="Item Access Rights">
          <Value xsi:type="xsd:string">readWritable</Value>
        </Properties>
        <Properties Name="euType" Description="Item EU Type">
          <Value xsi:type="xsd:string">enumerated</Value>
        </Properties>
      </PropertyLists>
  </GetPropertiesResponse>
</soap:Body>
```

4. Transports

The OPC-XML-DA specification defines a set of services by defining the request and response messages with the syntax defined by the SOAP specification. For purposes of compliance testing and multi-vendor interoperability, this specification requires that clients and servers use HTTP as a means to transport these messages.

That said, vendors might choose to implement systems that use the OPC-XML-DA message formats but use transport protocols other than HTTP (such as SMTP).

5. Appendix A - Patent Issues

As of the time of publication, the OPC Foundation has been made aware of four patents that may be relevant to implementers using XML DA components. All four patents were the subject of a suit between Schneider Automation and Opto 22, Inc. alleging patent infringement.

1. U.S. Patent 5,805,442 (Crater et al.) (the '442 Patent) and
   U.S. Patent 5,975,737 (Crater et al.) (the '737 Patent)

Based on a review of the claims of the '442 and '737 Patents, OPC believes that a system would not necessarily infringe any of the claims of either the '442 Patent or the '737 Patent, if the system lacks a controller or control system having instructions associated with data gathered by the controller or control system, where the instructions are retrievable and executable by a computer and cause the computer to present the data in a predetermined format. The XML DA specification does not include this function as a requirement or as an option. Accordingly, our view is that the proposed XML DA protocol can be implemented without necessarily infringing the '442 Patent or the '737 Patent.

2. U.S. Patent 6,061,603 (Papadopolous et al.) (The '603 Patent) and
   U.S. Patent 6,282,454 (Papadopolous et al.) (The '454 Patent)

These patents may be relevant to those implementing an interface module that connects to the backplane of a programmable logic controller (PLC) for coupling the PLC to a network to allow access to the PLC using a web browser, particularly for interface module implementations including a microprocessor with a real-time operating system. Systems that do not include an interface module for coupling the backplane of a PLC to a network would not necessarily infringe any of the claims of either the '603 Patent or the '454 Patent. The XML DA specification does not include this function as a requirement or as an option. Accordingly, our view is that the proposed XML DA protocol can be used without necessarily infringing the '603 Patent or the '454 Patent.

Parties considering use of XML DA must be aware that some specific implementations, or features added to otherwise non-infringing implementations, may raise an issue of infringement with respect to these patents or to some other patent. In particular, incorporation of XML DA components into an otherwise infringing system cannot be relied on to cure the otherwise infringing system.

This statement should not be relied upon by any party as an opinion or guarantee that any implementation it might make or use would not infringe the '442, '737, '603, or '454 Patents or any other patents. Moreover, Schneider might disagree with the above interpretations of the claims of these patents.

The OPC Foundation does not indemnify the members or non-members using specifications or sample code provided by the OPC Foundation.

The OPC Foundation does not guarantee that using the OPC Foundation components prevents users from infringing on any patented technology whatsoever. The vendors and end-users are encouraged to validate that the products and systems that are constructed do not infringe on patented technology. OPC Foundation specifically disclaims any liability for any infringement by members or non-members.

6. Appendix B - Formal Schemas (WSDL)

This section contains the complete WSDL for the OPC-XML-DA WebService.

NOTE:

The WSDL shown in the document is a snapshot of the latest available WSDL.

Implementers must use the published WSDL when building web applications that comply with this specification.

The published WSDL is available at this URL: http://opcfoundation.org/webservices/XMLDA/1.0/

For many of the requests and responses in the WSDL that follows, "minOccurs=0" is used. However, the actual minumum number required by the request or response is specified above for each request and response.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
      COPYRIGHT (c) 2003 OPC Foundation. All rights reserved.
      http://www.opcfoundation.org
      Use subject to the OPC Foundation License Agreement found at the following URL:
      http://www.opcfoundation.org/Downloads/LicenseAgreement.asp
-->
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://opcfoundation.org/webservices/XMLDA/1.0/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://opcfoundation.org/webservices/XMLDA/1.0/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://opcfoundation.org/webservices/XMLDA/1.0/">
      <s:element name="GetStatus">
        <s:complexType>
          <s:attribute name="LocaleID" type="s:string" />
          <s:attribute name="ClientRequestHandle" type="s:string" />
        </s:complexType>
      </s:element>
```

```
      <s:element name="GetStatusResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetStatusResult" type="s0:ReplyBase" />
            <s:element minOccurs="0" maxOccurs="1" name="Status" type="s0:ServerStatus" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ReplyBase">
        <s:attribute name="RcvTime" type="s:dateTime" use="required" />
        <s:attribute name="ReplyTime" type="s:dateTime" use="required" />
        <s:attribute name="ClientRequestHandle" type="s:string" />
        <s:attribute name="RevisedLocaleID" type="s:string" />
        <s:attribute name="ServerState" type="s0:serverState" use="required" />
      </s:complexType>
      <s:simpleType name="serverState">
        <s:restriction base="s:string">
          <s:enumeration value="running" />
          <s:enumeration value="failed" />
          <s:enumeration value="noConfig" />
          <s:enumeration value="suspended" />
          <s:enumeration value="test" />
          <s:enumeration value="commFault" />
        </s:restriction>
      </s:simpleType>
      <s:complexType name="ServerStatus">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="StatusInfo" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="VendorInfo" type="s:string" />
          <s:element minOccurs="0" maxOccurs="unbounded" name="SupportedLocaleIDs" type="s:string" />
          <s:element minOccurs="0" maxOccurs="unbounded" name="SupportedInterfaceVersions"
type="s0:interfaceVersion" />
        </s:sequence>
        <s:attribute name="StartTime" type="s:dateTime" use="required" />
        <s:attribute name="ProductVersion" type="s:string" />
      </s:complexType>
      <s:simpleType name="interfaceVersion">
        <s:restriction base="s:string">
```

```
      <s:enumeration value="XML_DA_Version_1_0" />
    </s:restriction>
  </s:simpleType>
  <s:element name="Read">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Options" type="s0:RequestOptions" />
        <s:element minOccurs="0" maxOccurs="1" name="ItemList" type="s0:ReadRequestItemList" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:complexType name="RequestOptions">
    <s:attribute default="true" name="ReturnErrorText" type="s:boolean" />
    <s:attribute default="false" name="ReturnDiagnosticInfo" type="s:boolean" />
    <s:attribute default="false" name="ReturnItemTime" type="s:boolean" />
    <s:attribute default="false" name="ReturnItemPath" type="s:boolean" />
    <s:attribute default="false" name="ReturnItemName" type="s:boolean" />
    <s:attribute name="RequestDeadline" type="s:dateTime" />
    <s:attribute name="ClientRequestHandle" type="s:string" />
    <s:attribute name="LocaleID" type="s:string" />
  </s:complexType>
  <s:complexType name="ReadRequestItemList">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:ReadRequestItem" />
    </s:sequence>
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ReqType" type="s:QName" />
    <s:attribute name="MaxAge" type="s:int" />
  </s:complexType>
  <s:complexType name="ReadRequestItem">
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ReqType" type="s:QName" />
    <s:attribute name="ItemName" type="s:string" />
    <s:attribute name="ClientItemHandle" type="s:string" />
    <s:attribute name="MaxAge" type="s:int" />
  </s:complexType>
  <s:element name="ReadResponse">
    <s:complexType>
```

```
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="ReadResult" type="s0:ReplyBase" />
        <s:element minOccurs="0" maxOccurs="1" name="RItemList" type="s0:ReplyItemList" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:complexType name="ReplyItemList">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:ItemValue" />
    </s:sequence>
    <s:attribute name="Reserved" type="s:string" />
  </s:complexType>
  <s:complexType name="ItemValue">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="DiagnosticInfo" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="Value" />
      <s:element minOccurs="0" maxOccurs="1" name="Quality" type="s0:OPCQuality" />
    </s:sequence>
    <s:attribute name="ValueTypeQualifier" type="s:QName" />
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ItemName" type="s:string" />
    <s:attribute name="ClientItemHandle" type="s:string" />
    <s:attribute name="Timestamp" type="s:dateTime" />
    <s:attribute name="ResultID" type="s:QName" />
  </s:complexType>
  <s:complexType name="OPCQuality">
    <s:attribute default="good" name="QualityField" type="s0:qualityBits" />
    <s:attribute default="none" name="LimitField" type="s0:limitBits" />
    <s:attribute default="0" name="VendorField" type="s:unsignedByte" />
  </s:complexType>
  <s:simpleType name="qualityBits">
    <s:restriction base="s:string">
      <s:enumeration value="bad" />
      <s:enumeration value="badConfigurationError" />
      <s:enumeration value="badNotConnected" />
      <s:enumeration value="badDeviceFailure" />
      <s:enumeration value="badSensorFailure" />
```

```
      <s:enumeration value="badLastKnownValue" />
      <s:enumeration value="badCommFailure" />
      <s:enumeration value="badOutOfService" />
      <s:enumeration value="badWaitingForInitialData" />
      <s:enumeration value="uncertain" />
      <s:enumeration value="uncertainLastUsableValue" />
      <s:enumeration value="uncertainSensorNotAccurate" />
      <s:enumeration value="uncertainEUExceeded" />
      <s:enumeration value="uncertainSubNormal" />
      <s:enumeration value="good" />
      <s:enumeration value="goodLocalOverride" />
    </s:restriction>
  </s:simpleType>
  <s:simpleType name="limitBits">
    <s:restriction base="s:string">
      <s:enumeration value="none" />
      <s:enumeration value="low" />
      <s:enumeration value="high" />
      <s:enumeration value="constant" />
    </s:restriction>
  </s:simpleType>
  <s:complexType name="OPCError">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Text" type="s:string" />
    </s:sequence>
    <s:attribute name="ID" type="s:QName" use="required" />
  </s:complexType>
  <s:complexType name="ArrayOfFloat">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="float" type="s:float" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfInt">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="int" type="s:int" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfUnsignedInt">
```

```
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="unsignedInt" type="s:unsignedInt" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfLong">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="long" type="s:long" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfUnsignedLong">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="unsignedLong" type="s:unsignedLong" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfDouble">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="double" type="s:double" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfUnsignedShort">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="unsignedShort" type="s:unsignedShort" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfBoolean">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="boolean" type="s:boolean" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfString">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="string" nillable="true" type="s:string" />
    </s:sequence>
  </s:complexType>
  <s:complexType name="ArrayOfDateTime">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="dateTime" type="s:dateTime" />
    </s:sequence>
```

```
</s:complexType>
<s:complexType name="ArrayOfAnyType">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="anyType" nillable="true" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfDecimal">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="decimal" type="s:decimal" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfByte">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="byte" type="s:byte" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfShort">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="short" type="s:short" />
  </s:sequence>
</s:complexType>
<s:element name="Write">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Options" type="s0:RequestOptions" />
      <s:element minOccurs="0" maxOccurs="1" name="ItemList" type="s0:WriteRequestItemList" />
    </s:sequence>
    <s:attribute name="ReturnValuesOnReply" type="s:boolean" use="required" />
  </s:complexType>
</s:element>
<s:complexType name="WriteRequestItemList">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:ItemValue" />
  </s:sequence>
  <s:attribute name="ItemPath" type="s:string" />
</s:complexType>
<s:element name="WriteResponse">
  <s:complexType>
```

```
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="WriteResult" type="s0:ReplyBase" />
        <s:element minOccurs="0" maxOccurs="1" name="RItemList" type="s0:ReplyItemList" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="Subscribe">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Options" type="s0:RequestOptions" />
        <s:element minOccurs="0" maxOccurs="1" name="ItemList" type="s0:SubscribeRequestItemList" />
      </s:sequence>
      <s:attribute name="ReturnValuesOnReply" type="s:boolean" use="required" />
      <s:attribute default="0" name="SubscriptionPingRate" type="s:int" />
    </s:complexType>
  </s:element>
  <s:complexType name="SubscribeRequestItemList">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:SubscribeRequestItem" />
    </s:sequence>
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ReqType" type="s:QName" />
    <s:attribute name="Deadband" type="s:float" />
    <s:attribute name="RequestedSamplingRate" type="s:int" />
    <s:attribute name="EnableBuffering" type="s:boolean" />
  </s:complexType>
  <s:complexType name="SubscribeRequestItem">
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ReqType" type="s:QName" />
    <s:attribute name="ItemName" type="s:string" />
    <s:attribute name="ClientItemHandle" type="s:string" />
    <s:attribute name="Deadband" type="s:float" />
    <s:attribute name="RequestedSamplingRate" type="s:int" />
    <s:attribute name="EnableBuffering" type="s:boolean" />
  </s:complexType>
  <s:complexType name="SubscribeReplyItemList">
    <s:sequence>
```

```
      <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:SubscribeItemValue" />
    </s:sequence>
    <s:attribute name="RevisedSamplingRate" type="s:int" />
  </s:complexType>
  <s:complexType name="SubscribeItemValue">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="ItemValue" type="s0:ItemValue" />
    </s:sequence>
    <s:attribute name="RevisedSamplingRate" type="s:int" />
  </s:complexType>
  <s:element name="SubscribeResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="SubscribeResult" type="s0:ReplyBase" />
        <s:element minOccurs="0" maxOccurs="1" name="RItemList" type="s0:SubscribeReplyItemList" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
      </s:sequence>
      <s:attribute name="ServerSubHandle" type="s:string" />
    </s:complexType>
  </s:element>
  <s:element name="SubscriptionPolledRefresh">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Options" type="s0:RequestOptions" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="ServerSubHandles" type="s:string" />
      </s:sequence>
      <s:attribute name="HoldTime" type="s:dateTime" />
      <s:attribute default="0" name="WaitTime" type="s:int" />
      <s:attribute default="false" name="ReturnAllItems" type="s:boolean" />
    </s:complexType>
  </s:element>
  <s:complexType name="SubscribePolledRefreshReplyItemList">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Items" type="s0:ItemValue" />
    </s:sequence>
    <s:attribute name="SubscriptionHandle" type="s:string" />
  </s:complexType>
  <s:element name="SubscriptionPolledRefreshResponse">
```

```
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="SubscriptionPolledRefreshResult"
type="s0:ReplyBase" />
            <s:element minOccurs="0" maxOccurs="unbounded" name="InvalidServerSubHandles" type="s:string"
/>
            <s:element minOccurs="0" maxOccurs="unbounded" name="RItemList"
type="s0:SubscribePolledRefreshReplyItemList" />
            <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
          </s:sequence>
          <s:attribute default="false" name="DataBufferOverflow" type="s:boolean" />
        </s:complexType>
      </s:element>
      <s:element name="SubscriptionCancel">
        <s:complexType>
          <s:attribute name="ServerSubHandle" type="s:string" />
          <s:attribute name="ClientRequestHandle" type="s:string" />
        </s:complexType>
      </s:element>
      <s:element name="SubscriptionCancelResponse">
        <s:complexType>
          <s:attribute name="ClientRequestHandle" type="s:string" />
        </s:complexType>
      </s:element>
      <s:element name="Browse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="unbounded" name="PropertyNames" type="s:QName" />
          </s:sequence>
          <s:attribute name="LocaleID" type="s:string" />
          <s:attribute name="ClientRequestHandle" type="s:string" />
          <s:attribute name="ItemPath" type="s:string" />
          <s:attribute name="ItemName" type="s:string" />
          <s:attribute name="ContinuationPoint" type="s:string" />
          <s:attribute default="0" name="MaxElementsReturned" type="s:int" />
          <s:attribute default="all" name="BrowseFilter" type="s0:browseFilter" />
          <s:attribute name="ElementNameFilter" type="s:string" />
          <s:attribute name="VendorFilter" type="s:string" />
```

```
      <s:attribute default="false" name="ReturnAllProperties" type="s:boolean" />
      <s:attribute default="false" name="ReturnPropertyValues" type="s:boolean" />
      <s:attribute default="false" name="ReturnErrorText" type="s:boolean" />
    </s:complexType>
  </s:element>
  <s:simpleType name="browseFilter">
    <s:restriction base="s:string">
      <s:enumeration value="all" />
      <s:enumeration value="branch" />
      <s:enumeration value="item" />
    </s:restriction>
  </s:simpleType>
  <s:complexType name="BrowseElement">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Properties" type="s0:ItemProperty" />
    </s:sequence>
    <s:attribute name="Name" type="s:string" />
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ItemName" type="s:string" />
    <s:attribute name="IsItem" type="s:boolean" use="required" />
    <s:attribute name="HasChildren" type="s:boolean" use="required" />
  </s:complexType>
  <s:complexType name="ItemProperty">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Value" />
    </s:sequence>
    <s:attribute name="Name" type="s:QName" use="required" />
    <s:attribute name="Description" type="s:string" />
    <s:attribute name="ItemPath" type="s:string" />
    <s:attribute name="ItemName" type="s:string" />
    <s:attribute name="ResultID" type="s:QName" />
  </s:complexType>
  <s:element name="BrowseResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="BrowseResult" type="s0:ReplyBase" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="Elements" type="s0:BrowseElement" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
```

```
        </s:sequence>
        <s:attribute name="ContinuationPoint" type="s:string" />
        <s:attribute default="false" name="MoreElements" type="s:boolean" />
      </s:complexType>
    </s:element>
    <s:element name="GetProperties">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="ItemIDs" type="s0:ItemIdentifier" />
          <s:element minOccurs="0" maxOccurs="unbounded" name="PropertyNames" type="s:QName" />
        </s:sequence>
        <s:attribute name="LocaleID" type="s:string" />
        <s:attribute name="ClientRequestHandle" type="s:string" />
        <s:attribute name="ItemPath" type="s:string" />
        <s:attribute default="false" name="ReturnAllProperties" type="s:boolean" />
        <s:attribute default="false" name="ReturnPropertyValues" type="s:boolean" />
        <s:attribute default="false" name="ReturnErrorText" type="s:boolean" />
      </s:complexType>
    </s:element>
    <s:complexType name="ItemIdentifier">
      <s:attribute name="ItemPath" type="s:string" />
      <s:attribute name="ItemName" type="s:string" />
    </s:complexType>
    <s:complexType name="PropertyReplyList">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="Properties" type="s0:ItemProperty" />
      </s:sequence>
      <s:attribute name="ItemPath" type="s:string" />
      <s:attribute name="ItemName" type="s:string" />
      <s:attribute name="ResultID" type="s:QName" />
    </s:complexType>
    <s:element name="GetPropertiesResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetPropertiesResult" type="s0:ReplyBase" />
          <s:element minOccurs="0" maxOccurs="unbounded" name="PropertyLists" type="s0:PropertyReplyList"
/>
          <s:element minOccurs="0" maxOccurs="unbounded" name="Errors" type="s0:OPCError" />
```

```
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</types>
<message name="GetStatusSoapIn">
  <part name="parameters" element="s0:GetStatus" />
</message>
<message name="GetStatusSoapOut">
  <part name="parameters" element="s0:GetStatusResponse" />
</message>
<message name="ReadSoapIn">
  <part name="parameters" element="s0:Read" />
</message>
<message name="ReadSoapOut">
  <part name="parameters" element="s0:ReadResponse" />
</message>
<message name="WriteSoapIn">
  <part name="parameters" element="s0:Write" />
</message>
<message name="WriteSoapOut">
  <part name="parameters" element="s0:WriteResponse" />
</message>
<message name="SubscribeSoapIn">
  <part name="parameters" element="s0:Subscribe" />
</message>
<message name="SubscribeSoapOut">
  <part name="parameters" element="s0:SubscribeResponse" />
</message>
<message name="SubscriptionPolledRefreshSoapIn">
  <part name="parameters" element="s0:SubscriptionPolledRefresh" />
</message>
<message name="SubscriptionPolledRefreshSoapOut">
  <part name="parameters" element="s0:SubscriptionPolledRefreshResponse" />
</message>
<message name="SubscriptionCancelSoapIn">
  <part name="parameters" element="s0:SubscriptionCancel" />
</message>
```

```
<message name="SubscriptionCancelSoapOut">
  <part name="parameters" element="s0:SubscriptionCancelResponse" />
</message>
<message name="BrowseSoapIn">
  <part name="parameters" element="s0:Browse" />
</message>
<message name="BrowseSoapOut">
  <part name="parameters" element="s0:BrowseResponse" />
</message>
<message name="GetPropertiesSoapIn">
  <part name="parameters" element="s0:GetProperties" />
</message>
<message name="GetPropertiesSoapOut">
  <part name="parameters" element="s0:GetPropertiesResponse" />
</message>
<portType name="Service">
  <operation name="GetStatus">
    <input message="s0:GetStatusSoapIn" />
    <output message="s0:GetStatusSoapOut" />
  </operation>
  <operation name="Read">
    <input message="s0:ReadSoapIn" />
    <output message="s0:ReadSoapOut" />
  </operation>
  <operation name="Write">
    <input message="s0:WriteSoapIn" />
    <output message="s0:WriteSoapOut" />
  </operation>
  <operation name="Subscribe">
    <input message="s0:SubscribeSoapIn" />
    <output message="s0:SubscribeSoapOut" />
  </operation>
  <operation name="SubscriptionPolledRefresh">
    <input message="s0:SubscriptionPolledRefreshSoapIn" />
    <output message="s0:SubscriptionPolledRefreshSoapOut" />
  </operation>
  <operation name="SubscriptionCancel">
    <input message="s0:SubscriptionCancelSoapIn" />
```

```
      <output message="s0:SubscriptionCancelSoapOut" />
    </operation>
    <operation name="Browse">
      <input message="s0:BrowseSoapIn" />
      <output message="s0:BrowseSoapOut" />
    </operation>
    <operation name="GetProperties">
      <input message="s0:GetPropertiesSoapIn" />
      <output message="s0:GetPropertiesSoapOut" />
    </operation>
  </portType>
  <binding name="Service" type="s0:Service">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="GetStatus">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/GetStatus"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="Read">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/Read" style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="Write">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/Write" style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
```

```
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="Subscribe">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/Subscribe"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="SubscriptionPolledRefresh">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/SubscriptionPolledRefresh"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="SubscriptionCancel">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/SubscriptionCancel"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="Browse">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/Browse" style="document"
/>
      <input>
        <soap:body use="literal" />
```

```
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="GetProperties">
      <soap:operation soapAction="http://opcfoundation.org/webservices/XMLDA/1.0/GetProperties"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
</definitions>
```