OPC Data eXchange Specification

Version 1.0

Release

March 5, 2003

Specification Type	Industry Standard Specification		
Title:	**OPC Data eXchange Specification**	Date:	**March 5, 2003**
Version:	1.0	Soft	MS-Word
		Source:	OPC DX 1.00 Specification.doc
Author:	OPC Foundation	Status:	**Release**

Synopsis:

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

Table of Contents

1. Introduction

1.1 Background

The OPC Foundation has defined interfaces to OPC Data Access Servers, Alarm and Event Servers, Batch Servers, and Historical Data Access Servers. These servers acquire data from the plant floor and move it vertically into the enterprise system.

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

1.2 Purpose

The purpose of this document is to continue OPC's goal of enabling and promoting interoperability of applications.

This document defines abstract services used to configure horizontal data transfers from servers with OPC DA interfaces to OPC DX servers. This document contains appendices that map these abstract services to specific interface technologies (e.g. Web Services and COM).

OPC DX does not specify a new method for these data transfers. Instead, it relies on OPC Data Access (OPC DA) data transfer capabilities already in use today. This document defines the behavior of the OPC DX server as it relates to control and monitoring the data transfer from DA servers to itself.

1.3 Target Audience

This specification is intended for developers of OPC compliant DX Clients and Servers. It is assumed that the reader is familiar with Microsoft COM, Web Services, XML, and SOAP. It is also recommended that the reader review the reference documents described in Section 1.5.

1.4 Goals

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

To support these capabilities, OPC DX has the following goals:

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

Insulate the design from communication technologies, while maintaining interoperability within as many products as possible.

1.5 Prerequisites

Readers are assumed to be familiar with the following applicable OPC Specifications.

- OPC Data Access Custom Interface Standard 2.04
- OPC Data Access Custom Interface Standard 2.05
- OPC Data Access Custom Interface Standard 3.0
- OPC Security Custom Interface 1.0
- OPC Common Definitions and Interfaces 1.0
- OPC XML DA Specification 1.0

The specifications are available from the OPC Foundation web site:

http://www.opcfoundation.org/

It is also assumed that the reader is familiar with Microsoft COM technology, Web Services, XML/SOAP. Information regarding XML, SOAP, Web Services and various DXConnections to related sites, white papers, specs, etc, can be found at the W3C Web site.

1.6 Revision History

2. Fundamental Concepts

Before the definition of OPC, there were multiple, incompatible standards for transferring data to and from automation devices. These standards were typically defined for I/O and bus technologies optimized for different environments. Client applications needing to access data from a variety of plant floor devices usually were required to implement and maintain multiple interfaces. Transferring data between devices of different technologies also usually required the use of specialty gateway products.

OPC DA improved the situation by defining a standard server architecture in which servers masked the details of the underlying technologies by providing a uniform device data interface to clients. OPC DA thereby provided a standard for vertical data integration between client applications and plant floor devices. Figure 1 illustrates this concept.

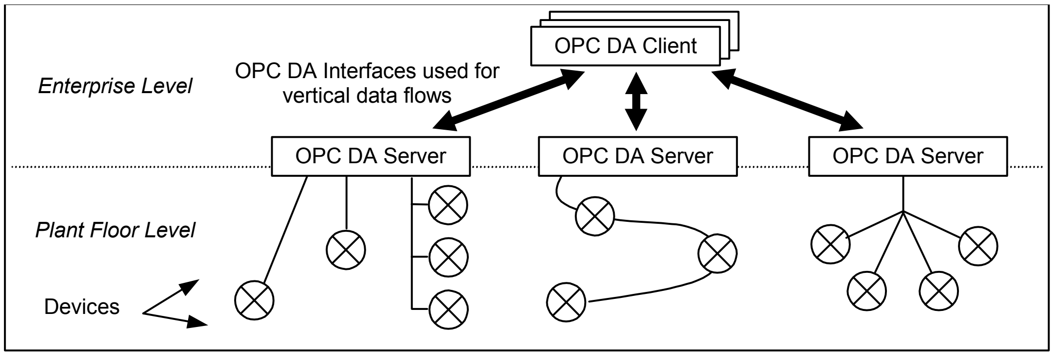

Figure 1 - Vertical OPC DA Integration

However, OPC DA did not address data transfers between servers. To do this, some OPC DA vendors developed intermediary products to read data from one OPC DA server and forward it to another. These products did not provide a standard configuration interface that allowed them to be accessed and configured from standard configuration clients. Instead, each provided proprietary interfaces for configuration.

OPC DX was developed to address data transfers directly between servers, without a store-and-forward intermediary, and with a standard configuration interface. To accomplish this, the OPC DX server was defined as an OPC DA server that has been enhanced to reliably transfer data from a source OPC server to itself. In addition, OPC DX interfaces were defined to allow clients to configure the OPC DX server for these horizontal data transfers. Figure 2 below illustrates this concept.

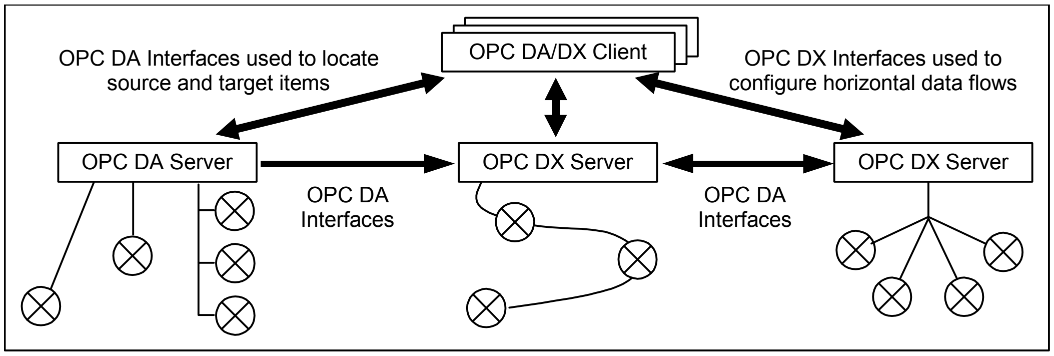
OPC DA Interfaces used to locate source and target items
OPC DA/DX Client
OPC DX Interfaces used to configure horizontal data flows
OPC DA Server
OPC DX Server
OPC DX Server
OPC DA Interfaces
OPC DA Interfaces

Figure 2 - Horizontal OPC DX Integration

3. Architectural Overview

3.1 System Model

OPC DX provides for non-time critical, horizontal transfers of data from a source OPC DA or DX server to a target OPC DX server. The source server contains the source DA item and the target contains the target DA item.

OPC DX defines the data transfer from a source item to a target item as a DXConnection. A DXConnection is represented as a read-only OPC DA item. DXConnections are configured by OPC DX clients using OPC DX specific services.

OPC DX defines related OPC DA items that clients can use to control and monitor DXConnections using OPC DA interfaces. Figure 3 illustrates the OPC DX architecture. A discussion of it follows.

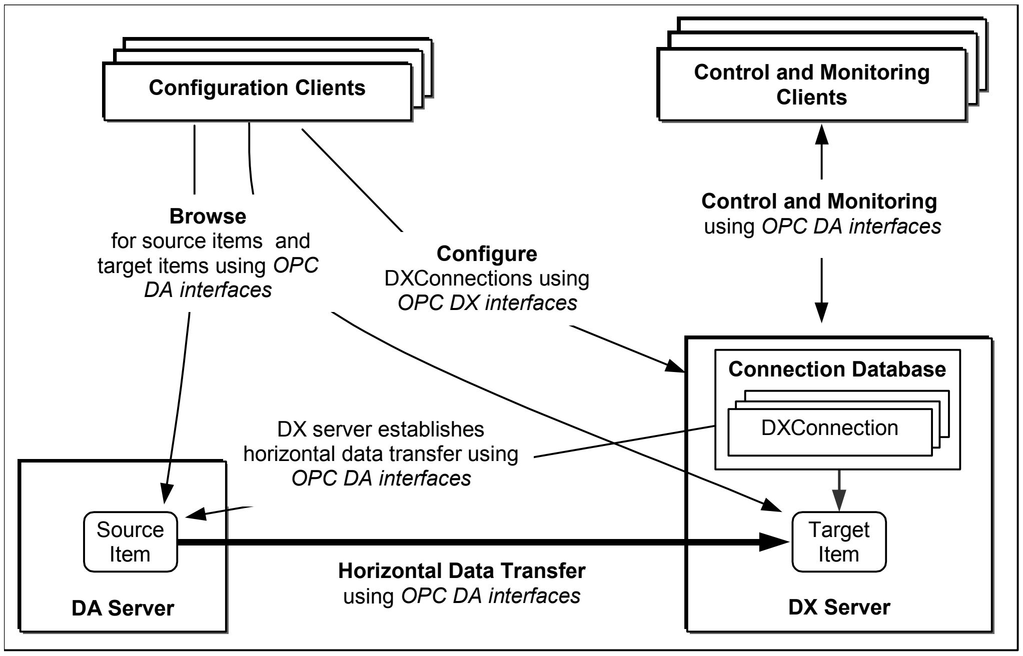
Configuration Clients
Control and Monitoring Clients
Browse for source items and target items using OPC DA interfaces
Configure DXConnections using OPC DX interfaces
Control and Monitoring using OPC DA interfaces
Connection Database
DXConnection
DX server establishes horizontal data transfer using OPC DA interfaces
Source Item
DA Server
Horizontal Data Transfer using OPC DA interfaces
Target Item
DX Server

Figure 3 - DX Interface Architecture

Browsing Source Items, Target Items, and the DX Database

Configuration clients use OPC DA interfaces to browse OPC DA and OPC DX servers for source and target items. Target items are always resident in the OPC DX server. Source items may be resident in OPC DA servers or OPC DX servers, including the target OPC DX server.

The configuration client selects the source and target items and uses them to define its DXConnections. DXConnections are defined in terms of attributes that identify the source and target items, characterize the data transfer between them, and provide additional descriptive information about the DXConnection.

The DX server stores DXConnections in a DX Database in its OPC DA address space (see Section 4). The OPC DX server periodically persists its DX Database for use when restarting (see Section 5.3).

OPC DA clients are able to browse the DX Database in the DX server using the OPC DA browse interface. The DX server allows clients to define the browse structure for DXConnections using the concept of the *browse path*.

The browse path defines the path used to reach the DXConnection when browsing the DX Database. Branches within the browse path represent logical groupings of DXConnections. Clients are allowed to define more than one browse path for each DXConnection, thus enabling DXConnections to belong to more than one logical grouping.

Although not shown in Figure 3, the DX Database also contains a list of source servers. Source servers are OPC DA and DX servers that contain source items. Each source server entry in the DX Database specifies the address of the source server and the type of interface (e.g. DA 2.05) used by the DX server to access it.

Configuration

The OPC DX specific services specified in Section 5 may be used concurrently by configuration clients to configure and maintain DXConnection and source server definitions in the DX Database.

Clients may use these services to operate on source servers, DXConnections, and in some cases, branches in the DX Database. When used on branches, the services are applied to the branch and all DXConnections below the branch.

DXConnection and source server definitions are protected by a Version attribute. When a client adds or modifies a DXConnection or source server, the DX server assigns a unique identifier to it called the version. Clients wishing to modify or delete the item must supply its current version to the DX server. If the version supplied is incorrect, the DX server rejects the request.

DXConnections and source servers become operational when they are added, independent of whether or not the client that created them remains connected to the DX server. They remain defined in the DX server until a client explicitly removes them.

Branches, on the other hand, are added automatically when a DXConnection is added if they don't already exist, and deleted automatically when the last DXConnection or branch under them is deleted. They also may be deleted explicitly, causing all DXConnections below them to be deleted.

However, both DXConnections and source servers have configurable default attributes that define whether or not data transfer is to be enabled when they become operational. When access to a source server is disabled, the DX server is not permitted be connected to it. In this case, none of the DXConnections with source items contained by it may access data from it.

Horizontal Data Transfer

Once a DXConnection is configured and enabled for data transfer, the DX server accesses the source server to acquire the value of its source item. The DX server updates the target value after performing any necessary conversions, transformations, substitutions, or manual overrides.

The runtime behavior of DXConnections is specified in Section 6. It includes behavior related to the configuration of DXConnections, as well as source connectivity failure, data receipt failures, error handling, data conversions, and manual overrides.

Control and Monitoring

OPC DX defines a set of OPC DA accessible runtime attributes that allow OPC DA clients to control the data transfer at different points in the data flow. These attributes allow OPC DA clients, during runtime, to:

- connect/disconnect the source,
- enable/disable updates to the target,
- define values to be substituted when the source value is unavailable,
- specify manual override values, and
- enable/disable the use of the override values.

These attributes are defined in Section 4.3.2.19. OPC DX services allow clients to save these runtime attribute values as defaults for each DXConnection, and to restore them from the defaults when desired.

The following subset of these runtime attributes are defined for branches in the browse path.

- connect/disconnect the source,
- enable/disable updates to the target, and
- enable/disable the use of the override values.

When used on a branch, these attributes are applied to all DXConnections reachable through that branch. For example, if a client disconnects the source for a branch, all DXConnections below the branch are disconnected from their sources. Because individual DXConnections can be subsequently reconnected, the value of these attributes on the branch is not meaningful when read by a client. Therefore, they are defined as write-only attributes for branches.

A related runtime attribute is defined for source servers that controls whether or not the DX server is permitted to connect to the source server. When access is denied, the DX server releases its connection to the source server, causing data transfer on all DXConnections from source items in that server to be stopped. Data transfer on these DXConnections remains stopped until access to the source server is granted. Then those whose connectivity to that source is enabled are connected.

OPC DX also defines runtime attributes for DXConnections to allow OPC DA clients to monitor the data transfer on DXConnections. OPC DA clients may read or subscribe to these attributes in addition to configuration attributes to monitor the status of a DXConnection.

3.2 Server Model

The DX server is defined as an extension from a fully compliant OPC DA server, conformant to one of the OPC DA specifications listed in Section 1.5. The underlying OPC DA server provides monitoring and control access to the DX Database specified in Section 4. The interface technology used by the underlying OPC DA server (DCOM and/or Web Service) is used for the DX server configuration services specified in Section 5. The DX server configuration or monitoring/control components must support the same interface technologies (DCOM and/or Web Service) as the underlying OPC DA server. The DX server provides the following capabilities in addition to those provided by the DA server.

DX Database	The DX server maintains a DX Database in its address space that contains descriptions of the DX server, DXConnections, and source servers. See Section 4 for a description of the DX Database.
Configuration Interface	This interface provides services for adding, modifying, and deleting DXConnections and source servers. The services of this interface and their externally visible behavior are defined in Sections 5 and 6. They are mapped to DCOM and Web Service interface technologies as specified in the appendices.
Source Access	The DX server contains a source access component that subscribes to source data. The DX server may support access to more than one source server type. See 4.2.7 for a list of defined source server types.
Runtime Behavior	The DX server contains a Runtime Behavior component that copies the data received from the source into the target. The rules for maintaining the subscription with the source server allow clients to disable the subscription, and define how the DX server is to recover from subscription failures. The rules for updating the target item allow clients to disable the update, manually override the received source value, and define a substitute value to use when the source data is not available. See Section 6 for a description of the runtime behavior.
Control and Monitoring Support	The DX server contains DA items defined specifically for runtime control and monitoring of the DX server, source servers, and individual DXConnections. These items are accessed through the OPC DA interface of the DX server. See the OPC DA specification for a description of the DA interface.

Figure 4 below illustrates this model.

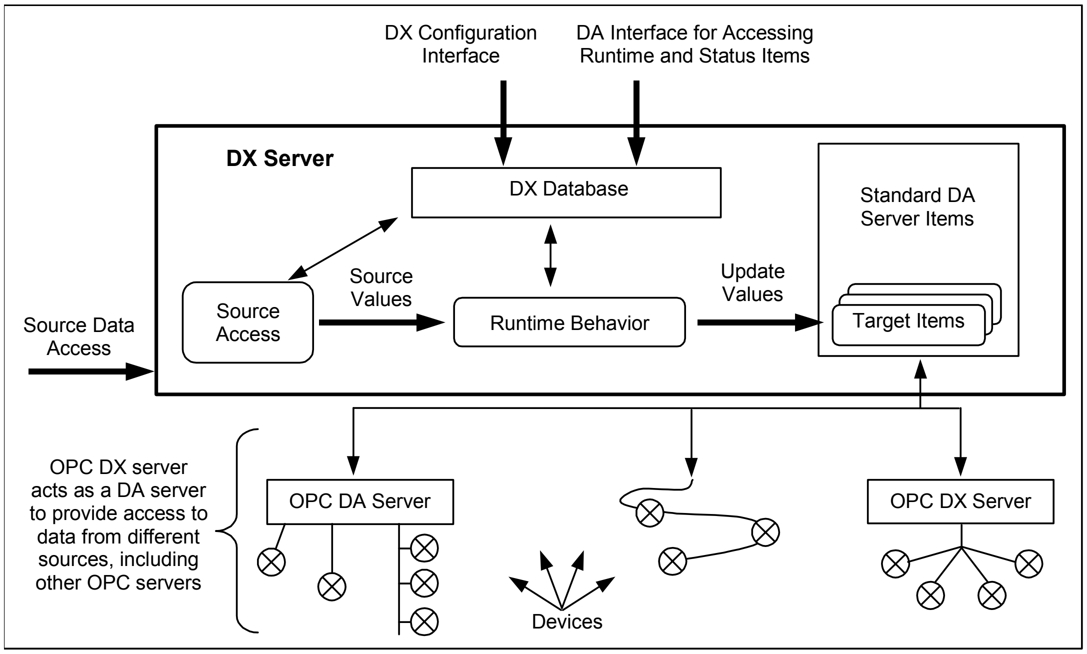
DX Configuration Interface
DA Interface for Accessing Runtime and Status Items
DX Server
DX Database
Standard DA Server Items
Source Access
Source Values
Runtime Behavior
Update Values
Target Items
Source Data Access
OPC DX server acts as a DA server to provide access to data from different sources, including other OPC servers
OPC DA Server
OPC DX Server
Devices

Figure 4 - DX Server Model

As a result, the DX server operates in three roles:

1. In the role of an OPC DA server, the OPC DX server provides the configuration client standard OPC DA interfaces to browse and access its data. Its data may come from devices, other OPC servers, or from other data sources to which it is connected.
2. In the role of an OPC DX server, the OPC DX server provides the configuration client a configuration interface for adding, modifying, and deleting DXConnections.
3. In the role of source access entity, the OPC DX server examines each of the DXConnections and determines if it is able to support accessing data for it from the source server. If so, it accesses the source server and transfers the data as specified in the DXConnection definition. For example, for OPC DA source servers that use DCOM, the DX server organizes source items from DXConnections into groups for access.

3.3 OPC Data eXchange Server Security

OPC DX Servers are essentially configurable OPC Data Access clients, with some logical extensions for enhanced reliability.

Therefore, the OPC DX server should follow the guidelines for security as defined in the OPC Security Custom Interface, Version 1.0, October 17, 2000 specification. Specifically the section that applies to the OPC Data eXchange initiative is described in the chapter titled DCOM Security Setup and Settings in the Guidelines chapter of the specification. As new versions of the OPC Security Specification are released, the OPC Data eXchange specification may be updated to support the new interfaces of functionality or the OPC Security specification.

4. DX Database

The DX server uses a reserved subtree of its address space for DX server items, as shown in Figure 5. This reserved subtree represents the DX Database. Every branch and leaf node within this subtree is an OPC DA item. This allows access to them through OPC DA interfaces.

As a result, each has a *Node Name* that is used for browsing, and a DA *Item ID* that uniquely identifies the item within the DX server. Node Names are either standardized by this specification, or assigned by DX configuration clients. The Item ID is vendor-specific and is assigned by the DX server. OPC DA clients browse to get the Node Name, call IOPCBrowseServerAddressSpace::GetItemID() using the Node Name, and then use the returned Item ID to access the item.

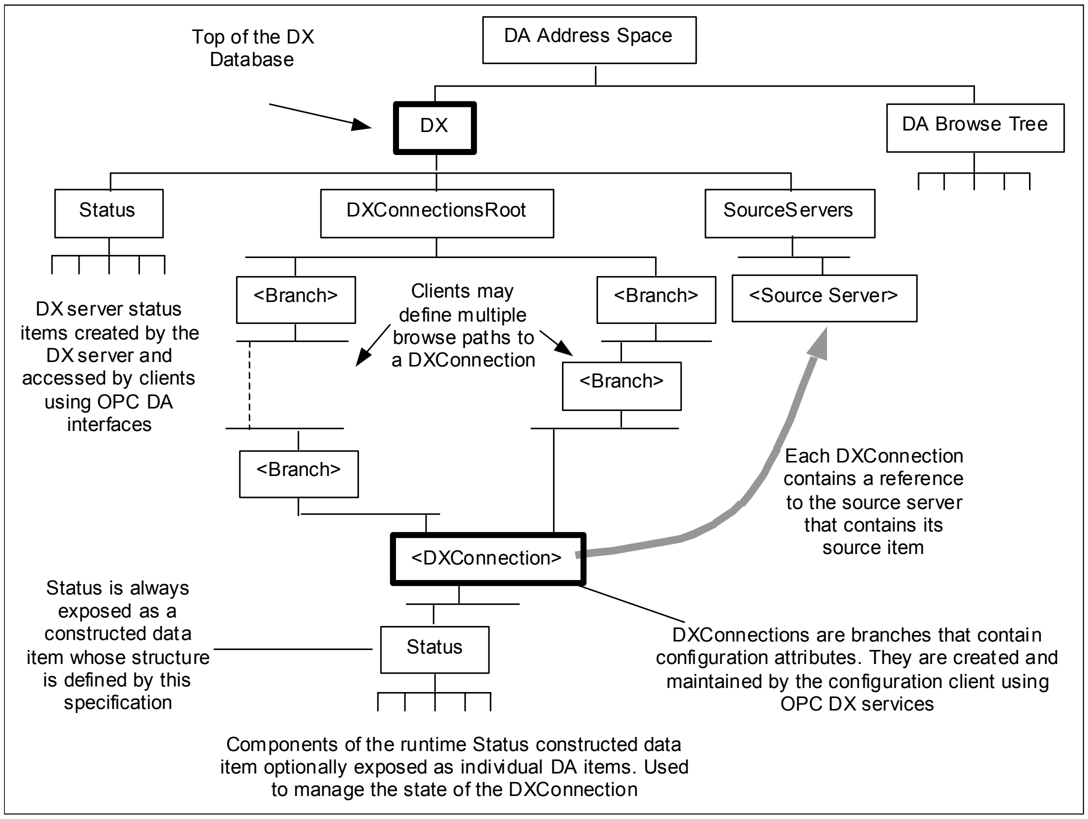
Top of the DX Database
DA Address Space
DX
DA Browse Tree
Status
DXConnectionsRoot
SourceServers
DX server status items created by the DX server and accessed by clients using OPC DA interfaces
<Branch>
Clients may define multiple browse paths to a DXConnection
<Branch>
<Source Server>
<Branch>
<Branch>
Each DXConnection contains a reference to the source server that contains its source item
<DXConnection>
Status is always exposed as a constructed data item whose structure is defined by this specification
Status
DXConnections are branches that contain configuration attributes. They are created and maintained by the configuration client using OPC DX services
Components of the runtime Status constructed data item optionally exposed as individual DA items. Used to manage the state of the DXConnection

Figure 5 - The Structure of the DX Branch of the OPC DA Address Space

This section provides a logical description of each item in the DX Database. Each item is described using the following table. The schema and examples are specified in this section. These descriptions may also define error codes appropriate for the item being described.

Top └ … └ …	Describes the browse hierarchy for the item. Angle brackets ("< >") surrounding node names in this tree indicate that the item name is assigned by the DX configuration client, and is not standardized by this specification. Names not in angle brackets, such as "Status", are standardized and reserved by this specification. They may not be used for client assigned names (those in angle brackets). Additionally, the "/" character is used to delimit branch names in the DX services defined in Section 5. Therefore, the "/" may not be used in client assigned names. Some nodes in the browse tree, such as a DXConnection node and a Source Server node, act as both a branch and a leaf. That is, they have children, and they have a value. But, because each is a single item in the browse tree, each has a single unique Item Identifier, and QueryItemProperties for that Item Identifier returns both leaf and branch properties.
Description	Description of the item
DA Access Rights	Client access to the item using OPC DA. Values are Read-only, Read/Write, and Write-Only.
COM Data Type	The Microsoft COM data type of the item.
XML DA Type	The XML data type of the item
Data Type ID	This is the name of the XML type definition for the constructed data item. It is used as the value of the OPC DX *Type ID* property Its property ID is 601, and its string name is "DataTypeID". XML data items are transferred using OPC DA as a string value (VT_BSTR). The string is an XML document that contains an element whose type is identified by this element.
XML Schema for the item	

4.1 DX Database Root

The root of the DX subtree is the top of the DX Database. It is a branch whose parent node is the DA Address Space root. It is always present in a DX server.

Top └ **DX**	
DA Access Rights	None (branch)

4.2 ServerStatus

This item is a branch whose subordinate items are attributes that are used to provide clients with status information about the DX server. This branch cannot appear in the BrowsePath of a DXConnection. This item is updated only by the DX server. It may be used by both configuration and status/monitoring clients.

The DX Status attributes can be accessed by OPC DA clients as a constructed item. Monitoring clients may subscribe to it to receive notifications when any of the component values change.

```
Top
 └ DX
    └ ServerStatus
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR that contains an XML document of the following DX specific XML Complex Type
XML DA Type	string that contains the following DX specific XML Complex Type
Data Type ID	”ServerStatus”

```
<s:complexType name="ServerStatus">
  <s:sequence>
    <s:element name="ServerState" type="s0:serverState" />
    <s:element name="ConfigurationVersion" type="s:string" />
    <s:element name="DXConnectionCount" type="s:unsignedInt" />
    <s:element name="MaxDXConnections" type="s:unsignedInt" />
    <s:element name="DirtyFlag" type="s:boolean" />
    <s:element name="ErrorID" type="s0:OPCError" />
    <s:element name="ErrorDiagnostic" type="s:string" />
    <s:element name="SourceServerTypes" type="s0:ArrayOfString" />
    <s:element name="MaxQueueSize" type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

<s:complexType name="ArrayOfString">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
               name="ServerType" nillable="true" type="s:string" />
  </s:sequence>
</s:complexType>

<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" />
</s:complexType>
```

4.2.1 ConfigurationVersion

This item uniquely identifies the current version of the DX server configuration information. Any change to the configuration through one of the DX configuration services specified in Section 5 causes the DX server to generate a new, unique, previously unused ConfigurationVersion. The Version value is not changed for any other reason, and the configuration may not be changed via any other mechanism. The DX server is not allowed to reuse a ConfigurationVersion value, even through restarts.

Clients can use this information to determine if the configuration they have locally is up-to-date. They can read this item or they can subscribe to it to receive notifications when other clients make changes to the DX server configuration.

Top └ DX └ ServerStatus └ **ConfigurationVersion**	
DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.2.2 ServerState

This item identifies the current state of the OPC-DX server. The states and their definitions are a superset of those defined in the OPC DA Custom Interface Specification Version 3.0 and the OPC XML-DA Specification.

This attribute is updated internally by the DX server and cannot be updated by DX or DA clients.

Top └ DX └ ServerStatus └ **ServerState**	
DA Access Rights	Read-only
COM Data Type	VT_BSTR (See Table 1 for a list of the valid values)
XML DA Type	ServerState

```
<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="shutdown" />
    <s:enumeration value="test" />
    <s:enumeration value="commFault" />
    <s:enumeration value="unknown" />
  </s:restriction>
</s:simpleType>
```

Table 1 – Server State Values

Value	Description
"running"	The server is running normally. This is the usual state for a server
"failed"	A vendor specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor specific. An error code of E_FAIL should generally be returned from any other server method.
"noConfig"	The server is running but has no configuration information loaded and thus do not transfer data. DX servers enter this state prior to their initial configuration and after having their configuration cleared.
"suspended"	The server has been temporarily suspended via some vendor specific method and is not getting or sending data. Note that Quality will be returned as QUALITY_OUT_OF_SERVICE.
"shutdown"	The server has shutdown. Depending on the implementation, this may or may not be visible to clients.
"test"	The server is in Test Mode. The outputs are disconnected from the real hardware but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.
"commFault"	The server is running properly but is having difficulty accessing data from its data sources. This may be due to communication problems, or some other problem preventing the underlying device, control system, etc. from returning valid data. It may be complete failure, meaning that no data is available, or a partial failure, meaning that some data is still available. It is expected that items affected by the fault will individually return with a BAD quality indication for the items.
"unknown"	This state is not used for the local DX server. It is used only for SourceServers in the DX Database. It indicates that the DX server does not know the state of the source server.

4.2.3 DXConnectionCount

This item stores the current number of DXConnections in the configuration. The DX server modifies this value whenever a DXConnection is added to or deleted from the configuration.

```
Top
 └ DX
    └ ServerStatus
       └ DXConnectionCount
```

DA Access Rights	Read-only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.2.4 MaxDXConnections

This item stores the maximum number of DXConnections that a DX server supports due to licensing or other limitations. If only system resources limit the server, then the value of this item is 0xFFFFFFFF.

```
Top
 └ DX
    └ ServerStatus
       └ MaxDXConnections
```

DA Access Rights	Read-only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.2.5 DirtyFlag

This item indicates, when FALSE, that the current configuration, as identified by the Configuration Version (Section 4.2.1), has been persisted. If the configuration has been modified and has not been persisted, the Dirty flag will be TRUE (see Section 5.3). The Dirty flag is cleared when the configuration is successfully persisted.

Top └ DX └ ServerStatus └ **DirtyFlag**	
DA Access Rights	Read-only
COM Data Type	VT_BOOL
XML DA Type	boolean

4.2.6 ErrorID

This item identifies the active fault with the highest severity detected for the DX server itself (not including errors associated with individual DXConnections). For example, a failure occurring during the persisting of the current configuration would be an error reported at this level. Table 2 specifies the error codes for this attribute. When used with COM DA, all errors shown in the table are prefixed with OPC_. For example, "E_FAIL" becomes "OPC_E_FAIL" for COM DA.

Some of the errors that may occur that are represented as the current active fault, may not be able to be resolved programmatically or by the OPC DX server. Specifically errors that require the configuration file to be altered may require a human to resolve the issue that would subsequently resolve the error associated with the active fault of the OPC DX server. These errors include the OPC DX defined errors as well as the vendor specific errors.

Top └ DX └ ServerStatus └ **ErrorID**	
DA Access Rights	Read-only
COM Data Type	VT_I4
XML DA Type	Qname

Table 2 – DX Server Error Codes

Code	Description
S_OK	No error code is set.
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7. The server is not operating to due an internal error.
E_OUTOFMEMORY	See Section 5.1.7. The server is not operating to due memory limitations.
E_SERVER_STATE	See Section 5.1.7. The server is not operating to due being in an invalid state.
E_INVALID_CONFIG_FILE	See Section 5.1.7
E_PERSIST_FAILED	See Section 5.1.7
E_TARGET_FAULT	See Section 5.1.7
E_TARGET_NO_ACCESS	See Section 5.1.7

4.2.7 ErrorDiagnostic

This item contains the vendor specific additional detail for the ErrorID.

Top
└ DX
    └ ServerStatus
        └ **ErrorDiagnostic**	

DA Access Rights	Read-only
COM Data Type	BSTR
XML DA Type	string

4.2.8 SourceServerTypes

This item identifies Supported Source Server Types that the DX server is capable of supporting for connections to source servers. A DX server may be capable of supporting vendor specific server types.

Source servers are added to the DX Source Servers branch using the AddServers configuration service (see Section 5.2.1.2) and must be of one of the server types defined by this item. A list of standard server types is described contained in Table 3. URIs are used to identify server types.

Top
└ DX
    └ ServerStatus
        └ **SourceServerTypes**	

DA Access Rights	Read-only
COM Data Type	VT_ARRAY | VT_BSTR
XML DA Type	ArrayOfString

```
<s:complexType name="ArrayOfString">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
               name="ServerType" nillable="true" type="s:string" />
  </s:sequence>
</s:complexType>
```

Table 3 – Supported Source Server Types

"COM-DA1.0"	
"COM-DA2.04"	Server data type conversions may not be supported.
"COM-DA2.05"	Server data type conversions fully supported.
"COM-DA3.0"	
"XML-DA1.0"	

4.2.9 MaxQueueSize

This item stores the maximum number of source updates that a DX server can queue per DXConnection, when data is received faster than it can be written to the target. If the DX server does not support queuing, this value should be set to 1 indicating that only the latest source update will be cached for target updates.

```
Top
 └ DX
    └ ServerStatus
        └ MaxQueueSize
```

DA Access Rights	Read-only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.3 DXConnectionsRoot

This item is a branch that contains the DXConnection definitions. It is used by one or more configuration clients to define source servers and DXConnections. OPC DX allows each configuration client to organize its DXConnections under one or more branches beneath this branch.

```
Top
 └ DX
    └ DXConnectionsRoot
```

DA Access Rights	None

4.3.1 <Branch>

DXConnections exist directly under the DXConnections branch or under branches that are under the DXConnections branch. Branches are added to the DX Database automatically, if they do not already exist, when DXConnections are added. The OPC DX BrowsePath service parameter (see Section 5.1.1) defines the branches for a DXConnection. The BrowsePath parameter represents the path from the DXConnections branch to DXConnection under it. A DXConnection may be defined with one, or more browse paths.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <Branches>  ← This level represents a permitted, but optional, hierarchy of BranchNames
            └ <BranchName>
```

DA Access Rights	None

4.3.1.1 Status

Status is a runtime attribute that the DX server automatically adds as a DA item below a branch when it adds the branch to the DX Database. Status is updated during runtime by the DX server. Its components are used to control the operational status of a DXConnection.

The Status attribute is composed of the following runtime attributes..

Top
└ DX
   └ DXConnectionsRoot
      └ <Branches>
         └ <BranchName>
            └ **Status**	

DA Access Rights	None

4.3.1.1.1 Overridden

Overridden is a runtime attribute that the DX server automatically adds as a DA item below a branch when it adds the branch to the DX Database.

When a client writes a value to this attribute, the DX server writes this value to the Overridden runtime attribute of all DXConnections located at any level under the branch.

This attribute is not readable because the value of this attribute might not reflect the current value of this attribute for the DXConnections below this branch.

Top
└ DX
   └ DXConnectionsRoot
      └ <Branches>
         └ <BranchName>
            └ Status
               └ **Overridden**	

DA Access Rights	Write-only
COM Data Type	VT_BOOL
XML DA Type	Boolean

4.3.1.1.2 SourceItemConnected

SourceItemConnected is a runtime attribute that the DX server automatically adds as a DA item below a branch when it adds the branch to the DX Database.

When a client writes a value to this attribute, the DX server writes this value to the SourceItemConnected runtime attribute of all DXConnections located at any level under the branch.

This attribute is not readable because the value of this attribute does not reflect the current value of this attribute for the DXConnections below this branch.

Top
└ DX
  └ DXConnectionsRoot
    └ <Branches>
      └ <BranchName>
        └ Status
          └ **SourceItemConnected**	

DA Access Rights	Write-only
COM Data Type	VT_BOOL
XML DA Type	boolean

4.3.1.1.3 TargetItemConnected

TargetItemConnected is a runtime attribute that the DX server automatically adds as a DA item below a branch when it adds the branch to the DX Database.

When a client writes a value to this attribute, the DX server writes this value to the TargetItemConnected runtime attribute of all DXConnections located at any level under the branch.

This attribute is not readable because the value of this attribute does not reflect the current value of this attribute for the DXConnections below this branch.

Top
└ DX
  └ DXConnectionsRoot
    └ <Branches>
      └ <BranchName>
        └ Status
          └ **TargetItemConnected**	

DA Access Rights	Write-only
COM Data Type	VT_BOOL
XML DA Type	Boolean

4.3.2 DXConnection

Each DXConnection in the DX Database is represented as a DXConnection branch. Each has its own browseable name that may appear under one or more browse paths in the DX Database, as assigned by the client when adding the DXConnection. BrowsePath are the bracnches between the DXConnectionRoot branch and the DXConnection. This name is referred to as the DXConnection Name, and is unique within the scope of the branch that contains the DXConnection.

Clients can recognize DXConnections in the browse tree through the Data Type ID property (value = “DXConnection”).

Each DXConnection has a one-to-one relationship with its Target Item. Each DXConnection references a single Target Item, and each Target Item is allowed to have only one DXConnection.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR that contains an XML document that contains an XML Complex Type
XML DA Type	string that contains the following DX specific XML Complex Type
Data Type ID	”DXConnection” (see Section 5.1.2 and Appendix tjb)

Each DXConnection has configuration attributes used by configuration clients to define the transfer of data on the DXConnection. They may be updated only using the DX configuration services specified in Section 5. Configuration attributes are not writable through OPC DA.

The DXConnection configuration attributes are accessible through OPC DA as a read-only constructed data item. Status and monitoring clients may subscribe to the DXConnection to receive notifications when the Version component of the DXConnection configuration changes.

The subsections that follow specify the attributes of the DXConnection. As an option, DX servers may expose these attributes as read-only OPC DA items below the DXConnection. When they do, monitoring clients may subscribe to them to receive notifications when their values change.

4.3.2.1 Version

This configuration attribute is used to uniquely identify the current version of the DXConnection's configuration attributes. Any change to the DXConnection's configuration through one of the DX configuration add, modify, or delete services specified in Section 5 causes the DX server to generate a new, unique, previously unused Version. The Version value is not changed for any other reason, and these attributes may not be changed via any other mechanism. The DX server is not allowed to reuse a Version value, even through restarts.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Version
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.2 Description

This configuration attribute is a description of the DXConnection. It is exposed as ItemProperty 101 of the DXConnection item.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Description
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.3 Keyword

This configuration attribute is a string assigned to this DXConnection by the client. The same keyword may be assigned to one or more DXConnections. Clients may use this element to construct categories of DXConnections or to assign administrative authorities to DXConnections. This element can be very useful in the QueryDXConnections to filter DXConnections.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Keyword
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.4 DefaultSourceItemConnected

This configuration attribute defines the default value for the SourceItemConnected runtime attribute of this DXConnection (see Section 4.3.2.19.15). The value of this attribute is copied to that runtime attribute of the DXConnection if and only if a value has not yet been assigned to that runtime attribute.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ DefaultSourceItemConnected
```

DA Access Rights	Read-only
COM Data Type	VT_BOOL
XML DA Type	boolean

4.3.2.5 DefaultTargetItemConnected

This configuration attribute defines the default value for the TargetItemConnected runtime attribute of this DXConnection (see Section 4.3.2.19.16). The value of this attribute is copied to that runtime attribute of the DXConnection if and only if a value has not yet been assigned to that runtime attribute .

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ DefaultTargetItemConnected
```

DA Access Rights	Read-only
COM Data Type	VT_BOOL
XML DA Type	boolean

4.3.2.6 DefaultOverridden

This configuration attribute defines the default value for the Overridden runtime attribute of this DXConnection (see Section 4.3.2.19.17). The value of this attribute is copied to that runtime attribute of the DXConnection if and only if a value has not yet been assigned to that runtime attribute.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ DefaultOverridden
```

DA Access Rights	Read-only
COM Data Type	VT_BOOL
XML DA Type	Boolean

4.3.2.7 DefaultOverrideValue

This configuration attribute defines the default value for the OverrideValue runtime attribute of this DXConnection (see Section 4.3.2.19.18). The value of this attribute is copied to that runtime attribute of the DXConnection if and only if a value has not yet been assigned to that runtime attribute.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ DefaultOverrideValue
```

DA Access Rights	Read-only
COM Data Type	Datatype of target item
XML DA Type	Datatype of target item

4.3.2.8 SubstituteValue

This configuration attribute defines the value to be substituted by the DX server for the source value when the source value is not available. See Section 6.3.5 for the specification of when the SubstituteValue is used. This is only used if EnableSubstituteValue is set to TRUE.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ SubstituteValue
```

DA Access Rights	Read-only
COM Data Type	Datatype of target item
XML DA Type	Datatype of target item

4.3.2.9 EnableSubstituteValue

This configuration attribute indicates, when TRUE, that the SubstituteValue, if present, is used to update the target value. When FALSE, the SubstituteValue is not used. The default value for this attribute is FALSE. See Section 6.3.5 for the specification of the use of the SubstituteValue.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ EnableSubstituteValue
```

DA Access Rights	Read-only
COM Data Type	VT_BOOL
XML DA Type	Boolean

4.3.2.10 TargetItemPath

This configuration attribute may be used for OPC DA COM and OPC XML DA target items. When used for OPC DA COM, may contain the OPC DA AccessPath of the target item. When used for OPC XML DA, it contains the OPC XML DA ItemPath of the target item.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ TargetItemPath
```

Data Type	String
DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.11 TargetItemName

This configuration attribute identifies the target item. For OPC COM-DA, it contains the fully qualified *ItemID.* For OPC COM-DA, it contains the OPC XML-DA *ItemName*.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ TargetItemName
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.12 SourceServerName

This configuration attribute identifies the source server of the source item. It contains the node name of the source server item in the DX Database. See Section 4.4.1 for a description of source servers.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ SourceServerName
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.13 SourceItemPath

This configuration attribute may be used for OPC DA COM and OPC XML DA source items. When used for OPC DA COM, it may contain the OPC DA AccessPath of the source item. When used for OPC XML DA, it contains the OPC XML DA ItemPath of the source item.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ SourceItemPath
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.14 SourceItemName

This configuration attribute identifies the source item. For OPC COM-DA, it contains the fully qualified *ItemID.* For OPC COM-DA, it contains the OPC XML-DA *ItemName*.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ SourceItemName
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.15 QueueSize

This configuration attribute defines the maximum number of source updates to queue per DXConnection, when data is received faster than it can be written to the target. The range of this number is 1 to MaximumSourceItemUpdateQueueSize, the upper limit supported by the DX server.

This attribute has no affect on buffering in the source DA server.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ QueueSize
```

DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.3.2.16 UpdateRate

This configuration attribute specifies the time interval that reflects the rate at which source data changes are provided by the source server. This value is specified in milliseconds. See OPC DA 2.0 or higher for a more precise definition.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ UpdateRate
```

DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.3.2.17 Deadband

This configuration attribute specifies the percent change in the source item value that will cause the source server to transfer the data to the DX server. See OPC DA 2.0 or higher for a more precise definition.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Deadband
```

DA Access Rights	Read-Only
COM Data Type	VT_R4
XML DA Type	single

4.3.2.18 VendorData

This configuration attribute contains DX server vendor specific information about the DXConnection. VendorData is reserved for use by DX server vendors for transferring data that may only be understood by certain clients and servers for the purpose of custom configuration.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ VendorData
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.19 Status

Status is a runtime attribute that the DX server automatically adds as a DA item below a DXConnection when it adds the DXConnection to the DX Database.

Status contains two types of components, those used for monitoring and those used to manage the state of the DXConnection. The DX server makes these available in a single access (a DA Read or Write of constructed data), and it may optionally make them accessible as individual items using the name of each. Components used for monitoring items are read-only, and those used for state management are read/write.

The monitoring components are updated during runtime by the DX server. They describe the operational status of a DXConnection. Clients may monitor them using the DA interface to subscribe to this constructed data item. Component definitions below indicate whether the component triggers notifications.

Writing to the status attribute is only possible directly through the DA client interface. When writing to it as a constructed data item, the values of the read-only components are ignored. Note that with XML, it may be possible for the client to omit the read-only components.

Changes to configuration attributes are not reflected in the status attributes unless the DX client explicitly synchronizes them. The OPC DX services to synchronize the configuration and status attributes are described in Sections 5.2.1.5 and 5.2.2.6.

Top
└ DX
  └ DXConnectionsRoot
    └ <BrowsePath>
      └ <DXConnectionName>
        └ **Status**	

DA Access Rights	Read/Write
COM Data Type	VT_BSTR that contains an XML document of the following DX specific XML Complex Type
XML DA Type	string that contains the following DX specific XML Complex Type
Data Type ID	” DXConnectionStatus”

```
<s:complexType name="DXConnectionStatus">
  <s:sequence>
    <s:element name="DXConnectionState" type="s0:DXConnectionState" />
    <s:element name="WriteValue" />
    <s:element name="WriteTimestamp" type="s:dateTime" />
    <s:element name="WriteQuality" type="s0:DXQuality" />
    <s:element name="WriteErrorID" type="s0:OPCError" />
    <s:element name="WriteErrorDiagnostic" type="s:string" />
    <s:element name="SourceValue" />
    <s:element name="SourceTimestamp" type="s:dateTime" />
    <s:element name="SourceQuality" type="s0:DXQuality" />
    <s:element name="SourceErrorID" type="s0:OPCError" />
    <s:element name="SourceErrorDiagnostic" type="s:string" />
    <s:element name="ActualUpdateRate" type="s:unsignedInt" />
    <s:element name="QueueHighWaterMark" type="s:unsignedInt" />
    <s:element name="QueueFlushCount" type="s:unsignedInt" />
    <s:element name="SourceItemConnected" type="s:boolean" />
    <s:element name="TargetItemConnected" type="s:boolean" />
    <s:element name="Overridden" type="s:boolean" />
    <s:element name="OverrideValue" />
  </s:sequence>
</s:complexType>

<s:complexType name="DXQuality">
  <s:attribute name="Quality" type="s0:qualityStatus" />
  <s:attribute name="LimitBits" type="s0:limitStatus" />
  <s:attribute name="VendorBits" type="s:unsignedLong" />
</s:complexType>

<s:simpleType name="qualityStatus">
  <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="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="limitStatus">
  <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>
```

4.3.2.19.1 DXConnectionState

This item identifies the current state of the DXConnection. The state of the DXConnection is reflected by one of the values specified in Table 4.

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this item occur.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ DXConnectionState
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR (See Table 4 for a list of the valid values)
XML DA Type	DXConnectionState

```
<s:simpleType name="DXConnectionState">
  <s:restriction base="s:string">
    <s:enumeration value="initializing" />
    <s:enumeration value="operational" />
    <s:enumeration value="deactivated" />
    <s:enumeration value="sourceServerNotConnected" />
    <s:enumeration value="subscriptionFailed" />
    <s:enumeration value="targetItemNotFound" />
  </s:restriction>
</s:simpleType>
```

Table 4 – DXConnectionState Values

Value	Description
“initializing”	The DXConnection is initializing or reinitializing
“operational”	The DXConnection is operating normally.
“deactivated”	The transfer of data from the source item has been terminated or was never initiated
“sourceServerNotConnected”	The DXConnection to the source is not established
“subscriptionFailed”	A subscription for the DXConnnection could not be established or the source item was not found.
“targetItemNotFound”	The target item could not be located in the DX server

4.3.2.19.2 WriteValue

This status attribute indicates the last value the DX server attempted to write to the target item, whether or not the write succeeded.

The WriteValue, the WriteTimestamp, the WriteQuality, and the WriteErrorID elements are updated together after each write attempt, even it the write failed, and even if the target timestamp and quality are not writable.

The value for this element when no writes have been attempted is undefined, and its quality is QUALITY_NOT_CONNECTED.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ WriteValue
```

DA Access Rights	Read-Only
COM Data Type	Datatype of target item
XML DA Type	Datatype of target item

4.3.2.19.3 WriteTimestamp

This status attribute indicates the time that the DX server last attempted to write to the target item, whether or not the write succeeded. This timestamp is generated by the DX server.. WriteTimestamp is independent of whether or not the target accepts writes to timestamp.

The value for this element when no writes have been attempted is undefined, and its quality is QUALITY_NOT_CONNECTED.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ WriteTimestamp
```

DA Access Rights	Read-Only
COM Data Type	VT_DATE
XML DA Type	dateTime

4.3.2.19.4 WriteQuality

This status attribute indicates the quality of the value that the DX server last attempted to write to the target item, whether or not the write succeeded. WriteQuality is independent of whether or not the target accepts writes to quality. Note that vendor defined quality codes should be mapped on their broader category such as 'bad', 'good' or 'uncertain'.

The value for this element when no writes have been attempted is undefined, and its quality is QUALITY_NOT_CONNECTED.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ WriteQuality
```

DA Access Rights	Read-Only
COM Data Type	VT_I2 (OPCQUALITY stored in an I2)
XML DA Type	DXQuality

```
<s:complexType name="DXQuality">
  <s:attribute name="Quality"    type="s0:qualityStatus" />
  <s:attribute name="LimitBits"  type="s0:limitStatus" />
  <s:attribute name="VendorBits" type="s:unsignedLong" />
</s:complexType>

<s:simpleType  name="qualityStatus">
  <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="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="limitStatus">
  <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>
```

4.3.2.19.5 WriteErrorID

This status attribute indicates the error associated with last attempt to write a value to the target. If no error occurred, the value is S_OK. The value for this element when the target is disconnected is E_TARGETITEM_DISCONNECTED. The value for this element when the target is connected and no writes have been attempted is E_NOWRITESATTEMPTED.

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this element occur.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ WriteErrorID
```

DA Access Rights	Read-Only
COM Data Type	VT_I4
XML DA Type	QName

Table 5 – Target Item Error Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_TARGET_ITEM_DISCONNECTED	See Section 5.1.7
E_TARGET_FAULT	See Section 5.1.7
E_TARGET_NO_ACCESS	See Section 5.1.7
E_TARGET NO_WRITES_ATTEMPTED	See Section 5.1.7
E_TARGET_INVALID _ITEM	See Section 5.1.7
E_TARGET UNKNOWN _ITEM	See Section 5.1.7
E_TARGET_ITEM_BADTYPE	See Section 5.1.7
E_TARGET_ITEM_RANGE	See Section 5.1.7
S_TARGET_SUBSTITUTED	See Section 5.1.7
S_TARGET_OVERRIDEN	See Section 5.1.7
S_CLAMP	See Section 5.1.7

4.3.2.19.6 WriteErrorDiagnostic

This item contains the vendor specific additional detail for the WriteErrorID.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ WriteErrorDiagnostic
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.19.7 SourceValue

This status attribute is the value of the source data item. It is not modified by the DX server and simply indicates the last value received from the data source on the DXConnection.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ SourceValue
```

DA Access Rights	Read-Only
COM Data Type	Datatype of the received value
XML DA Type	Datatype of the received value

4.3.2.19.8 SourceTimestamp

This status attribute is the timestamp generated by the source server for this data item. It is not modified by the DX server and simply indicates the time this value was last validated by the source server.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ SourceTimestamp
```

DA Access Rights	Read-Only
COM Data Type	VT_DATE
XML DA Type	dateTime

4.3.2.19.9 SourceQuality

This status attribute stores the quality value that was received from the data source item. Clients that read this item can expect to receive quality codes supported by the SourceServerType. Note that vendor defined quality codes should be mapped on their broader category such as ‘bad’, ‘good’ or ‘uncertain’.

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this element occur.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ SourceQuality
```

DA Access Rights	Read-Only
COM Data Type	VT_I2 (OPCQUALITY stored in an I2)
XML DA Type	DXQuality (see Section 4.3.2.19.4)

4.3.2.19.10 SourceErrorID

This status attribute identifies an error condition associated with the source server accessing the the source item for this DXConnection. When the error condition clears, this element is cleared. Therefore, this element contains only error ids, and is not present in the XML for the SourceStatus unless it has a valid error id value.

It is the responsibility of DX server implementor to define an appropriate conversion between COM error codes and XML qualified names if the interface technology used by the DX server is different than the technology used by the source server. In other words, the source error id may not necessarily be an error code returned directly from the source server. It is recommended that DX servers, at a minimum, call GetErrorString on any COM-DA source server and place the results in SourceErrorDiagnostic so the DX monitoring client will have some idea what the underlying source server error was. XML-DA source servers provide error text in the response that can be directly placed in SourceErrorDiagnostic.

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this element occur.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ SourceErrorID
```

DA Access Rights	Read-Only
COM Data Type	VT_I4
XML DA Type	QName

Table 6 – Source Item Error Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_SOURCE_SERVER_NOT_CONNECTED	See Section 5.1.7
E_SOURCE_SERVER_FAULT	See Section 5.1.7
E_SOURCE_SERVER_ NO_ACCESSS	See Section 5.1.7
E_SOURCE_ITEM_BADRIGHTS	See Section 5.1.7
E_SOURCE_ITEM_BAD_QUALITY	See Section 5.1.7
E_SOURCE_ITEM_BADTYPE	See Section 5.1.7
E_SOURCE_ITEM_RANGE	See Section 5.1.7
E_SOURCE UNKNOWN_ ITEM	See Section 5.1.7
E_SOURCE_INVALID_ITEM	See Section 5.1.7
E_SUBSCRIPTION_FAULT	See Section 5.1.7

4.3.2.19.11 SourceErrorDiagnostic

This item contains the vendor specific additional detail for the SourceErrorID.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ SourceErrorDiagnostic
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.3.2.19.12 ActualUpdateRate

This status attribute is the actual update rate that is being used on the DXConnection. It may be different than the UpdateRate configured for the DXConnection. Specifically this is the actual update rate that the DX server computes from the configured arguments of UpdateRate and Deadband, and will be the actual update rate of the subscription.

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this element occur.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ ActualUpdateRate
```

DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.3.2.19.13 QueueHighWaterMark

This status attribute indicates the highest number of entries that have been concurrently queued in the source data queue. It is cleared each time the queue is flushed, and each time the ActualUpdateRate changes.

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this element occur.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ QueueHighWaterMark
```

DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.3.2.19.14 QueueFlushCount

This status attribute indicates the number of queue flushes for this DXConnection since DX server start

The DX server must provide a notification to any DA clients that have subscribed to the parent Status item if any changes to this element occur.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ QueueFlushCount
```

DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	unsignedInt

4.3.2.19.15 SourceItemConnected

This status attribute is used to control the data flow from the source item to the DX server. When TRUE, the DX server attempts to acquire the source data from the source server if the source server is connected (see Section 4.4.1.6.9). When FALSE, the DX server no longer attempts to acquire the source data from the source.

When SourceItemConnected is set to FALSE in all DXConnections to a specific source server, then the DX server disconnects from the source server (e.g., removes all groups, cancels all subscriptions, releases all interfaces, closes all connections).

When a DXConnection is added, the value of this attribute is copied from the value of the DXConnection's DefaultSourceItemConnected configuration attribute (see Section 4.3.2.4). Once the DXConnection is initialized, OPC DA clients may update this attribute. DX clients can return the value of this attribute to its default value using the CopyDefaultDXConnectionAttributes service (see Section 5.2.2.6).

Writing TRUE to this attribute when the SourceServerConnected attribute (see Section 4.4.1.6.9) of the Source Server associated with this DXConnection is set to FALSE has no immediate effect, because the DX server cannot attempt to connect to the source until the SourceServerConnected attribute changes to TRUE.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ SourceItemConnected
```

DA Access Rights	Read/Write
COM Data Type	VT_BOOL
XML DA Type	boolean

4.3.2.19.16 TargetItemConnected

This status attribute is used to control the data flow from the DX server to the target item. When TRUE, the DX server attempts to update the target value. When FALSE, the DX server does not attempt to update the target value. See Section 6.3.5 for the rules for updating the target value. The default value for this attribute is configured for the DXConnection using DefaultTargetItemConnected (see Section 4.3.2.5).

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ TargetItemConnected
```

DA Access Rights	Read/Write
COM Data Type	VT_BOOL
XML DA Type	boolean

4.3.2.19.17 Overridden

This status attribute indicates, when TRUE, that the OverrideValue, if present, overrides the source and substitute values. When FALSE or when the OverrideValue is not present, then the OverrideValue does not override the source and substitute values. The default value for this attribute is FALSE.

Note that overriding is comparable to "forcing" in other systems or specifications. Overriding the target requires that TargetConnected = TRUE.

```
Top
 └ DX
    └ DXConnectionsRoot
       └ <BrowsePath>
          └ <DXConnectionName>
             └ Status
                └ Overridden
```

DA Access Rights	Read/Write
COM Data Type	VT_BOOL
XML DA Type	boolean

4.3.2.19.18 OverrideValue

This status attribute contains the value used to override the source and substitute values. The default value for this element is configured for the DXConnection using the DefaultOverrideValue.

Attempting to clear the value of this element when Overriden is TRUE causes the attempt to fail and the error E_INVALIDOVERRIDEATTEMPT to be returned.

```
Top
 └ DX
    └ DXConnectionsRoot
        └ <BrowsePath>
            └ <DXConnectionName>
                └ Status
                    └ OverrideValue
```

DA Access Rights	Read/Write
COM Data Type	Datatype of target item
XML DA Type	Datatype of target item

4.4 Source Servers

The Source Servers branch contains a list of client-defined servers that support DXConnections. This branch is always present. Only source server nodes may be added directly beneath this branch. That is, branches, other than source servers, and DXConnections may not be added beneath this branch.

```
Top
 └ DX
    └ SourceServers
```

DA Access Rights	None

4.4.1 SourceServer

This item defines a source server. Clients add, modify, and delete source servers using only the DX services specified in Section 5. Each source server has its own browseable name that may appear under the Source Servers branch in the DX Database, as assigned by the client when adding the source server. This name is referred to as the Server Name, and is unique within the scope of the Source Servers branch.

To allow internal connections the DX server the client can add the definition of the DX server as a source server.

Status and monitoring clients may subscribe to this DA item to receive notifications when the source server Version attribute has been modified. If the source server name is changed, the DX server is permitted to change the ItemID of the source server, and therefore status and monitoring clients may have to resubscribe to this item.

The configuration and status attributes of the source server are specified in the subsections that follow. The configuration attributes comprise the source server constructed data item. DX servers expose the configuration attributes as read-only OPC DA items below the source server using the node name specified for each. Monitoring clients may subscribe to them to receive notifications when their values change.

Top └ DX └ SourceServers └ **<ServerName>**	
DA Access Rights	Read-only
COM Data Type	VT_BSTR that contains an XML document of the following DX specific XML Complex Type
XML DA Type	string that contains the following DX specific XML Complex Type
Data Type ID	”SourceServer” (see Section 5.1.8 and Appendix tjb)

4.4.1.1 Version

This configuration attribute is used to uniquely identify the current version of the ServerURL, ServerType, and DefaultSourceItemConnected source server attributes. The DX server assigns this value. Any change to these attributes through one of the DX configuration add, modify, or delete services specified in Section 5 causes the DX server to generate a new, unique, previously unused Version. The Version value is not changed for any other reason, and these attributes may not be changed via any other mechanism. The DX server is not allowed to reuse a Version value, even through restarts.

Top └ DX └ SourceServers └ <ServerName> └ **Version**	
DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.4.1.2 Description

This configuration attribute is a description of the source server, and is obtained from the source server. It is recommended that it contain the name of the company and the type of device(s) supported. It is exposed as ItemProperty 101 of the source server branch.

Top └ DX └ SourceServers └ <ServerName> └ **Description**	
DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.4.1.3 ServerType

This configuration attribute identifies the type of interface to be used to access the source server. URIs are used for identifying server types. The source server type identified by this attribute must be included in the Supported Source Server Types for the DX server (see Section 4.2.7). A list of standard server types is contained in Table 3 in Section 4.2.7.

Changes to this value cause the Version attribute to be updated.

```
Top
 └ DX
    └ SourceServers
            └ <ServerName>
                     └ ServerType
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

4.4.1.4 ServerURL

This configuration attribute provides adequate information for the DX server to connect to the source server. The form of the URL is dependent on the ServerType.

Changes to this value cause the Version attribute to be updated.

```
Top
 └ DX
    └ SourceServers
            └ <ServerName>
                     └ ServerURL
```

DA Access Rights	Read-only
COM Data Type	VT_BSTR
XML DA Type	string

The syntax and an example for server URLs are shown below.

OPC XML-DA:	`<a real URL>`
OPC COM-DA:	"opcda://<ServerComputerName>/<OPCDA_Server_ProgID>/<OPCDA_Server_CLSID>"

Example:

Server Type	**URL Example**
OPC XML-DA:	"`http://opcfoundation.org/DX/ServerType/COM-DA2.05`"
OPC DA:	"opcda://Server24/OPC.Fix/{4210FF60-D373-11CE-B4B5-C46F03C10000}"

4.4.1.5 DefaultSourceServerConnected

This configuration attribute defines the default value for the SourceServerConnected status attribute of this source server (see Section 4.4.1.6.9). The value of this attribute is copied to that status attribute of the source server if and only if a value has not yet been assigned tothat status attribute.

Changes to this value cause the Version attribute to be updated.

```
Top
 └ DX
    └ SourceServers
           └ <ServerName>
                  └ DefaultSourceServerConnected
```

DA Access Rights	Read-only
COM Data Type	VT_BOOL
XML DA Type	boolean

4.4.1.6 Status

Status is a runtime attribute that the DX server automatically adds as a DA item below a source server when it adds the source server to the DX Database. Status is updated during runtime by the DX server. Its components are used to monitor and control the operational status of a source server. .It is not part of the source server complex item.

When this item is written using OPC DA, only SourceServerConnected is writable. All other elements are ignored by the DX server, and may be omitted in the Write request.

The Status attribute indicates the current status of the DX server's ability to access the source server.

```
Top
 └ DX
    └ SourceServers
          └ <ServerName>
                └ Status
```

DA Access Rights	Read-Write
COM Data Type	VT_BSTR that contains an XML document of the following DX specific XML Complex Type
XML DA Type	string that contains the following DX specific XML Complex Type
Data Type ID	” DXSourceServerStatus”

```
<s:complexType name="DXSourceServerStatus">
  <s:sequence>

    <s:element name="ConnectStatus" type="s0:connectState" />
    <s:element name="ErrorID" type="s0:OPCError" />
    <s:element name="ErrorDiagnostic" type="s:string" />
    <s:element name="LastConnectTimestamp" type="s:dateTime" />
    <s:element name="LastConnectFailTimestamp" type="s:dateTime" />
    <s:element name="ConnectFailCount" type="s:unsignedInt" />
    <s:element name="PingTime" type="s:unsignedInt" />
    <s:element name="LastDataChangeTimestamp" type="s:dateTime" />
    <s:element name="SourceServerConnected" type="s:boolean" />
  </s:sequence>
</s:complexType>
```

4.4.1.6.1 ConnectStatus

This status attribute indicates the state of the connection to the source server.

```
Top
 └ DX
    └ SourceServers
          └ <ServerName>
                └ Status
                   └ ConnectStatus
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR (See Table 7 for a list of the valid values.)
XML DA Type	string that contains the following DX specific XML Complex Type
XML ComplexType	connectStatus

```
<s:simpleType name="connectStatus">
  <s:restriction base="s:string">
    <s:enumeration value="connected" />
    <s:enumeration value="disconnected" />
    <s:enumeration value="connecting" />
    <s:enumeration value="failed" />
  </s:restriction>
</s:simpleType>
```

Table 7 – ConnectStatus Values

Value	Description
"connected"	The DX server is connected to the source server.
"disconnected"	The DX server is not connected to the source server.
"connecting"	The DX server is in the process of connecting to the source server.
"failed"	The DX server is not able to connect to the source server.

4.4.1.6.2 ErrorID

This status attribute indicates, when ConnectStatus equals "Failed", the reason for the failure. When ConnectStatus is not equal to "Failed", the value indicates success.

It is the responsibility of DX server implementor to define an appropriate conversion between COM error codes and XML qualified names if the interface technology used by the DX server is different than the technology used by the source server. In other words, the error may not necessarily be an error code returned from directly from the source server.

```
Top
 └ DX
    └ SourceServers
          └ <ServerName>
                └ Status
                   └ ErrorID
```

DA Access Rights	Read-Only
COM Data Type	VT_I4
XML DA Type	Qname

Table 8 – Source Server Error Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_SOURCE_SERVER_NOT_CONNECTED	See Section 5.1.7
E_SOURCE_SERVER_FAULT	See Section 5.1.7
E_SOURCE_SERVER_ NO_ACCESSS	See Section 5.1.7
E_SOURCE_SERVER_TIMEOUT	See Section 5.1.7

4.4.1.6.3 ErrorDiagnostic

This item contains the vendor specific additional detail for the ErrorID.

```
Top
 └ DX
    └ SourceServers
          └ <ServerName>
                └ Status
                   └ ErrorDiagnostic
```

DA Access Rights	Read-Only
COM Data Type	VT_BSTR
XML DA Type	string

4.4.1.6.4 LastConnectTimestamp

This status attribute indicates the time when the ConnectStatus transitions to “connected” (seeTable 7). The value is not cleared when the connection transitions out of ”connected”. The timestamp is generated by the DX server.

Top └ DX └ SourceServers └ <ServerName> └ Status └ **LastConnectTimestamp**	
DA Access Rights	Read-Only
COM Data Type	VT_DATE
XML DA Type	DateTime

4.4.1.6.5 LastConnectFailTimestamp

This status attribute indicates the time when the ConnectStatus transitions to “failed” (seeTable 7). The value is not cleared when the connection transitions out of ”failed”. The timestamp is generated by the DX server.

Top └ DX └ SourceServers └ <ServerName> └ Status └ **LastConnectFailTimestamp**	
DA Access Rights	Read-Only
COM Data Type	VT_DATE
XML DA Type	DateTime

4.4.1.6.6 ConnectFailCount

This status attribute indicates the number of times that the ConnectStatus transitions to “failed” (seeTable 7).

Top └ DX └ SourceServers └ <ServerName> └ Status └ **ConnectFailCount**	
DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	UnsignedInt

4.4.1.6.7 PingTime

This status attribute indicates the duration between Ping requests in milliseconds. Ping requests are sent periodically from the DX server to the source server to verify that an existing connection is working. The value of the PingTime is set by the DX Server as follows:

PingTime = 3 X smallest ActualUpdateRate value (see Section 4.3.2.19.11) associated with the source server.

When the calculated PingTime is less than two seconds, the DX server may use a value between, and including, the calculated PingTime and two seconds. That is, DX servers are not required to support PingTimes less than two seconds, but they may.

Top
└ DX
  └ SourceServers
    └ <ServerName>
      └ Status
        └ **PingTime**	

DA Access Rights	Read-Only
COM Data Type	VT_UI4
XML DA Type	UnsignedInt

4.4.1.6.8 LastDataChangeTimestamp

This status attribute indicates the last time, as calculated by the DX server, that the DX Server received item values from the source server. The timestamp received with the source data is not reflected by this timestamp.

Top
└ DX
  └ SourceServers
    └ <ServerName>
      └ Status
        └ **LastDataChangeTimestamp**	

DA Access Rights	Read-Only
COM Data Type	VT_DATE
XML DA Type	DateTime

4.4.1.6.9 SourceServerConnected

SourceServerConnected is a status attribute that is used to control the data flow from the source server to the DX server for all DXConnections. When TRUE, the DX server permits connections to be established to the source server.

When FALSE, the DX server is not permitted to access the source server, and data transfer on DXConnections from the source server does not occur.

When a source server is added to the DX Database, the value of this attribute is copied from the value of the source server's DefaultSourceServerConnected configuration attribute (see Section 4.4.1.5). Once initialized, OPC DA clients may update this attribute. DX clients can return the value of this attribute to its default value using the CopyDefaultServerAttributes service (see Section 5.2.1.5).

```
Top
 └ DX
    └ SourceServers
          └ <ServerName>
                └ Status
                   └ SourceServerConnected
```

DA Access Rights	Read/Write
COM Data Type	VT_BOOL
XML DA Type	Boolean

5. OPC DX Configuration Services

This section specifies a set of abstract OPC DX services that may be used concurrently by multiple clients to maintain the DX Database. Refer to Appendix Aand Appendix Bfor the mapping of these abstract services to specific interface technologies including Web Services and DCOM.

The DX server processes service requests individually as they are received, updating the DX Database and related runtime status attributes as necessary. The DX server responds to each request after updating the DX Database, if necessary. It does not wait for the runtime changes to take effect (see Section 6), or for the change to be persisted (see Section 5.3) before responding. When a request results in an update to the database, the update is visible to DX and DA clients immediately, as are the associated runtime status attributes.

For example, if the DX server receives a valid request to add a DXConnection with its DefaultSourceItemConnected attribute set to TRUE, the DX server adds the DXConnection to the database, returns the response, and creates and updates the status attributes. After returning the response, the DX server completes the processing required to subscribe to the source item and begins data transfer, updating the status attributes as necessary.

This model of operation allows clients to maintain the DX Database, independent of whether or not the source servers are online and operational. That is, DX server configuration behavior is the same whether they are offline or online. Clients, therefore, maintain the DX Database using the services defined in this section, and monitor the resulting runtime behavior of DXConnections using DA interfaces.

In the descriptions that follow, two types of parameters are defined, primitive and constructed. Primitive parameters contain a single parameter. Their descriptions contain their abstract data types.

Constructed parameters are composed of two or more component parameters. Their descriptions define their “use”. The following codes are defined for the “Use” column. See Appendix A and Appendix Bfor the definition of how optional and conditional parameters are implemented in each mapped interface technology.

Use	Description
M	Mandatory
O	Optional. The parameter may be provided by the client.
C	Conditional. The parameter is present only under the conditions specified in the parameter description
-	The parameter is not used.

5.1 Parameter Definitions

This section contains definitions of the parameter used in the DX Configuration services. They are presented alphabetically.

5.1.1 BrowsePath

This primitive parameter, defined in Table 9, provides a mechanism for configuration clients to organize branches. The DX server processes this parameter as specified.

Table 9 – BrowsePath Parameter

Parameter Name	Type	Description
BrowsePath	string	BrowsePath provides a mechanism for configuration clients to organize their DXConnections in the browse tree. It contains the hierarchical path of branch names used to locate a branch under the DXConnectionsRoot branch. The first branch name in the browse path is always the first branch under the DX/DXConnections branch. The standard delimiter '/' follows each branch name in the BrowsePath, except for the last branch name in the BrowsePath. Therefore, the '/' charactercannot be used in a branch name. When the BrowsePath is used in a DXConnection definition, it identifies the branch that contains the DXConnection. In the following example, the DXConnection definition for the following connection: "DX/DXConnectionsRoot/Client1/Loop1/MyDXConnection1", has a BrowsePath of: "Client1/Loop1". If the DXConnection is to appear directly under the DXConnectionsRoot branch, such as: "DX/DXConnectionsRoot/MyDXConnection", Then the BrowsePath is simply the standard delimeter: "/". When the BrowsePath is used in a query (e.g. UpdateQueriedConnections, DeleteConnections, QueryDXConnections), it identifies all DXConnections beneath the branch. In the following example, if the client targets a service to operate on all DXConnections below the following branch: "DX/DXConnectionsRoot/Client1/Loop1", the BrowsePath is: "Client1/Loop1". Similarly, if the client targets a service to operate on all DXConnections below the following branch: "DX/DXConnectionsRoot/Client1", which includes all DXConnections below "DX/DXConnectionsRoot/Client1/Loop1", the BrowsePath is: "Client1".

5.1.2 DXConnection

This constructed parameter contains the attributes of a DXConnection. It is used for two purposes, as a DXConnection definition in service requests and responses, and as a search mask in service requests only.

When used as a DXConnection definition, it contains the new or modified configuration attributes for a DXConnection. When this parameter is used to add a DXConnection, the client is required to supply values for all attributes, except as noted in Table 10 below. When this parameter is used to modify one or more DXConnections, the client supplies values for only the attributes that it wishes to update, except as noted in Table 10 below. When used in UpdateDXConnections (see Section 5.2.2.3), that is, when used to modify a set of DXConnections, only a subset of the attributes are applicable. Each attribute in this subset is identified in Table 10 below. Clients supply values for one or more of these attributes, depending on which ones are to be updated. The DX server uses only these attributes to update the DXConnections and ignores all other attributes. Appendix A and Appendix B specify, for each interface technology, how clients identify which attributes contain values that are to be used, and which attributes do not.

When used as a search mask, the client provides values for one or more of the DXConnection attributes. These attributes are used to select DXConnections from the DX Database. The client also identifies a branch in the DX Database where the search is to begin. The DX server iterates through the DXConnections under the branch, searching for DXConnections whose attributes match those in the search mask. For a match to occur, all attributes in the search mask must match their corresponding attributes in a DXConnection definition The DX server performs the operations on each selected DXConnection as specified by the service.

With respect to the BrowsePath in the context of the query operation, a regular expression in support of the functionality of a wildcard for the query for DXConnections. The wildcard syntax follows the syntax equivalent to the Visual Basic LIKE operator.

Table 10 provides the description of each attribute. The “Use” column indicates the usage of the attributes for the DX service requestsFor ModifyDXConnections requests, the ItemIdentifier is required, but its Version component is optional. All other attributes are optional as well, except that at least one is required.

When used in responses, all attributes are mandatory if they have values defined for the DXConnection.

Table 10 – DXConnection Parameter

Parameter Name	Use in				Description
	Add	Modify	Update	Query Mask	
ItemIdentifier	-	M	-	O	This attribute is ItemIdentifier of the DXConnection (see Section 5.1.5). Uses specific to the DXConnection parameter are defined below. For service requests, clients optionally provide this parameter as a mask attribute to identify a DXConnection. In this case, the Version component is optional. This parameter is not used in requests when the DXConnection parameter is used as a DXConnection definition. For service responses, this attribute is always present.
BrowsePath[1:N]	M	O	O	-	The attribute contains the browse paths (See Section 5.1.1) of the DXConnection. Each DXConnection may have one, or more browse paths defined for it. The browse path, '/', indicates that the DXConnection is located directly below the DXConnectionsRoot branch in the DX Database. When a DXConnection has more than one BrowsePath, the DA 2 Browse Up function moves up using the first BrowsePath (BrowsePath[0]). Therefore, clients should assign BrowsePath[0] carefully to ensure that the DA 2 Browse Up function works as expected. Rules for the syntax: No two or more successive '/' characters No control characters (0x00-0x1f) No blanks directly after '/'
DXConnectionName	M	O	-	O	This parameter contains the NodeName (see Section 5.1.6) of the DXConnection node in the DX Database. When used as a mask attribute, this attribute can be used to identify one or more DXConnections, because the same name can be used in different branches of the DX Database. When used in the AddDXConnections request, if the ConnectionName already exists under the specified browse path branch, then the DX server rejects the request.
Description	O	O	O	-	See Section 4.3.2.2.
Keyword	O	O	O	O	See Section 4.3.2.3.
DefaultSourceItemConnected	M	O	O	O	See Section 4.3.2.4.
DefaultTargetItemConnected	M	O	O	O	See Section 4.3.2.5.
DefaultOverridden	O	O	O	O	See Section 4.3.2.6. The default value for this attribute is false if it not set in the AddDXConnection request,
DefaultOverrideValue	O	O	O	O	See Section 4.3.2.7. This attribute is required in requests when DefaultOverridden is TRUE. This attribute is required in responses when it is defined for the DXConnection.
SubstituteValue	O	O	O	O	See Section 4.3.2.8. This value is manadory if default enable substitute value is true.
EnableSubstituteValue	O	O	O	O	The default value for this attribute is false if it not set in the AddDXConnection request,
TargetItemPath	C	-	-	O	See Section 4.3.2.9.

Parameter Name	Use in				Description
	Add	Modify	Update	Query Mask	
					This attribute may not be used as a DXConnection definition attribute in the ModifyDXConnection service. This attribute is present in requests only when it is needed to identify the target item. It is present in responses only when it is defined for the DXConnection. If this is used as a DXConnection definition attribute in a request, and the target item path specified by this attribute does not exist in the DA address space of the DX server, then the DX server rejects the request.
TargetItemName	M	-	-	O	See Section 4.3.2.11. If this is used as a DXConnection definition attribute in a request, and if a DXConnection already exists for the target item identified by this parameter, then the DX server rejects the request. Each target item may have only a single DXConnection defined for it.
SourceServerName	M	O	O	O	See Section 4.3.2.12. If this is used as a DXConnection definition attribute in a request, and if the source server identified by this parameter does not exist in the DX Database, then the DX server rejects the request.
SourceItemPath	C	O	O	O	See Section 4.3.2.13. This attribute is present in requests only when it is needed to identify the source item. It is present in responses only when it is defined for the DXConnection.
SourceItemName	M	O	O	O	See Section 4.3.2.14.
SourceItemQueueSize	O	O	O	O	See Section 4.3.2.15. The default value is 1 if it is not provided in the AddDXConnections request.
UpdateRate	O	O	O	O	See Section 4.3.2.16. The default value is 0 if it is not provided in the AddDXConnections request.
DeadBand	O	O	O	O	See Section 4.3.2.17.
VendorData	O	O	O	O	See Section 4.3.2.18. This attribute is present in responses if it is defined for the DXConnection.

5.1.3 DXGeneralResponse

This constructed parameter is the general DX response. The DX server processes each parameter individually as specified in Table 11.

Table 11 – DX General Response Parameter

Parameter Name	Use	Description
ConfigurationVersion	M	See Section 4.2.1
IdentifiedResult[1:N]	M	There is one IdentifiedResult (see Section 5.1.4) for each item in the associated request. The order of the results match the order of the items in the request.

5.1.4 IdentifiedResult

This constructed parameter contains the item identifier and the result code. The DX server processes each parameter individually as specified in Table 12.

Table 12 – IdentifiedResult Parameter

Parameter Name	Use	Description
ResultCode	M	See Section 5.1.7.
ItemIdentifier	M	See Section 5.1.5.

5.1.5 ItemIdentifier

This constructed parameter identifies an individual DXConnection or server in the DX Database. The DX server processes each parameter individually as specified in Table 13.

Table 13 – ItemIdentifier Parameter

Parameter Name	Use	Description
ItemPath	C	This string parameter is present only by OPC DX servers that support only an OPC XML DA interface, and then only if the OPC DX server uses it to identify items. OPC DX servers that support an OPC COM DA interface do not use this parameter. This parameter corresponds to the ItemPath parameter used in XML DA to qualify the ItemName parameter. XML DA clients obtain the ItemPath in the XML DA Browse response as the ItemPath parameter.
ItemName	M	This string parameter is the unique identifier for an item in the DX Database. In COM-DA, it is the ItemID, and in XML-DA, it is the ItemName. Whether or not the value of this parameter is based on the value of the NodeName (see Section 5.1.6) of the item is DX server dependent. Clients obtain the ItemName as follows: **COM-DA Clients:** In response to IOPCBrowseServerAddressSpace::GetItemID() **XML-DA Clients:** In response to XML DA Browse response. **DX Clients** In response to adding or querying for a Source Server or DXConnection.
Version	M/O	Version identifies the current version of the Source Server or DXConnection. See Section 4.3.2.1 and 4.4.1.1 for the definition of the specific version item in the DX Database. For requests, this parameter is optional. For responses, this parameter is required.

5.1.6 NodeName

This primitive parameter is defined in Table 14.

Table 14 – NodeName Parameter

Parameter Name	Type	Description
NodeName	string	The branch local browse name of an item in the DX Database. It does not contain the name of the branch that contains it.

5.1.7 ResultCode

This parameter is used in two contexts. In some cases, it describes the reason for an overall failure of the request. In XML implementations the result code is returned as part of a SOAP fault (see Appendix A). In DCOM implementations the result code is the method return type. In other cases, it describes the result of an operation on a single item.Table 15 contains a summary of the result codes defined for DX. Note that each result code is qualified with the DX namespace for XML implementations or prefixed with 'OPCDX_' for DCOM implementations.

Use	Description
F	Method level fault or error code.
I	Item level error code.
R	Runtime error code reported as part of the DX server, connection or sourcr server status.

Table 15 – OPC DX Specific Result Codes

Code	Use	Description

Code	Use	Description
S_OK	I	No error code set.
S_XXX	F/I /R	Vendor specific success code.
E_XXX	F/I /R	Vendor specific error.
E_FAIL	F	The operation failed.
E_OUTOFMEMORY	F	The operation could not be executed due to memory limitations.
E_PERSISTING	F	Could not process request because the configuration is currently being persisted.
E_NOITEMLIST	F	No item list was passed in the request.
E_SERVER_STATE	F/R	The operation failed because the server is in the wrong state.
E_VERSION_MISMATCH	I	The current object version does not match the specified version.
E_UNKNOWN_ITEM_PATH	I	The specified item path no longer exists in the DX server's address space.
E_UNKNOWN_ITEM_NAME	I	The specified item name no longer exists in the DX server's address space.
E_INVALID_ITEM_PATH	I	The specified item path does not conform to the DX server syntax.
E_INVALID_ITEM_NAME	I	The specified item name does not conform to the DX server syntax.
E_INVALID_NAME	I	The source server name or connection name does not conform to the server syntax.
E_DUPLICATE_NAME	I	The connection name or source server name is already in use.
E_INVALID_BROWSE_PATH	F/I	The browse path does not conform to the DX server syntax or it cannot be modified.
E_INVALID_SERVER_URL	I	The syntax of the source server URL is not correct.
E_INVALID_SERVER_TYPE	I	The source server type is not recognized.
E_UNSUPPORTED_SERVER_TYPE	I	The source server does not support the specified server type.
E_CONNECTIONS_EXIST	I	The source server cannot be deleted because connections exist.
E_TOO_MANY_CONNECTIONS	I	The total number of connections would exceed the maximum supported by the DX server.
E_OVERRIDE_BADTYPE	I	The override value is not valid and overridden flag is set to true.
E_OVERRIDE_RANGE	I	The override value is outside of the target item’s range and overridden flag is set to true.
E_SUBSTITUTE_BADTYPE	I	The substitute value is not valid and the enable substritute value flag is set to true.
E_SUBSTITUTE RANGE	I	The substitute value is outside of the target item’s range the enable substritute value flag is set to true.
E_INVALID_TARGET_ITEM	I/R	The target item does not conform to the DX server syntax or the item cannot be used as a target.
E_UNKNOWN_TARGET_ITEM	I/R	The target item no longer exists in the DX server’s address space.
E_TARGET_ALREADY_CONNECTED	I	The target item is already connected or may

Code	Use	Description
		not be changed in the method.
E_UNKNOWN_SERVER_NAME	I	The specified source server does not exist.
E_UNKNOWN_SOURCE_ITEM	I/R	The source item is no longer in the source server’s address space.
E_INVALID_SOURCE_ITEM	I/R	The source item does not confirm to the source server’s syntax.
E_INVALID_QUEUE_SIZE	I	The update queue size is not valid.
E_INVALID_DEADBAND	I	The deadband is not valid.
E_INVALID_CONFIG_FILE	R	The DX server configuration file is not acessable.
E_PERSIST_FAILED	R	Could not save the DX server configuration.
E_TARGET_FAULT	R	Target is online, but cannot service any request due to being in a fault state.
E_TARGET_NO_ACCESSS	R	Target is not online and may not be accessed.
E_SOURCE_SERVER_FAULT	R	Source server is online, but cannot service any request due to being in a fault state.
E_SOURCE_SERVER_NO_ACCESSS	R	Source server is not online and may not be accessed.
E_SOURCE_ITEM_BADRIGHTS	R	The source items access rights ddo not permit the operation.
E_SOURCE_ITEM_BAD_QUALITY	R	The source item value could be used because it has bad quality.
E_SOURCE_ITEM_BADTYPE	R	The source item cannot be converted to the target’s data type. This error reported by the source server.
E_SOURCE_ITEM_RANGE	R	The source item is out of the range for the requested type. This error reported by the source server.
E_SOURCE_SERVER_NOT_CONNECTED	R	The source server is not connected.
E_SOURCE_SERVER_TIMEOUT	R	The source server was disconnected because to failed to respond to pings.
E_SUBSCRIPTION_FAULT	R	Source server is connected, however it could not create a subscription for the connection.
E_TARGET_ITEM_DISCONNECTED	R	The target item is disconnected.
E_TARGET_NO_WRITES_ATTEMPTED	R	The DX server has not attempted to write to the target.
E_TARGET_ITEM_BADTYPE	R	The target item cannot be converted to the target’s data type.
E_TARGET_ITEM_RANGE	R	The target item is outside the value range for the item.
S_TARGET_SUBSTITUTED	R	The substitute value was written to the target.
S_TARGET_OVERRIDEN	R	The override value was written to the target.
S_CLAMP	R	Value written was accepted but the output was clamped.

5.1.8 Source Server

This constructed parameter contains the attributes of a source server. It is used for two purposes, as a source server definition in service requests and responses, and as a search mask in service requests only.

When used as a source server definition, it contains the new or modified configuration attributes for a source server. When this parameter is used to add a source server, the client is required to supply values for all attributes, except as noted in Table 16 below. When this parameter is used to modify one or more source servers, the client supplies values for only the attributes that it wishes to update, except as noted in Table 16 below.

When used as a search mask, the client provides values for one or more of the source server attributes. These attributes are used to select source servers from the DX Database. The DX server iterates through all of the source servers, searching for servers whose attributes match those in the search mask. For a match to occur, all attributes in the search mask must match their corresponding attributes in a server definition. If all match, the DX server adds the server to the list of selected servers.

Table 16 provides the description each attribute. The "Use" column indicates the usage of the attributes for AddServer Service requests.

For ModifyServers requests, ItemIdentifier is required, but its Version component is optional. All other attributes are optional as well, except that at least one is required.

When used in responses, all attributes are mandatory if they have values defined for the server.

Table 16 – Server Parameter

Parameter Name	Use	Description
ItemIdentifier	-	This attribute is the ItemIdentifier of the server (see Section 5.1.5). For AddServers service requests, this attribute is not used. For all service responses, this attribute is always present.
Name	M	This parameter identifies the source server used for source items. It contains the NodeName (See Section 5.1.6) of the source server item to be added to the DX Database. See Section 4.4 for a description of source servers. If this name is invalid or not unique within the DX server, then the DX server rejects the request.
Description	O	See Section 4.4.1.2.
ServerType	M	See Section 4.4.1.3. If the ServerType parameter identifies a server type not supported by the DX server, then the DX server rejects the request.
ServerURL	M	See Section 4.4.1.4. If the DX server is able to determine that the URL is invalid or is already defined for another source server node, then the DX server rejects the request.
DefaultSourceServerConnected	M	See Section 4.4.1.5.

5.2 OPC DX Services

This section defines the OPC DX services used to maintain the configuration data in the DX Database. When one of these services is used that results in a change to the DX Database, the ConfigurationVersion attribute (see Section 4.2.1) of the DX server is updated and the DirtyFlag attribute is set (see Section 4.2.5).

5.2.1 Source Server Services

5.2.1.1 GetServers

This service is used by DX configuration clients to obtain the definitions of all servers in the DX Database.

This service has no effect on the DXConnections or on the status/version of the DX Database.

5.2.1.1.1 GetServers Request

This request has no parameters.

5.2.1.1.2 GetServers Response

This response contains the results of the requested service. There is one Server parameter in the response for each server in the DX Database. The parameters of the response are described in Table 17.

The response is returned after the service was successfully or unsuccessfully completed. Partial success indicates that not all server definitions were returned.

Table 17 – GetServers Response Parameters

Parameter Name	Use	Description
Server[0:N]	M	This parameter contains a list of server definitions (See Section 5.1.8.)

Table 18 – GetServers Fault Codes

Code	Description
`E_XXX`	See Section 5.1.7
`E_FAIL`	See Section 5.1.7
`E_OUTOFMEMORY`	See Section 5.1.7
`E_PERSISTING`	See Section 5.1.7
`E_NOITEMLIST`	See Section 5.1.7
`E_SERVER_STATE`	See Section 5.1.7

5.2.1.2 AddServers

This service is used by DX configuration clients to add one or more servers to the DX Database.

5.2.1.2.1 AddServers Request

This request contains parameters that define the servers to be added. Upon receipt of the request, the DX server attempts to add each source server node specified in the request to the DX Database. The parameters of the request are described in Table 19.

Table 19 – Add Servers Request Parameters

Parameter Name	Use	Description
Server[1:N]	M	This parameter contains a list of servers to be added. See Section 5.1.8 for a detailed description of the servers in the list.

5.2.1.2.2 AddServers Response

This response contains the results of the requested service. There is one IdentifiedResult parameter in the response for each server that the DX server attempted to add. The parameters of the response are described in Section 5.1.3.

The response is returned after the DX server completes its attempts to add servers and update the appropriate DX server status attributes (see Section 4.2).

Table 20 – AddServers Fault Codes

Code	Description
`E_XXX`	See Section 5.1.7
`E_FAIL`	See Section 5.1.7
`E_OUTOFMEMORY`	See Section 5.1.7
`E_PERSISTING`	See Section 5.1.7
`E_NOITEMLIST`	See Section 5.1.7
`E_SERVER_STATE`	See Section 5.1.7

Table 21 – AddServers Item Result Codes

Code	Description
`S_OK`	See Section 5.1.7
`E_XXX`	See Section 5.1.7
`E_INVALID_NAME`	See Section 5.1.7
`E_DUPLICATE_NAME`	See Section 5.1.7
`E_INVALID_SERVER_URL`	See Section 5.1.7
`E_INVALID_SERVER_TYPE`	See Section 5.1.7
`E_UNSUPPORTED_SERVER_TYPE`	See Section 5.1.7

5.2.1.3 ModifyServers

This service is used by DX configuration clients to modify servers in the DX Database. The client may specify one or more servers to be modified, using the ServerDefinitions parameter. When more than one is requested to be modified, the DX server modifies the servers in the order that they appear in the request.

Each ServerDefinition contains the ItemIdentifier (see Section 5.1.5) of the server to be modified. Each optionally contains the Version of the server. When the Version is present, it must match the Version of the specified server in the DX Database for the update to be applied.

Source server attributes define how the DX server accesses the source server. Modifying them means that either the source server is being relocated on the network and/or that the DX server should use a different interface to access the source server.

When servers are modified, data transfers may be disrupted while the modifications are being applied. To reduce loss of data on DXConnections, the DX server may:

1. open a new interface instance to the source server before it closes the first (the existing one),
2. move all DXConnections associated with the source server to the new interface instance, and
3. close the first interface instance .

However, this procedure may be too complicated for all DX servers to implement, and is, therefore, not required.

When source server names are modified, the DX server is permitted to change the ItemID of the source server, and therefore status and monitoring clients may have to resubscribe to this item.

If the DX server is not able to complete the transition from the existing interface instance to the new one, then it updates the source server Status runtime attribute (see Section 4.4.1.6) to indicate the problem.

5.2.1.3.1 ModifyServers Request

The parameters of the request are described in Table 22.

Table 22 – Modify Servers Request Parameters

Parameter Name	Use	Description
ServerDefinitions[1:N]	M	This parameter contains the server definition attributes to be used to update the selected server (See Section 5.1.8).

5.2.1.3.2 ModifyServers Response

This response contains the results of the requested service. There is one IdentifiedResult in the response for each Update in the request. The parameters of the response are described in Section 5.1.3.

The response is returned after the DX server completes its attempts to modify servers and update the appropriate DX server status attributes (see Section 4.2).

Table 23 – ModifyServers Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7

Table 24 – ModifyServers Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_VERSION_MISMATCH	See Section 5.1.7
E_UNKNOWN_ITEM_PATH	See Section 5.1.7
E_UNKNOWN_ITEM_NAME	See Section 5.1.7
E_INVALID_ITEM_PATH	See Section 5.1.7
E_INVALID_ITEM_NAME	See Section 5.1.7
E_INVALID_NAME	See Section 5.1.7
E_DUPLICATE_NAME	See Section 5.1.7
E_INVALID_SERVER_URL	See Section 5.1.7
E_INVALID_SERVER_TYPE	See Section 5.1.7
E_UNSUPPORTED_SERVER_TYPE	See Section 5.1.7

5.2.1.4 DeleteServers

This service is used by DX configuration clients to delete one or more servers in the DX Database.

The client identifies each server to be deleted by specifying its ItemIdentifier (see Section 5.1.5). Each optionally contains the Version of the server. When the Version is present, it must match the Version of the specified server item in the DX Database for the server to be deleted.

Clients also may request that all servers are to be deleted, by omitting the ItemIdentifier parameter.

5.2.1.4.1 DeleteServers Request

This request contains parameters that identify the servers to be deleted. The parameters of the request are described in Table 25.

Table 25 – DeleteServers Request Parameters

Parameter Name	Use	Description
ItemIdentifier[0:N]	O	This parameter contains a list of identifiers for the servers to be deleted (see Section 5.1.5). If this parameter is omitted, then all servers are to be deleted.

5.2.1.4.2 DeleteServers Response

This response contains the results of the requested service. There is one IdentifiedResult parameter in the response for each server that the DX server attempted to delete. The parameters of the response are described in Section 5.1.3.

The response is returned after the DX server completes its attempts to delete servers and update the appropriate DX server status attributes (see Section 4.2).

Table 26 – DeleteServers Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7

Table 27 – DeleteServers Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_CONNECTIONS_EXIST	See Section 5.1.7
E_VERSION_MISMATCH	See Section 5.1.7
E_UNKNOWN_ITEM_PATH	See Section 5.1.7
E_UNKNOWN_ITEM_NAME	See Section 5.1.7
E_INVALID_ITEM_PATH	See Section 5.1.7
E_INVALID_ITEM_NAME	See Section 5.1.7

5.2.1.5 CopyDefaultServerAttributes

This service is used by DX configuration clients to copy the default configuration attributes to their corresponding runtime status attributes, or vice-versa, for one or more source servers in the DX Database. These attributes are shown in Table 28 below.

Table 28 – Default Configuration Server Attributes and their Status Counterparts

Default Configuration Server Attribute	Corresponding Status Attribute
DefaultSourceServerConnected	SourceServerConnected

The client specifies the direction of the copy using the ConfigToStatus parameter. This parameter indicates whether the copy is to be performed from the configuration attributes to the status attributes or vice-versa.

In the status direction, the default configuration attributes are copied to the status attributes. Copies in this direction return the status attributes to their default values.

In the configuration direction, the status attribute values are copied to the default configuration attribute values. Copies in this direction update the configuration attributes with the existing values of their corresponding status attributes.

The client identifies each server by specifying its ItemIdentifier (see Section 5.1.5). Each optionally contains the Version of the server. When the Version is present, then it must match the Version of the specified server server in the DX Database for the attributes of the server to be copied.

5.2.1.5.1 CopyDefaultServerAttributes Request

This request contains parameters that identify the DXConnections whose attributes are to be copied. The parameters of the request are described in Table 29.

Table 29 – CopyDefaultServerAttributes Request Parameters

Parameter Name	Use	Description
ConfigToStatus	M	This boolean parameter indicates the direction of the copy operation, as follows: TRUE configuration to status FALSE status to configuration
ItemIdentifier[0:N]	O	This parameter contains a list of identifiers for the servers whose attributes are to be copied (see Section 5.1.5). If this parameter is omitted, then the attributes of all servers are to be copied.

5.2.1.5.2 CopyDefaultServerAttributes Response

This response contains the results of the requested service. There is one IdentifiedResult parameter in the response for each Source Server that the DX server attempted to copy. The parameters of the response are described in Section 5.1.3.

The response is returned after the DX server completes its attempts to copy DXConnection attributes and update the appropriate DX server status attributes (see Section 4.2). However, the DX server does not wait for the runtime behavior associated with changing runtime status attributes to complete.

Table 30 – CopyDefaultServerAttributes Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7

Table 31 – CopyDefaultServerAttributes Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_VERSION_MISMATCH	See Section 5.1.7
E_UNKNOWN_ITEM_PATH	See Section 5.1.7
E_UNKNOWN_ITEM_NAME	See Section 5.1.7
E_INVALID_ITEM_PATH	See Section 5.1.7
E_INVALID_ITEM_NAME	See Section 5.1.7

5.2.2 DXConnection Services

5.2.2.1 QueryDXConnections

This service is used by DX configuration clients to find one or more DXConnections under a specific branch in the DX Database. The BrowsePath parameter is used to specify this branch. If the BrowsePath parameter is “/”, then the entire database is searched.

The DXConnectionMasks parameter contains the query criteria. It consists of a list of DXConnection masks (see Section 5.1.2), each containing only the values of the attributes, including their ItemIdentifier, to be used in the search. See Appendix A and Appendix B for the definition of how attributes are identified for the mask in each mapped interface technology..

Each mask works as the logical AND of its attribute values. For the DX server to “find” a DXConnection, all attributes in the mask, including the ItemIdentifier (see Section 5.1.5) and BrowsePath (see Section 5.1.1), if present, have to match their corresponding attributes in the DXConnection. If any do not match, the DX server does not select the DXConnection. For example, if an ItemIdentifier and a BrowsePath are provided, both have to match the corresponding attributes of a connection for the connection to be selected.

The list of masks works as the logical OR between masks. A DXConnection that is returned as the result of the query will have been selected by at least one of the masks. That is, each DXConnection returned will have been selected by $mask_1$ OR $mask_2$ … OR $mask_n$. Stated another way, the DX server returns all DXConnections selected by $mask_1$, and all DXConnections selected by $mask_2$, and in general, all DXConnections selected by $mask_k$.

When the DXConnectionMasks parameter is empty, all DXConnections in the segment of the DX Database to be searched, as specified by the browse parameter, are returned.

This service has no effect on the DXConnections or on the status/version of the DX Database.

5.2.2.1.1 QueryDXConnections Request

This request contains parameters that specify the query criteria. The parameters of the request are described in Table 32.

Table 32 – QueryDXConnections Request Parameters

Parameter Name	Use	Description
BrowsePath	M	This parameter identifies the branch where the query is to begin. If the BrowsePath parameter is "/", then the search begins at the root of the connection branch. If the BrowsePath is the only criterion specified, then all DXConnections at the level specified by this parameter are selected.
DXConnectionMasks[1:N]	O	This parameter contains a list of DXConnection masks (see Section 5.1.2. For each only the attributes to be used in the search are specified.
Recursive	O	Boolean with a default value of FALSE. If TRUE, recurses down the browse tree from the starting point specified in the BrowsePath. If FALSE, only those items that match the DXConnectionMasks at the BrowsePath are returned.

5.2.2.1.2 QueryDXConnections Response

This response contains the results of the requested service. There is one ResultCode parameter in the response, and one DXConnection definition parameter for each DXConnection that was located. The parameters of the response are described in Table 33.

The response is returned after the service was successfully or unsuccessfully completed.

If any of the DXConnectionMasks are invalid, an error is returned indicating one of more of the DXConnectionMasks are invalid, and the Errors indicates the status of each of the DXConnectionMasks, and no DXConnections will be returned. The DX server will either return the DXConnection or Errors but never both.

Table 33 – QueryDXConnections Response Parameters

Parameter Name	Use	Description
DXConnection[0:N]	O	This parameter contains a list of DXConnection definitions (See Section 5.1.2)
MaskErrors[0:N]	O	This parameter contains the status of the individual DXConnectionMasks

Table 34 – QueryDXConnections Fault Codes

Code	Description
`E_XXX`	See Section 5.1.7
`E_FAIL`	See Section 5.1.7
`E_OUTOFMEMORY`	See Section 5.1.7
`E_PERSISTING`	See Section 5.1.7
`E_NOITEMLIST`	See Section 5.1.7
`E_SERVER_STATE`	See Section 5.1.7
`E_INVALID_BROWSE_PATH`	See Section 5.1.7

Table 35 – QueryDXConnections Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7. Returned only as a query mask error.
E_VERSION_MISMATCH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.

5.2.2.2 AddDXConnections

This service is used by DX configuration clients to add one or more DXConnections to the DX Database.

5.2.2.2.1 AddDXConnections Request

This request contains parameters that define the DXConnections to be added. The parameters of the request are described in Table 36.

Table 36 – AddDXConnections Request Parameters

Parameter Name	Use	Description
DXConnection[1:N]	M	This parameter contains a list of DXConnections (see Section 5.1.2).

5.2.2.2.2 AddDXConnections Response

This response contains the results of the requested service. There is one IdentifiedResult parameter in the response for each DXConnection that the DX server attempted to add. The parameters of the response are described in Section 5.1.3.

The client cannot assume that a successfully added DXConnection means that it can be established at runtime. Clients may monitor DXConnections using OPC DA interfaces to determine their runtime state.

The response is returned after the DX server completes its attempts to add DXConnections and update the appropriate DX server status attributes (see Section 4.2).

Table 37 – AddDXConnections Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7

Table 38 – AddDXConnections Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_TOO_MANY_CONNECTIONS	See Section 5.1.7
E_INVALID_NAME	See Section 5.1.7
E_DUPLICATE_NAME	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7
E_OVERRIDE_BADTYPE	See Section 5.1.7
E_OVERRIDE_RANGE	See Section 5.1.7
E_SUBSTITUTE_BADTYPE	See Section 5.1.7
E_SUBSTITUTE_RANGE	See Section 5.1.7
E_INVALID_TARGET_ITEM	See Section 5.1.7
E_UNKNOWN_TARGET_ITEM	See Section 5.1.7
E_TARGET_ALREADY_CONNECTED	See Section 5.1.7
E_UNKNOWN_SERVER_NAME	See Section 5.1.7
E_UNKNOWN_SOURCE_ITEM	See Section 5.1.7
E_INVALID_SOURCE_ITEM	See Section 5.1.7
E_INVALID_QUEUE_SIZE	See Section 5.1.7
E_INVALID_DEADBAND	See Section 5.1.7

5.2.2.3 UpdateDXConnections

This service is used by DX configuration clients to search for and modify one or more DXConnections under a specific branch in the DX Database. Clients use the BrowsePath parameter to specify this branch. If the BrowsePath parameter is "/", then the entire database is searched.

Clients also specify a DXConnectionMask that is used by the DX Server to select one or more connections to be updated. Its use is specified in the QueryDXConnections service (see Section 5.2.2.1). The DXConnectionMask parameter may be empty. In this case, the DXConnection attributes are applied to all DXConnections below the branch specified by the BrowsePath parameter.

Clients also may select an entire branch to be updated by supplying only the BrowsePath parameter.

Clients use the DXConnectionDefinition parameter to provide values for one or more attributes to be updated in each of the selected connections. Each value overwrites the corresponding attribute value in the DXConnections selected by the DXConnectionMasks parameter. Not all DXConnection attributes can be modified using this service. Section 5.1.2 specifies those that are.

The DX server must not delete a DXConnection on a failure to modify it. If it fails to modify the DXConnection in the way the client has requested it shall maintain the DXConnection as it was previous to the request and return an appropriate ErrorID.

5.2.2.3.1 UpdateDXConnections Request

The parameters of the request are described in Table 39.

Table 39 – Update DXConnections Request Parameters

Parameter Name	Use	Description
BrowsePath	M	This parameter identifies the branch where the query is to begin. If the BrowsePath parameter is "/", then the entire database is searched. If the BrowsePath is the only criterion specified, then all DXConnections below the branch specified by this parameter are selected.
DXConnectionMasks[0:M]	O	This parameter contains a list of DXConnection masks that identify the DXConnections to be modified (See Section 5.1.2 and Section 5.2.2.1). If this parameter is empty, then all DXConnections below the identified path are selected for modification. If this parameter is invalid, or the DX server cannot locate at least one DXConnection in its DX Database, an error code is returned in the response.
Recursive	O	Boolean with a default value of FALSE. If TRUE, recurses down the browse tree from the starting point specified in the BrowsePath. If FALSE, only those items that match the DXConnectionMasks at the BrowsePath are returned.
DXConnectionDefinition	M	This parameter contains the server definition attributes to be used to update the selected server (See Section 5.1.2).

Error return – there will be an IdentifiedResult for each DXConnection that matched the mask, and the IdentifiedResult will indicate success or failure

5.2.2.3.2 UpdateDXConnections Response

If any of the DXConnectionMasks are invalid, the service fails and no DXConnections are updated. In this case, only the MasksError parameter is returned. The MasksError parameter indicates the status of each of the DXConnectionMasks. Its values are described in Table 40.

If there were no DXConnectionMask errors, then the DX server returns only the GeneralResponse parameter. This parameter contains the new ConfigurationVersion and identifies the DXConnections that were updated. If no DXConnections were updated, the GeneralResponse contains only the ConfigurationVersion, which was not updated by this service invocation.

The response is returned after the service was successfully or unsuccessfully completed.

Table 40 – UpdateDXConnections Response Parameters

Parameter Name	Use	Description
GeneralResponse	M	There is one IdentifiedResult (see Section 5.1.4) for each item that was selected for update. This parameter indicates to the client that DX Connections were updated, and for which an error occurred. The parameters of the General Response are described in Section 5.1.3.
MaskErrors[0:N]	O	This parameter contains the status of the individual DXConnectionMasks. The valid values are shown in Table 41 and Table 42.

Table 41 – UpdateDXConnections Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7

Table 42 – UpdateDXConnections Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7. Returned only as a query mask error.
E_VERSION_MISMATCH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_NAME	See Section 5.1.7
E_DUPLICATE_NAME	See Section 5.1.7
E_OVERRIDE_BADTYPE	See Section 5.1.7
E_OVERRIDE_RANGE	See Section 5.1.7
E_SUBSTITUTE_BADTYPE	See Section 5.1.7
E_SUBSTITUTE_RANGE	See Section 5.1.7
E_UNKNOWN_SERVER_NAME	See Section 5.1.7
E_UNKNOWN_SOURCE_ITEM	See Section 5.1.7
E_INVALID_SOURCE_ITEM	See Section 5.1.7
E_INVALID_QUEUE_SIZE	See Section 5.1.7
E_INVALID_DEADBAND	See Section 5.1.7

5.2.2.4 ModifyDXConnections

This service is used by DX configuration clients to modify DXConnections in the DX Database. The client may specify one or more DXConnections to be modified, using the DXConnectionDefinition parameter. Each DXConnectionDefinition in the list identifies a connection to be modified and provides replacement values for its attributes. Only the changed attributes need to be included in the DXConnectionDefinition. When more than one is requested to be modified, the DX server modifies the DXConnections in the order that they appear in the request.

Each DXConnectionDefinition contains the ItemIdentifier (see Section 5.1.5) of the DXConnection to be modified. The ItemIdentifier optionally contains the Version of the DXConnection. When the Version is present, it must match the Version of the specified DXConnection in the DX Database for the update to be applied. The remaining attributes of the DXConnectionDefinition provide replacement values.

The DX server must not delete a DXConnection on a failure to modify it. If it fails to modify the DXConnection in the way the client has requested it shall maintain the DXConnection as it was previous to the request and return an appropriate ErrorID.

To modfiy a DXConnection, the DX server:

1. changes the DXConnection in the DX server's address space,
2. updates the ConfigurationVersion (see Section 4.2.1),
3. sets the DirtyFlag (see Section 4.2.5), and
4. executes the runtime behavior associated with modifying a DXConnection (see Section 6.3.1.4).

5.2.2.4.1 ModifyDXConnections Request

The parameters of the request are described in Table 43.

Table 43 – ModifyDXConnections Request Parameters

Parameter Name	Use	Description
DXConnectionDefinition[1:N]	M	This parameter contains the DXConnection definition attributes to be used to identify and update one or more DXConnections (See Section 5.1.2).

5.2.2.4.2 ModifyDXConnections Response

This response contains the results of the requested service. There is one IdentifiedResult in the response for each Update in the request. The parameters of the response are described in Section 5.1.3.

The response is returned after the DX server completes its attempts to modify servers and update the appropriate DX server status attributes (see Section 4.2).

Table 44 – ModifyDXConnections Fault Codes

Code	Description
`E_XXX`	See Section 5.1.7
`E_FAIL`	See Section 5.1.7
`E_OUTOFMEMORY`	See Section 5.1.7
`E_PERSISTING`	See Section 5.1.7
`E_NOITEMLIST`	See Section 5.1.7
`E_SERVER_STATE`	See Section 5.1.7

Table 45 – ModifyDXConnections Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_VERSION_MISMATCH	See Section 5.1.7
E_UNKNOWN_ITEM_PATH	See Section 5.1.7
E_UNKNOWN_ITEM_NAME	See Section 5.1.7
E_INVALID_ITEM_PATH	See Section 5.1.7
E_INVALID_ITEM_NAME	See Section 5.1.7
E_INVALID_NAME	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7
E_DUPLICATE_NAME	See Section 5.1.7
E_OVERRIDE_BADTYPE	See Section 5.1.7
E_OVERRIDE_RANGE	See Section 5.1.7
E_SUBSTITUTE_BADTYPE	See Section 5.1.7
E_SUBSTITUTE_RANGE	See Section 5.1.7
E_UNKNOWN_SERVER_NAME	See Section 5.1.7
E_UNKNOWN_SOURCE_ITEM	See Section 5.1.7
E_INVALID_SOURCE_ITEM	See Section 5.1.7
E_INVALID_QUEUE_SIZE	See Section 5.1.7
E_INVALID_DEADBAND	See Section 5.1.7

5.2.2.5 DeleteDXConnections

This service is used by DX configuration clients to search for and delete one or more DXConnections under a specific branch in the DX Database. Clients use the BrowsePath parameter to specify this branch. If the BrowsePath parameter is "/", then the entire database is searched.

Clients also specify a DXConnectionMask that is used by the DX Server to select one or more connections to be deleted. Its use is specified in the QueryDXConnections service (see Section 5.2.2.1). The DXConnectionMask parameter may be empty. In this case, the DXConnection attributes are applied to all DXConnections below the branch specified by the BrowsePath parameter.

Clients also may select an entire branch to be deleted by supplying only the BrowsePath parameter. In this case, all branches and DXConnections below it are deleted. Logically, the deletions take place in the reverse order, with the DXConnections being deleted first, followed by the branches.

To delete a DXConnection, the DX server:

1. removes the DXConnection from the DX server's address space,
2. updates the DXConnectionCount (see Section 4.2.3)
3. updates the ConfigurationVersion (see Section 4.2.1),
4. sets the DirtyFlag (see Section 4.2.5), and
5. executes the runtime behavior associated with deleting a DXConnection (see Section 6.3.1.4).

5.2.2.5.1 DeleteDXConnections Request

This request contains parameters that identify the DXConnections to be deleted. The parameters of the request are described in Table 46.

Table 46 – DeleteDXConnections Request Parameters

Parameter Name	Use	Description
BrowsePath	M	This parameter identifies the branch where the query is to begin. If the BrowsePath parameter is "/", then the entire database is searched. If the BrowsePath is the only criterion specified, then all DXConnections below the branch specified by this parameter are selected.
DXConnectionMasks[0:M]	O	This parameter contains a list of DXConnection masks that identify the DXConnections to be deleted (See Section 5.1.2 and Section 5.2.2.1). If this parameter is empty, then all DXConnections below the identified path are selected for deletion. If this parameter is invalid, or the DX server cannot locate at least one DXConnection in its DX Database, an error code is returned in the response.
Recursive	O	Boolean with a default value of FALSE. If TRUE, recurses down the browse tree from the starting point specified in the BrowsePath. If FALSE, only those items that match the DXConnectionMasks at the BrowsePath are returned.

5.2.2.5.2 DeleteDXConnections Response

If any of the DXConnectionMasks are invalid, the service fails and no DXConnections are deleted. In this case, only the MasksError parameter is returned. The MasksError parameter indicates the status of each of the DXConnectionMasks. Its values are described in Table 47.

If there were no DXConnectionMask errors, then the DX server returns only the GeneralResponse parameter. This parameter contains the new ConfigurationVersion and identifies the DXConnections that were deleted. If no DXConnections were deleted, the GeneralResponse contains only the ConfigurationVersion, which was not updated by this service invocation.

The response is returned after the DX server completes its attempts to delete DXConnections and update the appropriate DX server status attributes (see Section 4.2).

Table 47 – DeleteDXConnections Response Parameters

Parameter Name	Use	Description
GeneralResponse	M	There is one IdentifiedResult (see Section 5.1.4) for each item that was selected for deletion. This parameter indicates to the client that DX Connections were deleted, and for which an error occurred. The parameters of the General Response are described in Section 5.1.3.
MaskErrors[0:N]	O	This parameter contains the status of the individual DXConnectionMasks. The valid values are shown in Table 48 and Table 49.

Table 48 – DeleteDXConnections Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7

Table 49 – DeleteDXConnections Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7. Returned only as a query mask error.
E_VERSION_MISMATCH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.

5.2.2.6 CopyDefaultDXConnectionAttributes

This service is used by DX configuration clients to perform copies between default configuration DXConnection attributes and their corresponding runtime status attributes for one or more DXConnections in the DX Database. These attributes are shown in Table 50 below.

Table 50 – Default Configuration Attributes and their Status Counterparts

Default DXConnection Configuration Attribute	Corresponding Status Attribute
DefaultSourceItemConnected	SourceItemConnected
DefaultTargetItemConnected	TargetItemConnected
DefaultOverridden	Overridden
DefaultOverrideValue	OverrideValue

The client specifies the direction of the copy using the ConfigToStatus parameter. This parameter indicates whether the copy is to be performed from the configuration attributes to the status attributes or vice-versa.

In the status direction, the default configuration attributes are copied to the status attributes. Copies in this direction return the status attributes to their default values.

In the configuration direction, the status attribute values are copied to the default configuration attribute values. Copies in this direction update the configuration attributes with the existing values of their corresponding status attributes.

The client may specify that copies are to be performed for one or more DXConnections. The BrowsePath and DXConnectionMasks parameters are used by the DX server to select the DXConnections for the copies. Their use is specified in the QueryDXConnections service (see Section 5.2.2.1).

5.2.2.6.1 CopyDefaultDXConnectionAttributes Request

This request contains parameters that identify the DXConnections to be copied. The parameters of the request are described in Table 51.

Table 51 – CopyDefaultDXConnectionAttributes Request Parameters

Parameter Name	Use	Description
ConfigToStatus	M	This boolean parameter indicates the direction of the copy operation, as follows: TRUE configuration to status FALSE status to configuration
BrowsePath	M	This parameter identifies the branch where the query is to begin. If the BrowsePath parameter is "/", then the entire database is searched. If the BrowsePath is the only criterion specified, then all DXConnections below the branch specified by this parameter are selected.
Recursive	O	Boolean with a default value of FALSE. If TRUE, recurses down the browse tree from the starting point specified in the BrowsePath. If FALSE, only those items that match the DXConnectionMasks at the BrowsePath are returned.
DXConnectionMasks[0:M]	O	This parameter contains a list of DXConnection masks that identify the DXConnections to be modified (See Section 5.1.2 and Section 5.2.2.1). If this parameter is empty, then all DXConnections below the identified path are selected for modification. If this parameter is invalid, or the DX server cannot locate at least one DXConnection in its DX Database, an error code is returned in the response.

5.2.2.6.2 CopyDefaultDXConnectionAttributes Response

If any of the DXConnectionMasks are invalid, the service fails and no DXConnection attributes are copied. In this case, only the MasksError parameter is returned. The MasksError parameter indicates the status of each of the DXConnectionMasks. Its values are described in Table 52.

If there were no DXConnectionMask errors, then the DX server returns only the GeneralResponse parameter. This parameter contains the new ConfigurationVersion and identifies the DXConnections whose atttributes were copied. If no DXConnection attributes were copied, the GeneralResponse contains only the ConfigurationVersion, which was not updated by this service invocation.

The response is returned after the DX server completes its attempts to copy DXConnection attributes and update the appropriate DX server status attributes (see Section 4.2). However, the DX server does not wait for the runtime behavior associated with changing runtime status attributes to complete.

Table 52 – CopyDefaultDXConnectionAttributes Response Parameters

Parameter Name	Use	Description
GeneralResponse	M	There is one IdentifiedResult (see Section 5.1.4) for each item that was selected for the copy. This parameter identifies to the client the DX Connections for which the copy succeeded, and for which an error occurred. The parameters of the General Response are described in Section 5.1.3.
MaskErrors[0:N]	O	This parameter contains the status of the individual DXConnectionMasks. The valid values are shown in Table 41 and Table 42.

Table 53 – CopyDefaultDXConnectionAttributes Fault Codes

Code	Description
E_XXX	See Section 5.1.7
E_FAIL	See Section 5.1.7
E_OUTOFMEMORY	See Section 5.1.7
E_PERSISTING	See Section 5.1.7
E_NOITEMLIST	See Section 5.1.7
E_SERVER_STATE	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7

Table 54 – CopyDefaultDXConnectionAttributes Item Result Codes

Code	Description
S_OK	See Section 5.1.7
E_XXX	See Section 5.1.7
E_INVALID_BROWSE_PATH	See Section 5.1.7. Returned only as a query mask error.
E_VERSION_MISMATCH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_UNKNOWN_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_PATH	See Section 5.1.7. Returned only as a query mask error.
E_INVALID_ITEM_NAME	See Section 5.1.7. Returned only as a query mask error.

5.2.3 ResetConfiguration

This service is used by DX configuration clients to delete all DXConnections and servers from the DX Database. This returns the DX server to its “factory defaults” state, except for the ConfigurationVersion (see Section 4.2.1), which is updated.

To delete all configured items, the client may provide the ConfigurationVersion to ensure that the expected configuration is being deleted. If the ConfigurationVersion parameter is not present, or if it is present and its value matches the current ConfigurationVersion of the DX Database, then the DX server deletes all DXConnections using the procedures defined in Section 6.3.1.4, followed by deleting all servers using the procedures defined in Section 5.2.1.4. After all DXConnections and servers are deleted, the DX server resets all DX Database attributes to their factory defaults, except for the ConfigurationVersion, which it updates as defined in Section 4.2.1.

5.2.3.1 ResetConfiguration Request

This request contains parameters that identify the version of the DX Database that is to be deleted. The parameters of the request are described in Table 55.

Table 55 – Reset Configuration Request Parameters

Parameter Name	Use	Description
ConfigurationVersion	O	This parameter contains the value that the DX server compares with its ConfigurationVersion attribute (see Section 4.2.1). If present, and the value of this parameter is not equal to the value of the ConfigurationVersion attribute, the DX server rejects the request. Otherwise, the DX server attempts to reset the configuration.

5.2.3.2 ResetConfiguration Response

This response contains the results of the requested service. The parameters of the request are described in Table 56.

The response is returned after the DX server completes its attempts to delete source servers and DXConnections and update the appropriate DX server status attributes (see Section 4.2). However, the DX server does not wait for the runtime behavior associated with changing runtime status attributes to complete.

Table 56 – ResetConfiguration Response Parameters

Parameter Name	Use	Description
ConfigurationVersion	C	This parameter contains the value of the DX server's ConfigurationVersion attribute (see Section 4.2.1) after completion of the request. If the request contained a ConfigurationVersion that did not match the ConfigurationVersion attribute of the DX Database, then the value of that attribute is returned. If the value of the ConfigurationVersion attribute was changed as a result of this service invocation, then its new value is returned.

Table 57 – ResetConfiguration Fault Codes

Code	Description
`E_XXX`	See Section 5.1.7
`E_PERSISTING`	See Section 5.1.7
`E_VERSION_MISMATCH`	See Section 5.1.7
`E_FAIL`	See Section 5.1.7
`E_OUTOFMEMORY`	See Section 5.1.7
`E_SERVER_STATE`	See Section 5.1.7

5.3 Persistence

During runtime the configuration attributes of DX Database can be altered by clients using the DX configuration services. In addition, certain status attributes can be updated using OPC DA Writes. The DX Server is responsible for maintaining a saved copy of the DX Database and these status attributes.

The saved DX Database includes the complete list of source servers and DXConnections as well as the ConfigurationVersion. No assumption is made about the form, format, or media used to save the configuration.

When the configuration/status attributes are changed in a way that would cause an update of the saved DX Database the DirtyFlag (see Section 4.2.5) is set. Specifically, any change that causes the ConfigurationVersion to be changed, including, but not limited to, updates to runtime attributes that were made via an OPC Data Access Client write operation will result in the DrityFlag being set. If an error occurs for a DXConnection/source server during execution of configuration or Write service, then the requested service is not performed for that DXConnection/source server and an appropriate error code is returned to the requestor. In this case the DirtyFlag will be set only if the requested service resulted in at least one DXConnection/source server being added, modified, or deleted.

The maximum time to elapse from when the DirtyFlag is set to when the DX Database is saved is one minute. When the DX Database is successfully saved the DirtyFlag is cleared by the OPC DX server. If an error occurs during the save procedure then the DirtyFlag is not cleared and the error is stored in the DX Server's Error attribute (see Section 4.2.6).

If a client tries to change the DX Database while the OPC DX server is currently saving the data the server can reject these changes by returning E_PERSISTING error code.

During DX Startup (see Section 6.1) the latest saved DX Database is used as the startup configuration. Connections to servers are established and DXConnections are activated during startup based on their saved definition.

If the DirtyFlag is set for the configuration the OPC DX Server persists the DX Database before shutting down.

6. OPC DX Runtime Model

6.1 DX Startup

When a DX server starts up, it restores its persisted database. This ensures continuity of operation from its last shutdown. Once the persisted database is restored, the DX server restores source servers and DXConnections to their runtime states as defined by their configuration and status attributes. Once all have been restored, the DX server sets its ServerState to “Running”, and opens communications for DX and DA clients.

6.2 Establishing and Maintaining Connections to Source Servers

6.2.1 Opening a Connection to a Source Server

The OPC DX server opens a connection to a source server to enable the transfer of data for one or more DXConnections. The OPC DX server inspects the ServerType (see Section 4.4.1.3) and ServerURL (see Section 4.4.1.4) configuration attributes to determine which procedures to use to connect to the source server. Once a connection to a source server is established, the DX server maintains its status attributes (see Section 4.4.1.6). The DX server opens a connection under the following conditions:

1. The SourceServerConnected status attribute (see Section 4.4.1.6.9) is TRUE.
2. There is at least one DXConnection whose source server (see Section 4.3.2.12) is this source server, and whose SourceItemConnected status attribute (see Section 4.3.2.19.15) is TRUE.

6.2.1.1 Security

This specification assumes that the credentials assigned by the system administrator to the DX server process are sufficient to authenticate the DX server when it connects to DCOM source servers. Therefore, the DX configuration services do not include security related parameters.

The same assumption is made for XML-DA source servers that use integrated Windows authentication for security. However, it is possible that additional security configuration information is necessary to connect to some XML-DA servers. In these cases, the additional security information can be provided, in an encrypted or clear text form, as part of the source server URL. The format should be based on published WSDL/HTTP specifications.

6.2.2 Modifying a Connection to a Server

OPC DX and OPC DA clients can modify a connection to a source server in one of the following ways:

1. DX clients can modify the ServerType (see Section 4.4.1.3) or ServerURL (see Section 4.4.1.4) configuration attributes using the ModifyServers service (see Section 5.2.1.3). Modifications to either of these cause the DX server to replace the existing connection with a new one, and reestablish all DXConnections associated with the source server to their runtime state. Because this requires that the existing connection be terminated, disruption of data transfer may occur on DXConnections associated with this source server. Other connections and DXConnections are not affected.
2. The SourceServerConnected attribute may be set to FALSE (see Section 4.4.1.6.9). This causes the connection to the source server to be terminated, and data transfer to cease on DXConnections associated with this source server.

When these modifications occur, the DX server updates the appropriate DX Server status attributes (see Section 4.4.1.6).

6.2.3 Recovering Connections to Source Servers

The DX Server is responsible for maintaining the connection to source servers. When the connection to a source server is lost, either during startup or runtime, the OPC DX Server tries to reestablish the connection. The minimum time between two attempts to reestablish a connection is defined by the source server PingTime status attribute (see Section 4.4.1.6.7).

If the DX server has not received data from the source server for any DXConnection for a PingTime period (see Section 4.4.1.6.7), it checks the status of connection to the source server using either the OPC DA GetStatus service or the OPC DA 3.0 KeepAlive method, depending on the type of the source server. For generality, the description below uses GetStatus.

Each time the DX Server receives a response to the GetStatus request or data for any DXConnection it assumes the connection to the source server is operational and restarts the Ping Timer. It also sets the DXConnectionState attribute (see Section 4.3.2.19.1) of each DXConnection associated with the source server to “Operational” if its SourceItemConnected attribute (see Section 4.3.2.19.15) is TRUE.

After the first two successive expirations of the PingTimer without receiving data or a GetStatus response from the source server, the DX server issues a GetStatus to the source server and restarts the PingTimer for period of twice the PingTime. This allows the source server to receive the request and respond to it.

After three successive expirations of the PingTimer without receiving data or a GetStatus response from the source server, the DX server assumes that the connection has failed and attempts to reestablish it. It also sets the ConnectStatus attribute (see Section 4.4.1.6.1) of the SourceServer to “disconnected” and the DXConnectionState attribute (see Section 4.3.2.19.1) of each DXConnection associated with the source server to “sourceServerNotConnected”. The Error attribute of the SourceServer will be set to E_SOURCE_SERVER_TIMEOUT or specific errors (e.g. RPC or HTTP-related) received from GetStatus.

This behavior is specified in the state table below (Table 58) and illustrated in Figure 6.

Table 58 – Connection Failure Recovery

Event	Conditional Expression	Action

ConnectionReestablished
|| Data Received
|| GetStatusResponseReceived
		RestartPingTimer (PingTime)
		GetStatusCount = 0
		FailedConnection = FALSE

PingTimerExpires	FailedConnection == TRUE	ReestablishConnection() RestartPingTimer (PingTime)
PingTimerExpires	FailedConnection == FALSE && GetStatusCount < 2	GetStatus() RestartPingTimer (PingTime * 2) GetStatusCount ++

PingTimerExpires	FailedConnection == FALSE
	&& GetStatusCount == 2
		FailedConnection = TRUE
		GetStatus()
		RestartPingTimer (PingTime)

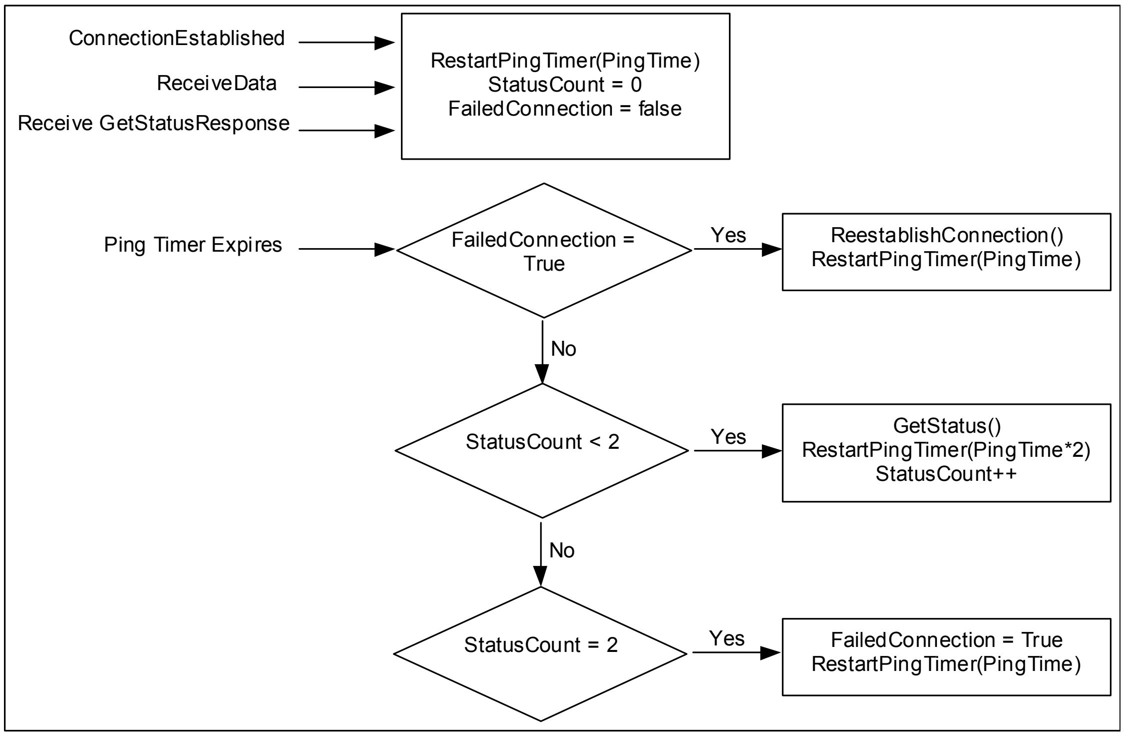

Figure 6 - Connection Failure Recovery Flow Diagram

6.2.4 Closing Connections to Source Servers

Connections to a source server are closed when the last DXConnection is deleted. Beginning the process of closing a connection to a source server causes its ConnectStatus attribute (see Section 4.4.1.6.1) to be set to "Disconnected".

6.3 Transferring Data on DXConnections

6.3.1 Maintaining DXConnections

6.3.1.1 Managing Subscriptions

6.3.1.1.1 Creating Subscriptions

Once a connection is established between the DX Server and a Source Server, the DX Server will acquire data for all DXConnections with this server where SourceConnected is set to TRUE. This section provides guidelines for acquiring source data for DXConnections. In general it is up to the vendor how this acquisition is performed.

How data is acquired from a source server depends on the type of the source server (eg,, DA2, DA3, XML-DA), but in general, the mechanism is referred to as a *subscription* in this specification. Subscribing to the data is supported by callbacks in COM DA, and by the Subscribe services in XML DA.

The general requirement for DX servers relates to the performance attributes, UpdateRate, and Deadband defined for DXConnections. The DX server has to request an update rate from the source server for the source data that is less than or equal to that defined for the DXConnection. If the Source server does not support a rate that small, then it responds with the smallest update rate that it supports. The same principles apply for the Deadband attribute. However, in the cases where Deadband is not supported by the source server, it is not used.

COM DA Groups

In COM-DA, the DX server creates one or more groups for each source server, specifying the UpdateRate, and Deadband for each. Once the groups are defined, the DX server adds data items to them to tell the source server which items to access. The data items that the DX server adds are the source items of the DXConnections.

This specification does not restrict the approach that the DX server uses for defining groups and adding items to them. For example, the DX server could create a set of groups when it connects to the source server, independent of the DXConnections defined for it. Then, as a separate operation, it could add the source items of DXConnections to an appropriate group based on its UpdateRate, and Deadband,. Alternatively, when the connection to the source server is established, the DX server could scan the DX Database for all DXConnections for that source server, organize them into groups, and add them to the source server. These two approaches, as well as others, are available to the DX server implementer for creating and managing groups.

XML DA Subscriptions

In XML-DA the DX Server uses the Subscribe and SubscriptionPolledRefresh services to acquire source data. The Subscribe service allows the client to define, in a single operation, a set of items for the source server to access. This means that the DX server will have to organize its DXConnections into subscription lists before it issues a Subscribe request, because once the subscription is defined, there are no services defined to add items to it.

In addition to UpdateRate, and Deadband. , XML-DA uses the “HoldTime” and “WaitTime” parameters to optimize the subscription to data. How the DX server provides these parameter is not defined by this specification, but the DX Server can use the UpdateRate as “HoldTime” and a multiple of the UpdateRate as “WaitTime”.

As with COM-DA, this specification does not restrict the approach the DX server uses for defining subscriptions. For example, the DX server could initially create a subscription for a DXConnection when it is added, and then periodically combine subscriptions together by adding a new subscription that contains the source items from each, and then delete the previous subscriptions. Other alternatives exist, and each DX server may implement its own approach.

6.3.1.1.2 Activating and Deactivating DXConnections Within a Subscription

Activating a DXConnection means that the source server begins sending data to the DX server on the subscription. DXConnections are activated when their SourceItemConnected attribute is TRUE, and when the source server is connected. Deactivating a DXConnection means that the source server stops sending the data.

Although the exact method of implementing activation and deactivation is not specified, activation or deactivation of one DXConnection must not interrupt the data flow on other DXConnections. In addition, when a subscription is activated or deactivated, the DX server is responsible for setting the DXConnectionState attribute (see Section 4.3.2.19.1) of the DXConnection to the appropriate value.

As with subscriptions, there are different approaches available to DX servers for activating/deactivating DXConnections within a subscription. In COM-DA, activating a DXConnection can be accomplished by activating the source item in a group or by adding it to an existing group. Similarly, deactivating a DXConnection can be accomplished by deactivating the source item in the group or by removing it from the group.

In XML-DA, subscriptions can be added and deleted, but not modified. Therefore, the DX server can assign a single DXConnection to each subscription, and manage them individually, or it can assign multiple DXConnections to a subscription. When a subscription contains multiple DXConnections, activation and deactivation is accomplished by making a copy of an existing subscription definition, and then adding or deleting the source item to/from it to create a new subscription. The new subscription is added, and its previous version is deleted. Adding the newly updated subscription definition first, and then deleting the previous one assures that the data flow is not interrupted

6.3.1.1.3 Handling Subscription Errors

A fatal error could occur when creating or modifying a subscription for one or more DXConnections. For example, a COM-DA call to AddGroup, AddItems or SetState may fail even if the source server is otherwise functioning normally. In this situation, the DX server should verify that the source server is still functioning normally by calling GetStatus and checking the source server state. If the source server appears to be fine then the DX server must set the state of each affected DXConnection to “subscriptionFailed” and set the SourceErrorID to indicate why (eg E_SUBSCRIPTION_FAULT or E_UNKNOWN_SOURCE_ITEM). The DX server should not interrupt other operational DXConnections to the same source server because of a problem with a specific subscription. If the source server is not fine the DX server should place the source server in the "failed" state and initiate the recovery process described in 6.2.3. Some DX servers may wish to employ more complex algorithms to recover from this situation. Note that a DX server should never report a type conversion or data range error while creating the subscription because the DX server should always attempt to use the source item’s canonical data type if the source server does not support conversion to the target item’s data type.

6.3.1.2 Adding DXConnections

When a new DXConnection is added to the DX Database, the DX server prepares the DXConnection for operation by creating and initializing the runtime Status attributes and the source item queue of the DXConnection. The source item queue may be created at any time prior to data transfer.

If the source server is connected, it also adds the DXConnection to an existing subscription, or creates a new subscription for it. Then, depending on the value of other DXConnection attributes, it begins data transfer on the DXConnection. Table 59 and Table 60 summarize the runtime behavior associated with these attributes. Attributes not shown in these tables have no effect on the runtime behavior.

During this process, the DX server maintains the current state of all status attributes related to the DXConnection (see Section 4.3.2.19).

Adding a DXConnection has no effect on other existing DXConnections. They continue operation.

Table 59 – Source Server Attribute Runtime Behaviors

Attribute	DX Runtime Behavior
SourceServerConnected	The DX server examines this source server status attribute to determine whether or not it can add the DXConnection to a subscription.
ServerState	The DX server examines this source server status attribute to determine whether or not the source server is connected. If not, and if SourceServerConnected = TRUE, the DX server is either attempting reconnection as specified in Section 6.2.3, or the connection to the source server has failed. In either case, the DX server will not be able to add this DXConnection to a subscription until the source server is reconnected.

Table 60 – DXConnection Attribute Runtime Behaviors

Attribute	DX Runtime Behavior
DefaultSourceItemConnected	The DX server copies the value of this DXConnection configuration attribute to the SourceItemConnected Status attribute (see Section 4.3.2.19.15).
DefaultTargetItemConnected	The DX server copies the value of this DXConnection configuration attribute to the TargetItemConnected Status attribute (see Section 4.3.2.19.16).
DefaultOverridden	The DX server copies the value of this DXConnection configuration attribute to the Overridden Status attribute (see Section 4.3.2.19.17).
DefaultOverrideValue	The DX server copies the value of this attribute to the OverrideValue Status attribute (see Section 4.3.2.19.18).
TargetItemName TargetItemPath	The DX server uses these to locate and access the TargetItem. If the target item cannot be located, the DX server sets the DXConnectionState attribute (see Section 4.3.2.19.1) to indicate this condition.
SourceServerName	The DX server uses this to identify the source server and determine its state.
SourceItemName, SourceItemPath	The DX server uses these to identify the source item for the subscription. If the source item cannot be acquired from the source server, the DX server sets the DXConnectionState attribute (see Section 4.3.2.19.1) to indicate this condition.
UpdateRate, Deadband	The DX Server uses these to locate or create an appropriate subscription for the DXConnection and add the source item of the DXConnection to it. The DX server may defer this step until the SourceServerConnected attribute is TRUE. After adding the source item of the DXConnection to the subscription, the update rate of the subscription is copied to the ActualUpdateRate Status attribute (see Section 4.3.2.19.11) of the DXConnection. See Section 6.3.1.1 for a description of subscriptions.
SourceItemQueueSize	The DX server uses this attribute to create and initialize a queue to receive the source data.
SourceServerConnected	If the value of this attribute is TRUE, the DX server activates the source item for the subscription. If FALSE, the source item is deactivated. The DX server may alternately defer adding the source item of the DXConnection to a subscription until the value of this attribute is TRUE.

6.3.1.3 Modifying DXConnections

When a DXConnection’s attributes are modified by a DX client, the DX server may be required to remove the existing DXConnection from a subscription and add the modified subscription back to the same or a different subscription. The new subscription may be to the same source server or to a different source server.

When such a modification is performed on a DXConnection, the DX server clears the source item queue, and the target value contains the last value from the previous subscription until the first value is received from the new subscription.

Table 61 summarize the runtime behavior associated with modifying DXConnection attributes. Modification of attributes not shown in this table has no effect on the runtime behavior.

During this process, the DX server maintains the current state of all status attributes related to the DXConnection (see Section 4.3.2.19).

Modifying a DXConnection has no effect on other existing DXConnections. They continue operation.

Table 61 – Runtime Modifications to DXConnection Attributes

DXConnection Attribute	DX Runtime Behavior
TargetItemName TargetItemPath	These attributes cannot be modified.
SourceServerName SourceItemName, SourceItemPath	This type of modification replaces the existing source item with a new source item. The new source item may be in the same server or in a different server. In both cases, the DXConnection is removed from its existing subscription, and added to a different subscription. However, the timing of these operations are dependent on) the SourceItemConnected attribute of the DXConnection (see Section 4.3.2.19.15). If the source server for the new subscription is connected and SourceItemConnected = TRUE, then before the DX server removes the DXConnection from the existing subscription, it adds the DXConnection to a new or existing subscription, and waits for data to be received. When data is received on the new subscription or the new subscription fails, the DX server removes the DXConnection from the previous subscription. Receiving the new data before removing the previous DXConnection prevents interruption of the data flow on the DXConnection. If the source server for the new subscription is connected, but SourceItemConnected = FALSE, the DX server removes the DXConnection from the existing subscription and adds it, deactivated, to the new subscription. In this case, the timing of the two operations does not matter because there is no data flow. Finally, if the source server for the new subscription is not connected, the DX server removes the DXConnection from the existing subscription and waits until the source server is reconnected to add the DXConnection to a new subscription. In this case the flow of data stops on the DXConnection.
UpdateRate, Deadband,	To change one of the parameters, the DXConnection has to be moved to another subscription (see Section 6.3.1.1). This may be an existing subscription, or the DX server may have to add a subscription. If the previous subscription to the source item is to be released after the move, the DX Runtime has to preserve data transfer on the DXConnection by waiting for the initial data update on the new subscription before the release. Any changes to UpdateRate, and Deadband will not modify the state of the update queue.
SourceItemQueueSize	If the SourceItemQueueSize is reduced to a size less than the current number of entries in the queue, the DX server considers this as a queue overrun. It moves the newest value to the head of the queue, flushes all other entries, and increments the QueueFlushCount status attribute. In this case, it clears the QueueHighWaterMark. If the SourceItemQueueSize is increased, the new entries are appended to the end of the queue.

6.3.1.4 Deleting DXConnections

To delete a DXConnection, the DX server:

1. stops the subscription between DX server and DA Source Server for the DXConnection,
2. releases all resources associated with the DXConnection,
3. generates an OnDataChange() event and returns E_UNKNOWNITEMID to all DA clients currently subscribed to the DXConnection. All further DA accesses to the DXConnection will be rejected with E_UNKNOWNITEMID.

6.3.2 Runtime Controls

Runtime controls are provided as OPC DA Status attributes that can be written by DA clients to affect the behavior of the source servers and DXConnections. They may also be updated by returning them to their default values using the OPC DX services, CopyDefaultServerAttributes (see Section 5.2.1.5) and CopyDefaultDXConnectionAttributes (see Section 5.2.2.6).

This section describes the runtime behavior associated with updating the values of these attributes. When the value of the source server attribute changes, the DX server performs the operations specified in Table 62. The DX server also persists the new value as described in Section 5.3 so that any restart of the DX server will maintain the new value. No change to the ConfigurationVersion (see Section 4.2.1) results from the persisting of this attribute.

Table 62 – Source Server Runtime Controls Behavior

Source Server Attribute	Value	DX Runtime Behavior
SourceServerConnected	TRUE	If one or more DXConnections are associated with this source server and one or more of these DXConnections is enabled (i.e. *SourceItemConnected* = TRUE, see Section 4.3.2.19.15) the DX Runtime also: • Sets the Source Server ConnectStatus attribute (see Section 4.4.1.6.1) to the "connecting" state. • Attempts to establish a connection to the source server as described in Section 6.2.1. If the connection establishment succeeds, the DX Server: • Updates the Source Server Status attributes (see Section 4.4.1.6) as follows: o o Sets the Error attribute to indicate a successful connection. o Sets the LastConnectTimestamp to the current time. o Sets the ConnectStatus to the "connected" state. o . If the connection establishment does not succeed, the DX Server: • Starts the source server Connection Recovery procedure described in Section 6.2.3. • Updates the Source Server Status attributes (see Section 4.4.1.6) as follows: o Increment the ConnectFailCount attribute by one. o Set Error to indicate the cause of the failure. o Set LastConnectionFailTimestamp to the current time. o Set ConnectionStatus to the 'failed' state.
	FALSE	If the connection to the source is established, starts the connection closing procedure described in Section 6.2.4. Updates the Source Server Status attributes (see Section 4.4.1.6) as follows: • Set the ConnectStatus to the "disconnected" state. • • Clear the Error attribute. Sets the DXConnectionState attribute (see Section 4.3.2.19.1) of each DXConnection associated with the source server to "SourceServerNotConnected".

When the value of one of the DXConnection runtime status attributes changes, the DX server performs the operations specified in Table 63. The DX server also persists the new value as described in Section 5.3 so that any restart of the DX server will maintain the new value. No change to the ConfigurationVersion (see Section 4.2.1) results from the persisting of this attribute.

These runtime status attributes may be modified in three ways: by writing to individual DA items; by writing an XML document to a single DA item; or by using the CopyDefaultDXConnectionAttributes configuration service. If a DA client writes to these attributes via individual DA items then the DX must apply the changes in the order specified in the DA write request. If a DA client writes to these attributes via and XML document or a DX client uses the CopyDefaultDXConnectionAttributes service then the DX server must apply the changes in the following order: OverrideValue, Overriden, TargetItemConnected, SourceItemConnected.

Table 63 – DXConnection Runtime Controls Behavior

DXConnection Attribute	New Value	DX Runtime Behavior
SourceItemConnected	TRUE	If the Source Server attribute, SourceServerConnected (see Section 4.4.1.6.9), is FALSE, the connection to the source server cannot be established, and the DX server takes no further action on this DXConnection. If SourceServerConnected = TRUE, and the source server is not connected, then the DX server follows the behavior described in Section 6.2.1 or Section 6.2.3 to establish the connection to the source server. When a connection to the Source Server exists, the DX server: • Sets the DXConnectionState attribute (see Section 4.3.2.19.1) to “Initializing”. • Sets the SourceQuality attribute to a ‘BAD – Not Connected’ state. • Activates the DXConnection as described in Section 6.3.1.1.2. Once the DXConnection has been activated the DX server, and after the first value is received: • Sets the DXConnectionState attribute (see Section 4.3.2.19.1) to “Operational”. • Begins queueing the received data as described in Sections 6.3.3. •
	FALSE	If the Source Server attribute, ConnectStatus (see Section 4.4.1.6.1), indicates that the connection to the source server is not established, the DX server takes no further action on this DXConnection. If the connection to the source server is established, the DX server: • Deactivates the DXConnection as described in Section 6.3.1.1.2. • Sets the DXConnectionState attribute (see Section 4.3.2.19.1) to “Deactivated”. • Sets the SourceQuality to a ‘BAD – Not Connected’ state. • Follows behavior described in Section 6.3.5 dealing with a change to the SourceQuality. If this was the last activated DXConnection associated with the Source Server, the DX server closes the connection to the Source server as described in Section 6.2.4.
TargetItemConnected	TRUE	The DX server begins dequeueing data from the source item queue as described in Section 6.3.3. and updates the target item as described in Section 6.3.5. The DX server updates the target item as described in Section 6.3.5.
	FALSE	The DX server flushes queued updates from the source item queue, except for the newest update, which is moved to the start of the queue. New updates overwrite the entry at the start of queue, leaving the remainder of the queue empty. This behavior ensures that when TargetItemConnected changes to TRUE, that the latest update will be written to the target (see Section (6.3.3). The DX server no longer updates the target item (see Section 6.3.5).
Overridden	TRUE	The DX server updates the target item as described in Section 6.3.5. The effect of setting this attribute to TRUE when TargetItemConnected = TRUE (see Section 4.3.2.19.16), is that the DX server immediately writes the Override value to the target value one time, instead of the source item value, as follows: • TargetQuality to a ‘Good – Local Override’ state. • TargetValue to the value of the OverrideValue attribute). • TargetTimestamp to the current DX server time. The DX server also flushes queued updates from the source item queue, except for the newest update, which is moved to the start of the queue. New updates overwrite the entry at the start of queue, leaving the remainder of the queue empty. This behavior ensures that when Overridden changes to FALSE, the latest update will be available to be written to the target (see Section (6.3.3).
	FALSE	When TargetItemConnected = TRUE, the DX server begins dequeueing data from the source item queue as described in Section 6.3.3 and updates the target item as

		described in Section 6.3.5. The effect of setting this attribute to FALSE when TargetItemConnected = TRUE (see Section 4.3.2.19.16), is that the DX server immediately dequeues the first value from the source item queue writes it to the target value.
OverrideValue	Valid Type	If the DXConnection is currently overridden (i.e. Overridden = TRUE) and the target is connected (i.e. TargetItemConnected = TRUE), the DX server writes the new override value to the target item, as follows: • Set the TargetQuality to a 'Good – Local Override' state. • Set the TargetValue to the new OverrideValue. • Set the TargetTimestamp to the current DX server time. Otherwise, the DX server takes no additional action.
	Empty	If the DXConnection is currently not overridden (i.e. Overridden = FALSE), the DX server takes no additional action. If the DXConnection is currently overridden any attempts to clear the value are rejected and there is no effect on the runtime behavior.

6.3.3 Queueing Source Data

Each DXConnection has a source queue that queues source data that is waiting to be written to the target item. The maximum size of the source item queue is defined by the DX server status attribute, MaxQueueSize (see Section 4.2.9). The size of the source item queue for a DXConnection is a configuration attribute, SourceItemQueueSize (see Section 4.3.2.15)

In cases where the DX server receives source updates faster than it can write to the target, the queue holds the data in first-in-first-out (FIFO) order. It updates the QueueHighWaterMark status attribute (see Section 4.3.2.19.13) to reflect the highest number of entries that have been in the queue at the same time.

Upon completion of a write to the target item (success or failure), the DX server dequeues the oldest value from the queue and attempts to write it to the target. The DX server continues to dequeue and write target values until the queue is empty.

If source data is received, and the queue is full (SourceItemQueueSize is reached), the DX server considers this as a queue overrun flushes the queue by discarding all entries, and adds the received data to the start of the queue. It also increments the QueueFlushCount status attribute (see Section 4.3.2.19.14) and clears the QueueHighWaterMark attribute. Flushing the queue in this manner prevents the queue from introducing additional delay to the data transfer.

A queue with "SourceItemQueueSize = 1" represents a special case in which the DX server treats the queue as a cache. In this case, the DX server always writes the latest received value to the target.

6.3.4 Data Conversion

When a DXConnection is configured the data type of the source item is not known by the DX Server. When the DXConnection is established to the source server the DX Server uses the canonical data type of the Target Item as the requested data type for the Source Item. If the Source Server is not capable of performing the data conversion this has to be done by the DX Server.

The DX server performs data conversions as specified for the DA server on which it is based. See the DA specifications for more detail on data conversions.

The SourceErrorID will be set toE_SOURCE_ITEM_BADTYPE, if the DX server is unable to perform the conversion.

Note: In some cases OPC DA Server support special conversion routines (e.g. enums). Therefore the Source Server should do the conversion if possible.

6.3.5 Updating Target Values

6.3.5.1 Target Update Truth Table

The DX server is responsible for managing the data flow on a DXConnection from source to target. Generally, it subscribes to data from the source and copies it to the target when it is received. However, there are conditions that alter this general behavior. This section identifies these conditions and specifies the rules that DX servers apply to update target items.

Figure 7 illustrates the data flow from the source item to target item. The rules that govern this data flow are specified using a truth table (see Table 64 below) that uses a set of boolean conditions. The truth table indicates how the server updates the target based on the different combinations of the conditions.

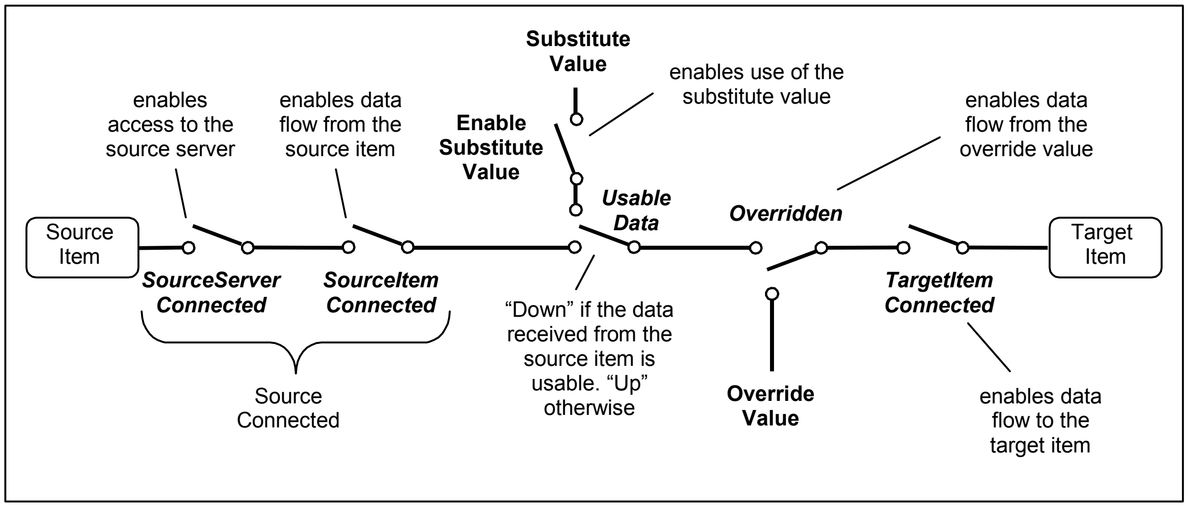
Substitute Value
enables use of the substitute value
enables access to the source server
enables data flow from the source item
Enable Substitute Value
enables data flow from the override value
Source Item
Usable Data
Overridden
Target Item
SourceServer Connected
SourceItem Connected
"Down" if the data received from the source item is usable. "Up" otherwise
TargetItem Connected
Source Connected
Override Value
enables data flow to the target item

Figure 7 – Target Update Data Flow

The conditions are defined in Table 64 below and the truth table follows. This behavior is also described in a flow chart, which is presented after the truth table.

After each write attempt to the target item the following status parameters have to be updated together even if the write attempt failed: WriteValue, WriteTimestamp, WriteQuality and WriteErrorID.

Some DX Servers might not be capable of writing quality codes to the target item. Even if the DX server cannot write the quality codes directly to the target, these codes should be written to the status parameters according to Table 65 below. This means that the quality information received from the status parameters and the target item itself might be different. The DX server will update the TargetErrorID runtime attribute of a DXConnection if it encounters that the target is no longer accessible by the DX server. If the target eventually becomes re-accessible, TargetErrorID will be cleared.

Table 64 – Truth Table Columns

Condition	Description
TargetItemConnected	The TargetItemConnected DXConnection attribute is TRUE and DX server has established access to the target. When FALSE, the DX server does not access the target item, and therefore, does not update its value.
Overridden	The Override Value attribute contains a valid value.
SourceConnected	SourceConnected is TRUE if the SourceItemConnected DXConnection status attribute and the SourceServerConnected SourceServer status attribute are TRUE and the DX server is connected to the source server.
Data Received	The DXConnection has received data from the source. The data includes the value, quality, and its timestamp.
Connection Fail	The connection to the source server has failed.
Usable Data	Data has been received and can be used to update the target. Errors or conditions encountered during processing of the received data may identify the data as unusable. Examples include out-of-range values and incompatible data types.
Substitute Value	TRUE when the EnableSubstituteValue DXConnection configuration attribute is TRUE
Target Value	The value written to the target by the DX server. The value "Same" in this column indicates that the value is not changed by the DX server, but the associated quality is. This may require the DX server to write the same value back to the TargetItem, but with the new quality.
Target Quality	The quality associated with the value written to the target by the DX Server

Table 65 – Target Value and Quality Truth Table

Target Connected[1]	Overridden	Source Connected	Data Received	Connection Failed[2]	Usable Data[3]	Enable Substitute Value	Target Value	Target Quality
No	Don't Care	Don't Care	Don't Care	Don't Care	Don't Care	Don't Care	No Update[4]	No Update
Yes	Yes	Don't Care	Don't Care	Don't Care	Don't Care	Don't Care	Override Value	Good (Local Override)
Yes	No	No	Don't Care	Don't Care	Don't Care	Don't Care	No Update	BAD (Not Connected)
Yes	No	Yes	Yes	No	No	Yes	Substitute Value	Uncertain (uncertainSubNormal)
Yes	No	Yes	Yes	No	No	No	No Update	BAD (Last Known Value)
Yes	No	Yes	Yes	No	Yes	Don't Care	Received Value	Received Quality
Yes	No	Yes	No	Yes	Don't Care	Yes	Substitute Value	Uncertain (uncertainSubNormal)
Yes	No	Yes	No	Yes	Don't Care	No	No Update	BAD (Last Known Value)
Yes	No	Yes	No	No	Don't Care	Don't Care	No Update	No Update

Note 1: When Target Connected = Yes, the target item is not writable through OPC DA.
When Target Connected = No, OPC Writes to the target item are permitted if the item is writable.

Note 2: The source server must clear the queue and set the source error id if it knows that no updates are possible do to a failed connection.
If the source can connect it must clear the SourceErrorID and re-enable queuing.

Note 3: Un-usuable values should be placed in the queue and written at the appropriate time following the rules above.
This behavior is necessary since unusuable values may be followed by usable values if the connection has not failed.
The queue should include the source error id for the sample as well as the value, quality and timestamp.

Note 4: When the target is disconnected the DX server writes nothing to the target item and does not update the WriteValue, WriteTimestamp or WriteQuality. The WriteErrorID should be OPCDX_E_TARGET_ITEM_DISCONNECTED.
When the target is reconnected the WriteErrorID should be set to OPCDX_E_TARGET_NO_WRITES_ATTEMPTED.

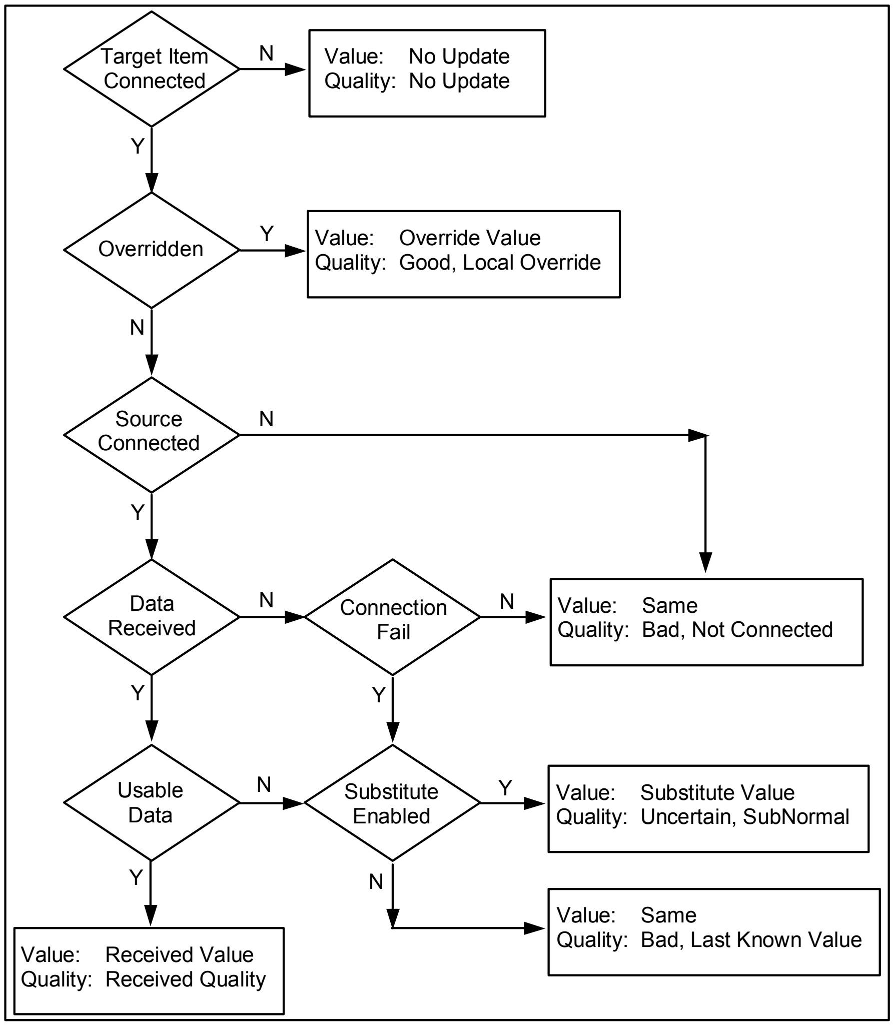

Figure 8 – Updating the Target

6.3.5.2 Relationship to DXConnection States

Any DXConnection implementation must specifically check for the E_INVALID_TARGET_ITEM or E_UNKNOWN_TARGET_ITEM error codes (or their DA equivalents) returned during any write to the target. If detected then the DXConnectionState is forced to "targetItemNotFound". It remains in this state until a subsequent write to the target indicates success. The truth table logic for target updates still applies - just the Write Error ID will indicate why the target item was not found. This correctly handles transient issues where the target item takes time to be available. The DXConnection should **not** behave as if Target Item Connected is false in this situation.

If Source Server Connected is false then the Source Error ID should indicate a failure (i.e. the source server is disconnected due to a ping timeout). In this case the DXConnectionState is forced to "sourceServerNotConnected" and the Source Error ID indicates the reason. The DXConnection remains in this state until either the "targetItemNotFound" condition is met or the source server able to re-establish the connection. A DXConnection in this state may be activated if the Source Error ID indicates E_SOURCE_SERVER_NOT_CONNECTED by setting Source Server Connected to true. The DXConnection must ignore data updates received after being forced into this state since thread timing issues could cause an update to the Source Error ID that hides the reason for being in the "sourceServerNotConnected" state. If the DXConnection has Source Item Connected set to false then the DXConnectionState is "deactivated". Note that any fatal source server error (i.e. the DXConnectionState is "sourceServerNotConnected") or target item error (i.e. the DXConnectionState is "targetItemNotFound") masks this state. The DXConnection returns to this state if either higher state is cleared (i.e. the source server is activated).

Any active DXConnection (defined by Source Item Connected and Source Server Connected set to true) will be in the DXConnectionState “subscriptionFailed” if either a problem occurred creating the subscription while actvating the DXConnection or if the data updates indicate a fatal error condition such as E_UNKNOWN_SOURCE_ITEM. In some cases, this error condition may resolve itself in a subsequent data update. As long as the server continues to receive data updates the values received should be treated as un-usable data and handled normally. However, if the source server knows that the problem will not resolve itself then it must clear the queue and set the Source Error ID to indicate the reason why (e.g. E_SUBSCRIPTION_FAULT). In this case the connection has failed. The DXConnection leaves this state if any higher-level state occurs (ie the DXConnection is deactivated) or if the Source Error ID indicates either success or a only a type conversion or data range error. The DXConnection moves into either the “initializing” or “operational” states depending on whether the source server was receiving data updates while in this state. Note that a DX server should never report a type conversion or data range error while creating the subscription because the DX server should always attempt to use the source item’s canonical data type if the source server does not support conversion to the target item’s data type.

Any active DXConnection that has not received any updates from the source server is in the "initializing" state. This check is performed each time the DXConnectionState changes and the queue is currently empty (e.g. data was received but could not be written since due to the "targetItemNotFound" state). Any write to the queue causes the DXConnection to move to the "operational" state unless the Source Error ID indicates that the source item was not found. All higher-level states mask this state.

Any active connection that as received a data update (even if the data was not usable) should placed in the "operational" state. Any Source Error ID other than a type conversion or range error means the connection has failed and the DXConnection moves into the “subscriptionFailed” state until the error condition resolves itself.

Based on the above, an algorithm for determining the correct DXConnectionState after an event that could have changed the state is:

- If the write error id is “failed” and indicates the target item is not found then the new state "targetItemNotFound";
- If Source Server Connected is false and the Source Error ID is “failed” then the the new state is "sourceServerNotConnected";
- If either SourceItemConnected or Source Server Connected is false the the new state is "deactivated".
- If the Source Error ID indicates any error other than a data type conversion or data range errors then the new state is “subscriptionFailed”.
- If the current state is not "operational" then new state is "initializing".
- If the new state is "operational";

Events that could change the state include:

- A failure while writing to the target.
- A fatal error reported by the source server.
- A change to either source server connected or source item connected.

- A failure reported by the source server while creating or modifying a subscription.
- A failure reported by the source server during an update from the source server.
- Any update from the source server.

6.3.5.3 TargetErrorID Runtime Behavior

The DX server will update the TargetErrorID runtime attribute of a DXConnection if it encounters that the target is no longer accessible by the DX server. If the target eventually becomes re-accessible, TargetErrorID will be cleared.

Listed below are some examples:

- E_TARGET_ITEM_DISCONNECTED [Target is no longer online and can not be accessed]
- E_TARGET_NO_ACCESS [Target may be online, but can no longer be accessed. (e.g., security, configuration has been modified in the target device, rejecting request due to maintenance)]
- E_TARGET_FAULT [Target is online, but can not service any request due to being in a fault state]
- E_XXX [Vendor specific error conditions]

6.4 Status and Monitoring

DA clients are capable of monitoring the status of the DX server, source servers, and DXConnections via the status attributes of each. The general descriptions of these attributes are provided in Sections 4.2, 4.3.2.19, and 4.4.1.6, and the specific runtime behavior associated with them is described earlier in this section.

When accessing these attributes via OPC DA interfaces, the quality and timestamp parameters reflect the quality and timestamp as determined by the DX server. Therefore, in the absence of DX server errors, all qualities should be good, and all timestamps should reflect the policy for timestamps defined by the OPC DA specifications.

6.5 DX Shutdown

In typical configurations with several DX Servers connected, shutting down DX Servers has to follow certain rules. This is due to the recovery rules that requires DX Servers to reestablish connections within a few hundred milliseconds up to a few seconds. Without special shutdown rules this might cause immediate restart of the server trying to shutdown.

This is a problem in particular for COM, where source and target are closely coupled. The following rules are therefore for COM-based communication only.

Rules for the DX server being shut down:

1. The DX server shall change its ServerState attribute to “OFFLINE”.
2. The DX Server shall terminate its own connections to source servers (remove items, groups, release the server, disconnect targets).
3. The DX server shall notify all connected clients of shutdown. This is done by calling the OnShutdown connection point.
4. The DX server calls CoDisconnectObject() on the OPC Server & OPC Group objects.
5. If the Dirty flag is set for the configuration, the OPC DX Server persists the configuration before shutting down.
6. The DX server shall fail new client requests during a shutdown procedure. I.e., for COM DX servers, clients may be able to create a server instance (due to how COM works) but will not be able to add groups.

Rules for DX servers that are clients to the DX server shutting down:

1. When receiving the shutdown event from a source server, a DX server will terminate all connections and DXConnections to this source and will mark this server as being “offline”.
2. The DX Server will try to reestablish connections to servers that are “OFFLINE” at a rate of one minute. Every minute, after successfully connecting to the server, first the status is checked (GetStatus). If the status is still “OFFLINE”, the server is released again, terminating this attempt to reestablish connections.

Appendix A. Web Services Implementation

This appendix describes, how the configuration interface will be mapped to WebServices. OPC XML-DA /XYX/ specifies the DataAccess interface for WebServices.

The WebService Interface for the Configuration Services is fully specified by the WSDL (Web Service Description Language) later in this section. The semantic of all services is specified in Chapter YZX, above.

A.1. Implementation Guidelines

A.1.1. Error Handling

When abnormal conditions arise, web services must be able to communicate exception information to applications on a per-operation and a per-item (server, DXConnection) 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) or other invalid arguments. No other results (e.g., DXConnection data) 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. See 5.1.2 for a list of the defined fault codes.

However, even when an operation as a whole succeeds, individual elements (servers, DXConnections) may encounter critical or non-critical exceptions. The result of each service element contains a valid result code. OPC DX defines a series of standard result codes (see Section 5.1.7) that have specific applications in DX configuration operations. These codes are always qualified with the namespace "http://opcfoundation.org/webservices/OPCDX/".

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.

Success codes and error codes for each service are listed as part of the response messages in the sections that describe those services. As a convention, success codes begin with “S_” and error codes begin with “E_”.

The following fault codes should be used on a complete failure of a call. Additional faults may occur due to protocol or parsing errors.

A.1.2. Locale Ids and Verbose Errors

WebServices assume no persistent connection between the client and the server. Additional roundtrips should be avoided if possible. Because of this, the WebService version for DX Configuration requires locale information to be requested in each call ("LocaleID") and verbose error information to be returned if necessary. In COM this functionality maps to IOPCCommon (SetLocaleID, GetErrorString).

The locale string is formatted according to RFC 3066 "Tags for the Identification of Languages", using ISO639 and ISO3166 codes for language and country, respectively. Example locale strings include "en-us" for US English (Windows LCID:0x0409), "fr-ca" for Canadian French (Windows LCID:0x0C0C) and "de" for German standard (Windows LCID:0x0407). In the event that the server does not support the requested locale it is expected that the server will return its default locale identifier (RevisedLocaleID, see responses).

Only strings for errors will be localized.

A.2. Mapping of Parameters and Services

A.2.1. Parameter Definitions

A.2.1.1 DXConnection

```
<s:complexType name="DXConnection">
  <s:complexContent mixed="false">
    <s:extension base="s0:ItemIdentifier">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded"
                   name="BrowsePath" type="s:string" />
        <s:element name="Description" type="s:string" />
        <s:element name="DefaultOverrideValue" nillable="true" />
        <s:element name="SubstituteValue" nillable="true" />
        <s:element name="VendorData" type="s:string" />
      </s:sequence>
      <s:attribute name="Name" type="s:string" />
      <s:attribute name="Keyword" type="s:string" />
      <s:attribute name="DefaultSourceItemConnected" type="s:boolean" />
      <s:attribute name="DefaultTargetItemConnected" type="s:boolean" />
      <s:attribute name="DefaultOverridden" type="s:boolean" />
      <s:attribute name="EnableSubstituteValue" type="s:boolean" />
      <s:attribute name="TargetItemPath" type="s:string" />
      <s:attribute name="TargetItemName" type="s:string" />
      <s:attribute name="SourceServerName" type="s:string" />
      <s:attribute name="SourceItemPath" type="s:string" />
      <s:attribute name="SourceItemName" type="s:string" />
      <s:attribute name="QueueSize" type="s:unsignedInt" />
      <s:attribute name="UpdateRate" type="s:unsignedInt" />
      <s:attribute name="Deadband" type="s:double" />
    </s:extension>
  </s:complexContent>
</s:complexType>
```

A.2.1.2 DXGeneralResponse

```
<s:complexType name="DXGeneralResponse">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
               name="ItemResult" type="s0:IdentifiedResult" />
  </s:sequence>
  <s:attribute name="ConfigurationVersion" type="s:string" />
</s:complexType>
```

A.2.1.3 IdentifiedResult

```
<s:complexType name="IdentifiedResult">
  <s:sequence>
    <s:element name="Identifier" type="s0:ItemIdentifier" />
  </s:sequence>
  <s:attribute name="ResultCode" type="s:QName" />
</s:complexType>
```

A.2.1.4 ItemIdentifier

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

A.2.1.5 SourceServer

```
<s:complexType name="SourceServer">
  <s:complexContent mixed="false">
    <s:extension base="s0:ItemIdentifier">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string" />
      </s:sequence>
      <s:attribute name="Name" type="s:string" />
      <s:attribute name="ServerURL" type="s:string" />
      <s:attribute name="ServerType" type="s:string" />
      <s:attribute name="DefaultSourceServerConnected" type="s:boolean" />
    </s:extension>
  </s:complexContent>
</s:complexType>
```

A.2.2. OPC DX Services

A.2.2.1 Server Services

A.2.2.1.1 GetServers

```
<s:element name="GetServers">
  <s:complexType />
</s:element>

<s:complexType name="ArrayOfSourceServer">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
               name="SourceServer" nillable="true" type="s0:SourceServer" />
  </s:sequence>
</s:complexType>

<s:element name="GetServersResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="GetServersResult" type="s0:ArrayOfSourceServer" />
    </s:sequence>
  </s:complexType>
</s:element>
```

A.2.2.1.2 AddServers

```
<s:element name="AddServers">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="SourceServer" type="s0:SourceServer" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
  </s:complexType>
</s:element>

<s:element name="AddServersResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="AddServersResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.1.3 ModifyServers

```
<s:element name="ModifyServers">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="ServerDefinition" type="s0:SourceServer" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
  </s:complexType>
</s:element>

<s:element name="ModifyServersResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="ModifyServersResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.1.4 DeleteServers

```
<s:element name="DeleteServers">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="ServerID" type="s0:ItemIdentifier" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
  </s:complexType>
</s:element>

<s:element name="DeleteServersResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="DeleteServersResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.1.5 CopyDefaultServerAttributes

```
<s:element name="CopyDefaultServerAttributes">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="ServerID" type="s0:ItemIdentifier" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="ConfigToStatus" type="s:boolean" />
  </s:complexType>
</s:element>

<s:element name="CopyDefaultServerAttributesResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="CopyDefaultServerAttributesResult"
                                        type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.2 DXConnection Services

A.2.2.2.1 QueryDXConnections

```
<s:element name="QueryDXConnections">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnectionMasks" type="s0:DXConnection" />
    </s:sequence>
    <s:attribute name="BrowsePath" type="s:string" />
    <s:attribute name="Recursive" type="s:boolean" />
  </s:complexType>
</s:element>

<s:complexType name="ArrayOfDXConnection">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnection" nillable="true" type="s0:DXConnection" />
  </s:sequence>
</s:complexType>

<s:element name="QueryDXConnectionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="QueryDXConnectionsResult" type="s0:ArrayOfDXConnection" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="MaskErrors" type="s:QName" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
  </s:complexType>
</s:element>
```

A.2.2.2.2 AddDXConnections

```
<s:element name="AddDXConnections">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnection" type="s0:DXConnection" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
  </s:complexType>
</s:element>

<s:element name="AddDXConnectionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="AddDXConnectionsResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.2.3 UpdateDXConnections

```
<s:element name="UpdateDXConnections">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnectionMasks" type="s0:DXConnection" />
      <s:element name="DXConnectionDefinition" type="s0:DXConnection" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="BrowsePath" type="s:string" />
    <s:attribute name="Recursive" type="s:boolean" />
  </s:complexType>
</s:element>

<s:element name="UpdateDXConnectionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="UpdateDXConnectionsResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="MaskErrors" type="s:QName" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.2.4 ModifyDXConnections

```
<s:element name="ModifyDXConnections">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnectionDefinition" type="s0:DXConnection" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
  </s:complexType>
</s:element>

<s:element name="ModifyDXConnectionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="ModifyDXConnectionsResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.2.5 DeleteDXConnections

```
<s:element name="DeleteDXConnections">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnectionMasks" type="s0:DXConnection" />
    </s:sequence>
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="BrowsePath" type="s:string" />
    <s:attribute name="Recursive" type="s:boolean" />
  </s:complexType>
</s:element>

<s:element name="DeleteDXConnectionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="DeleteDXConnectionsResult" type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="MaskErrors" type="s:QName" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.2.6 CopyDefaultDXConnectionsAttributes

```
<s:element name="CopyDefaultDXConnectionAttributes">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="DXConnectionMasks" type="s0:DXConnection" />
    </s:sequence>

    <s:attribute name="BrowsePath" type="s:string" />
    <s:attribute name="LocaleID" type="s:string" />
    <s:attribute name="Recursive" type="s:boolean" />
  </s:complexType>
</s:element>
<s:element name=" CopyDefaultDXConnectionAttributesResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="CopyDefaultDXConnectionAttributesResult"
                                                  type="s0:DXGeneralResponse" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="MaskErrors" type="s:QName" />
      <s:element minOccurs="0" maxOccurs="unbounded"
                 name="Errors" nillable="true" type="s0:OPCError" />
    </s:sequence>
    <s:attribute name="RevisedLocaleID" type="s:string" />
  </s:complexType>
</s:element>
```

A.2.2.3 ResetConfiguration

```
<s:element name=" ResetConfiguration">
  <s:complexType>

  </s:complexType>
</s:element>
<s:element name="ResetConfigurationResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="ResetConfigurationResult" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
```

Appendix B. IDL Implementation

The OPC DX IDL defines the mapping of the configuration interface to COM.

B.1. Implementation Guidelines

B.1.1. Error Handling

All methods return a failed HRESULT when an entire method fails (i.e. it was not possible to complete any of the requested operations). All methods return S_OK if all requested operations were completed successfully. The method will return S_FALSE if some requested operations were completed successfully, however, errors occurred while processing individual operations (e.g. for specific Source Servers or DXConnections). The DXGeneralResponse return parameter will contain the item specific error code.

Success codes and error codes for each service are listed as part of the response messages in the sections that describe those services. As a convention, success codes begin with "OPCDX_S_" and error codes begin with "OPCDX_E_".

B.1.2. Locale Ids and Verbose Errors

COM must clients will use the standard IOPCCommon interface to set the desired localeID and retrieve verbose error information for any of the HRESULTs. The DX proxy/stub DLL contains the English versions of the errors messages in its resource block. DX implementations may retrieve these messages with the WIN32 API called 'FormatMessage'.

B.1.3. Masks in Structures

The DXConnection and SourceServer structures defined in the IDL have a bit mask that specifies which elements of the structures contain valid data. Both the COM client and server must set and/or check these masks when using these structures. Implementors are expected to ignore the contents of a field if the co-responding mask bit is not set.

B.1.4. IDL

```
//==============================================================================
// TITLE: OpcDx.idl
//
// CONTENTS:
//
// The OPC Data Exchange 1.00 IDL file.
//
// This IDL was derived from the XML service description defined in the
// specification. The mapping the XML service and the IDL elements is
// a direct as possible (i.e. XML data types map onto IDL structures, XML
// services map onto IDL interfaces/methods et. al.). However, there is no
// standard representation in IDL for 'missing' attributes (i.e. values that
// are not contained in the XML). The IDL defines bit masks for each optional
// field in a structure. Clients and servers must use these masks to test
// and/or indicate whether a given field contains a meaningful value.
//
// (c) Copyright 2002 The OPC Foundation
// ALL RIGHTS RESERVED.
```

```
//
// DISCLAIMER:
//  This code is provided by the OPC Foundation solely to assist in
//  understanding and use of the appropriate OPC Specification(s) and may be
//  used as set forth in the License Grant section of the OPC Specification.
//  This code is provided as-is and without warranty or support of any sort
//  and is subject to the Warranty and Liability Disclaimers which appear
//  in the printed OPC Specification.
//
// MODIFICATION LOG:
//
// Date       By    Notes
// ---------- ---   -----
// 2002/09/09 RSA   First release.
// 2003/03/03 RSA   Added fields to ensure natural byte alignment for structures.
//

import "oaidl.idl";
import "ocidl.idl";
import "objidl.idl";

//==============================================================================
// Category ID declarations (defined as interfaces to ensure they show up in the typelib).

[uuid(A0C85BB8-4161-4fd6-8655-BB584601C9E0)] interface CATID_OPCDXServer10 : IUnknown {}

cpp_quote("#define CATID_OPCDXServer10 IID_CATID_OPCDXServer10")

//==============================================================================
// Structures, Typedefs and Enumerations.

typedef struct tagOpcDxItemIdentifier
{
       LPWSTR szItemPath;
       LPWSTR szItemName;
       LPWSTR szVersion;
       DWORD  dwReserved;
}
OpcDxItemIdentifier;

typedef struct tagOpcDxIdentifiedResult
{
    LPWSTR  szItemPath;
    LPWSTR  szItemName;
    LPWSTR  szVersion;
       HRESULT hResultCode;
}
OpcDxIdentifiedResult;

typedef struct tagOpcDxGeneralResponse
{
                          LPWSTR                 szConfigurationVersion;
                                        DWORD                  dwCount;
       [size_is(dwCount)] OpcDxIdentifiedResult* pIdentifiedResults;
                                        DWORD                  dwReserved;
}
OpcDxGeneralResponse;

typedef struct tagOpcDxSourceServer
{
    DWORD  dwMask;
       LPWSTR szItemPath;
       LPWSTR szItemName;
       LPWSTR szVersion;
       LPWSTR szName;
       LPWSTR szDescription;
       LPWSTR szServerType;
       LPWSTR szServerURL;
```

```
        BOOL    bDefaultSourceServerConnected;
        DWORD   dwReserved;
}
OpcDxSourceServer;

typedef struct tagOpcDxConnection
{
                                                            DWORD    dwMask;
                                              LPWSTR        szItemPath;
                                              LPWSTR        szItemName;
                                              LPWSTR  szVersion;
                                              DWORD  dwBrowsePathCount;
        [size_is(dwBrowsePathCount)] LPWSTR* pszBrowsePaths;
                                              LPWSTR  szName;
                                              LPWSTR  szDescription;
                                              LPWSTR  szKeyword;
                                              BOOL    bDefaultSourceItemConnected;
                                              BOOL    bDefaultTargetItemConnected;
                                              BOOL    bDefaultOverridden;
                                              VARIANT vDefaultOverrideValue;
                                              VARIANT vSubstituteValue;
                                              BOOL    bEnableSubstituteValue;
                                              LPWSTR  szTargetItemPath;
                                              LPWSTR  szTargetItemName;
                                              LPWSTR  szSourceServerName;
                                              LPWSTR  szSourceItemPath;
                                              LPWSTR  szSourceItemName;
                                              DWORD   dwSourceItemQueueSize;
                                              DWORD   dwUpdateRate;
                                              FLOAT   fltDeadBand;
                                              LPWSTR  szVendorData;
}
OpcDxConnection;

//===============================================================================
// IOPCConfiguration

[
    object,
       uuid(C130D281-F4AA-4779-8846-C2C4CB444F2A),
    pointer_default(unique)
]
interface IOPCConfiguration : IUnknown
{
    HRESULT GetServers(
        [out]                      DWORD*                pdwCount,
        [out, size_is(,*pdwCount)] OpcDxSourceServer**   ppServers
    );

    HRESULT AddServers(
        [in]                       DWORD                 dwCount,
        [in, size_is(dwCount)]     OpcDxSourceServer*    pServers,
        [out]                      OpcDxGeneralResponse* pResponse
    );

    HRESULT ModifyServers(
        [in]                       DWORD                 dwCount,
        [in, size_is(dwCount)]     OpcDxSourceServer*    pServers,
        [out]                      OpcDxGeneralResponse* pResponse
    );

    HRESULT DeleteServers(
        [in]                       DWORD                 dwCount,
        [in, size_is(dwCount)]     OpcDxItemIdentifier*  pServers,
        [out]                      OpcDxGeneralResponse* pResponse
    );

    HRESULT CopyDefaultServerAttributes(
            [in]                                    BOOL                    bConfigToStatus,
        [in]                        DWORD                   dwCount,
        [in, size_is(dwCount)]      OpcDxItemIdentifier*  pServers,
        [out]                       OpcDxGeneralResponse* pResponse
    );

    HRESULT QueryDXConnections(
        [in]                          LPWSTR                szBrowsePath,
        [in]                          DWORD                 dwNoOfMasks,
        [in, size_is(dwNoOfMasks)]    OpcDxConnection*      pDXConnectionMasks,
            [in]                                               BOOL                    bRecursive,
        [out, size_is(,dwNoOfMasks)] HRESULT**             ppErrors,
        [out]                         DWORD*                pdwCount,
        [out, size_is(,*pdwCount)]    OpcDxConnection**     ppConnections
    );

    HRESULT AddDXConnections(
        [in]                          DWORD                  dwCount,
        [in, size_is(dwCount)]        OpcDxConnection*       pConnections,
        [out]                         OpcDxGeneralResponse*  pResponse
    );

    HRESULT UpdateDXConnections(
            [in]                             LPWSTR                 szBrowsePath,
        [in]                          DWORD                  dwNoOfMasks,
        [in, size_is(dwNoOfMasks)]    OpcDxConnection*       pDXConnectionMasks,
            [in]                                               BOOL                    bRecursive,
            [in]                             OpcDxConnection*       pDXConnectionDefinition,
        [out, size_is(,dwNoOfMasks)] HRESULT**              ppErrors,
        [out]                         OpcDxGeneralResponse*  pResponse
    );

    HRESULT ModifyDXConnections(
        [in]                          DWORD                  dwCount,
        [in, size_is(dwCount)]        OpcDxConnection*       pDXConnectionDefinitions,
        [out]                         OpcDxGeneralResponse*  pResponse
    );

    HRESULT DeleteDXConnections(
            [in]                             LPWSTR                 szBrowsePath,
        [in]                          DWORD                  dwNoOfMasks,
        [in, size_is(dwNoOfMasks)]    OpcDxConnection*       pDXConnectionMasks,
            [in]                                               BOOL                    bRecursive,
        [out, size_is(,dwNoOfMasks)] HRESULT**              ppErrors,
        [out]                         OpcDxGeneralResponse*  pResponse
    );

    HRESULT CopyDXConnectionDefaultAttributes(
            [in]                                    BOOL                    bConfigToStatus,
            [in]                             LPWSTR                 szBrowsePath,
        [in]                          DWORD                  dwNoOfMasks,
        [in, size_is(dwNoOfMasks)]    OpcDxConnection*       pDXConnectionMasks,
            [in]                                               BOOL                    bRecursive,
        [out, size_is(,dwNoOfMasks)] HRESULT**              ppErrors,
        [out]                         OpcDxGeneralResponse*  pResponse
    );

    HRESULT ResetConfiguration(
        [in]  LPWSTR  szConfigurationVersion,
        [out] LPWSTR* pszConfigurationVersion
    );
};

//================================================================================
// Type Library

[
       uuid(3CA18B30-1088-47d5-8952-0B12B027ED32),
        version(1.00),
    helpstring("OPC Data eXchange 1.00 Type Library")
]
library OpcDxLib
{
    interface CATID_OPCDXServer10;
    interface IOPCConfiguration;

        module Names
        {
                // category description strings.
                const LPCWSTR OPC_CATEGORY_DESCRIPTION_DX10          = L"OPC Data Exchange Servers
Version 1.0";

                // common names.
                const LPCWSTR OPCDX_NAMESPACE_V10                    =
L"http://opcfoundation.org/webservices/OPCDX/10";
                const LPCWSTR OPCDX_DATABASE_ROOT                    = L"DX";
                const LPCWSTR OPCDX_SEPARATOR                                = L"/";
                const LPCWSTR OPCDX_ITEM_PATH                        = L"ItemPath";
                const LPCWSTR OPCDX_ITEM_NAME                        = L"ItemName";
                const LPCWSTR OPCDX_VERSION                          = L"Version";

                // server status.
                const LPCWSTR OPCDX_SERVER_STATUS_TYPE               = L"DXServerStatus";
                const LPCWSTR OPCDX_SERVER_STATUS                    = L"ServerStatus";
                const LPCWSTR OPCDX_CONFIGURATION_VERSION            = L"ConfigurationVersion";
                const LPCWSTR OPCDX_SERVER_STATE                     = L"ServerState";
                const LPCWSTR OPCDX_CONNECTION_COUNT                 = L"DXConnectionCount";
                const LPCWSTR OPCDX_MAX_CONNECTIONS                  = L"MaxDXConnections";
                const LPCWSTR OPCDX_SERVER_ERROR_ID                  = L"ErrorID";
                const LPCWSTR OPCDX_SERVER_ERROR_DIAGNOSTIC          = L"ErrorDiagnostic";
                const LPCWSTR OPCDX_DIRTY_FLAG                       = L"DirtyFlag";
                const LPCWSTR OPCDX_SOURCE_SERVER_TYPES              = L"SourceServerTypes";
                const LPCWSTR OPCDX_MAX_QUEUE_SIZE                   = L"MaxQueueSize";

                // connection configuration.
                const LPCWSTR OPCDX_CONNECTIONS_ROOT                 = L"DXConnectionsRoot";
                const LPCWSTR OPCDX_CONNECTION_TYPE                  = L"DXConnection";
                const LPCWSTR OPCDX_CONNECTION_NAME                  = L"Name";
                const LPCWSTR OPCDX_CONNECTION_BROWSE_PATHS          = L"BrowsePath";
                const LPCWSTR OPCDX_CONNECTION_VERSION               = L"Version";
                const LPCWSTR OPCDX_CONNECTION_DESCRIPTION           = L"Description";
                const LPCWSTR OPCDX_CONNECTION_KEYWORD               = L"Keyword";
                const LPCWSTR OPCDX_DEFAULT_SOURCE_ITEM_CONNECTED    =
L"DefaultSourceItemConnected";
                const LPCWSTR OPCDX_DEFAULT_TARGET_ITEM_CONNECTED    =
L"DefaultTargetItemConnected";
                const LPCWSTR OPCDX_DEFAULT_OVERRIDDEN               = L"DefaultOverridden";
                const LPCWSTR OPCDX_DEFAULT_OVERRIDE_VALUE           = L"DefaultOverrideValue";
                const LPCWSTR OPCDX_ENABLE_SUBSTITUTE_VALUE          = L"EnableSubstituteValue";
                const LPCWSTR OPCDX_SUBSTITUTE_VALUE                 = L"SubstituteValue";
                const LPCWSTR OPCDX_TARGET_ITEM_PATH                 = L"TargetItemPath";
                const LPCWSTR OPCDX_TARGET_ITEM_NAME                 = L"TargetItemName";
                const LPCWSTR OPCDX_CONNECTION_SOURCE_SERVER_NAME    = L"SourceServerName";
                const LPCWSTR OPCDX_SOURCE_ITEM_PATH                 = L"SourceItemPath";
                const LPCWSTR OPCDX_SOURCE_ITEM_NAME                 = L"SourceItemName";
                const LPCWSTR OPCDX_SOURCE_ITEM_QUEUE_SIZE           = L"QueueSize";
                const LPCWSTR OPCDX_UPDATE_RATE                      = L"UpdateRate";
                const LPCWSTR OPCDX_DEADBAND                         = L"Deadband";
                const LPCWSTR OPCDX_VENDOR_DATA                      = L"VendorData";

                // connection status.
                const LPCWSTR OPCDX_CONNECTION_STATUS                = L"Status";
                const LPCWSTR OPCDX_CONNECTION_STATUS_TYPE           = L"DXConnectionStatus";
                const LPCWSTR OPCDX_CONNECTION_STATE                 = L"DXConnectionState";
                const LPCWSTR OPCDX_WRITE_VALUE                      = L"WriteValue";
                const LPCWSTR OPCDX_WRITE_TIMESTAMP                  = L"WriteTimestamp";
                const LPCWSTR OPCDX_WRITE_QUALITY                      = L"WriteQuality";
                const LPCWSTR OPCDX_WRITE_ERROR_ID                     = L"WriteErrorID";
                const LPCWSTR OPCDX_WRITE_ERROR_DIAGNOSTIC             = L"WriteErrorDiagnostic";
                const LPCWSTR OPCDX_SOURCE_VALUE                       = L"SourceValue";
                const LPCWSTR OPCDX_SOURCE_TIMESTAMP                   = L"SourceTimestamp";
                const LPCWSTR OPCDX_SOURCE_QUALITY                     = L"SourceQuality";
                const LPCWSTR OPCDX_SOURCE_ERROR_ID                    = L"SourceErrorID";
                const LPCWSTR OPCDX_SOURCE_ERROR_DIAGNOSTIC            = L"SourceErrorDiagnostic";
                const LPCWSTR OPCDX_ACTUAL_UPDATE_RATE                 = L"ActualUpdateRate";
                const LPCWSTR OPCDX_QUEUE_HIGH_WATER_MARK              = L"QueueHighWaterMark";
                const LPCWSTR OPCDX_QUEUE_FLUSH_COUNT                  = L"QueueFlushCount";
                const LPCWSTR OPCDX_SOURCE_ITEM_CONNECTED              = L"SourceItemConnected";
                const LPCWSTR OPCDX_TARGET_ITEM_CONNECTED              = L"TargetItemConnected";
                const LPCWSTR OPCDX_OVERRIDDEN                         = L"Overridden";
                const LPCWSTR OPCDX_OVERRIDE_VALUE                     = L"OverrideValue";

                // source server configuration.
                const LPCWSTR OPCDX_SOURCE_SERVERS_ROOT                = L"SourceServers";
                const LPCWSTR OPCDX_SOURCE_SERVER_TYPE                 = L"SourceServer";
                const LPCWSTR OPCDX_SOURCE_SERVER_NAME                 = L"Name";
                const LPCWSTR OPCDX_SOURCE_SERVER_VERSION              = L"Version";
                const LPCWSTR OPCDX_SOURCE_SERVER_DESCRIPTION          = L"Description";
                const LPCWSTR OPCDX_SERVER_URL                         = L"ServerURL";
                const LPCWSTR OPCDX_SERVER_TYPE                        = L"ServerType";
                const LPCWSTR OPCDX_DEFAULT_SOURCE_SERVER_CONNECTED =
L"DefaultSourceServerConnected";

                // source server status.
                const LPCWSTR OPCDX_SOURCE_SERVER_STATUS_TYPE          = L"DXSourceServerStatus";
                const LPCWSTR OPCDX_SOURCE_SERVER_STATUS               = L"Status";
                const LPCWSTR OPCDX_SERVER_CONNECT_STATUS              = L"ConnectStatus";
                const LPCWSTR OPCDX_SOURCE_SERVER_ERROR_ID             = L"ErrorID";
                const LPCWSTR OPCDX_SOURCE_SERVER_ERROR_DIAGNOSTIC     = L"ErrorDiagnostic";
                const LPCWSTR OPCDX_LAST_CONNECT_TIMESTAMP             = L"LastConnectTimestamp";
                const LPCWSTR OPCDX_LAST_CONNECT_FAIL_TIMESTAMP        = L"LastConnectFailTimestamp";
                const LPCWSTR OPCDX_CONNECT_FAIL_COUNT                 = L"ConnectFailCount";
                const LPCWSTR OPCDX_PING_TIME                          = L"PingTime";
                const LPCWSTR OPCDX_LAST_DATA_CHANGE_TIMESTAMP         = L"LastDataChangeTimestamp";
                const LPCWSTR OPCDX_SOURCE_SERVER_CONNECTED            = L"SourceServerConnected";

                // quality
                const LPCWSTR OPCDX_QUALITY                            = L"DXQuality";
                const LPCWSTR OPCDX_QUALITY_STATUS                     = L"Quality";
                const LPCWSTR OPCDX_LIMIT_BITS                         = L"LimitBits";
                const LPCWSTR OPCDX_VENDOR_BITS                        = L"VendorBits";

                // error
                const LPCWSTR OPCDX_ERROR                              = L"OPCError";
                const LPCWSTR OPCDX_ERROR_ID                           = L"ID";
                const LPCWSTR OPCDX_ERROR_TEXT                         = L"Text";

                // source server url scheme names.
                const LPCWSTR OPCDX_SOURCE_SERVER_URL_SCHEME_OPCDA     = L"opcda";
                const LPCWSTR OPCDX_SOURCE_SERVER_URL_SCHEME_XMLDA     = L"http";
        }

        // possible quality status values.
        module QualityStatusName
        {
                const LPCWSTR OPCDX_QUALITY_BAD                              = L"bad";
                const LPCWSTR OPCDX_QUALITY_BAD_CONFIG_ERROR                 =
L"badConfigurationError";
                const LPCWSTR OPCDX_QUALITY_BAD_NOT_CONNECTED                = L"badNotConnected";
                const LPCWSTR OPCDX_QUALITY_BAD_DEVICE_FAILURE               = L"badDeviceFailure";
                const LPCWSTR OPCDX_QUALITY_BAD_SENSOR_FAILURE               = L"badSensorFailure";
                const LPCWSTR OPCDX_QUALITY_BAD_LAST_KNOWN_VALUE             = L"badLastKnownValue";
                const LPCWSTR OPCDX_QUALITY_BAD_COMM_FAILURE                 = L"badCommFailure";
                const LPCWSTR OPCDX_QUALITY_BAD_OUT_OF_SERVICE               = L"badOutOfService";
                const LPCWSTR OPCDX_QUALITY_UNCERTAIN                         = L"uncertain";
                const LPCWSTR OPCDX_QUALITY_UNCERTAIN_LAST_USABLE_VALUE   = 
L"uncertainLastUsableValue";
                const LPCWSTR OPCDX_QUALITY_UNCERTAIN_SENSOR_NOT_ACCURATE =
L"uncertainSensorNotAccurate";
                const LPCWSTR OPCDX_QUALITY_UNCERTAIN_EU_EXCEEDED           =
L"uncertainEUExceeded";
                const LPCWSTR OPCDX_QUALITY_UNCERTAIN_SUB_NORMAL          = L"uncertainSubNormal";
                const LPCWSTR OPCDX_QUALITY_GOOD                          = L"good";
                const LPCWSTR OPCDX_QUALITY_GOOD_LOCAL_OVERRIDE           = L"goodLocalOverride";
        }

        // possible limit status values.
        module LimitStatusName
        {
                const LPCWSTR OPCDX_LIMIT_NONE     = L"none";
                const LPCWSTR OPCDX_LIMIT_LOW      = L"low";
                const LPCWSTR OPCDX_LIMIT_HIGH     = L"high";
                const LPCWSTR OPCDX_LIMIT_CONSTANT = L"constant";
        }

        // possible source server interface types.
        module ServerTypeName
        {
                const LPCWSTR OPCDX_SERVER_TYPE_COM_DA10  = L"COM-DA1.0";
                const LPCWSTR OPCDX_SERVER_TYPE_COM_DA204 = L"COM-DA2.04";
                const LPCWSTR OPCDX_SERVER_TYPE_COM_DA205 = L"COM-DA2.05";
                const LPCWSTR OPCDX_SERVER_TYPE_COM_DA30  = L"COM-DA3.0";
                const LPCWSTR OPCDX_SERVER_TYPE_XML_DA10  = L"XML-DA1.0";
        }

        // possible DX server states.
        module ServerStateName
        {
                const LPCWSTR OPCDX_SERVER_STATE_RUNNING    = L"running";
                const LPCWSTR OPCDX_SERVER_STATE_FAILED     = L"failed";
                const LPCWSTR OPCDX_SERVER_STATE_NOCONFIG   = L"noConfig";
                const LPCWSTR OPCDX_SERVER_STATE_SUSPENDED  = L"suspended";
                const LPCWSTR OPCDX_SERVER_STATE_TEST       = L"test";
                const LPCWSTR OPCDX_SERVER_STATE_COMM_FAULT = L"commFault";
                const LPCWSTR OPCDX_SERVER_STATE_UNKNOWN    = L"unknown";
        }

        // possible source server connect states.
        module ConnectStatusName
        {
                const LPCWSTR OPCDX_CONNECT_STATUS_CONNECTED    = L"connected";
                const LPCWSTR OPCDX_CONNECT_STATUS_DISCONNECTED = L"disconnected";
                const LPCWSTR OPCDX_CONNECT_STATUS_CONNECTING   = L"connecting";
                const LPCWSTR OPCDX_CONNECT_STATUS_FAILED       = L"failed";
        }

        // possible connection states.
        module ConnectionStateName
        {
                const LPCWSTR OPCDX_CONNECTION_STATE_INITIALIZING                 =
L"initializing";
                const LPCWSTR OPCDX_CONNECTION_STATE_OPERATIONAL                  = L"operational";
                const LPCWSTR OPCDX_CONNECTION_STATE_DEACTIVATED                  = L"deactivated";
                const LPCWSTR OPCDX_CONNECTION_STATE_SOURCE_SERVER_NOT_CONNECTED =
L"sourceServerNotConnected";
                const LPCWSTR OPCDX_CONNECTION_STATE_SUBSCRIPTION_FAILED          =
L"subscriptionFailed";
                const LPCWSTR OPCDX_CONNECTION_STATE_TARGET_ITEM_NOT_FOUND        =
L"targetItemNotFound";
        }

        // source server type enumeration.
```

```
    typedef enum tagOpcDxServerType
    {
            OpcDxServerType_COM_DA10  = 1,
            OpcDxServerType_COM_DA204,
            OpcDxServerType_COM_DA205,
            OpcDxServerType_COM_DA30,
            OpcDxServerType_XML_DA10
    }
    OpcDxServerType;

    // source server state enumeration - intentionally compatible with OPCSERVERSTATE.
    typedef enum tagOpcDxServerState
    {
            OpcDxServerState_RUNNING = 1,
            OpcDxServerState_FAILED,
            OpcDxServerState_NOCONFIG,
            OpcDxServerState_SUSPENDED,
            OpcDxServerState_TEST,
            OpcDxServerState_COMM_FAULT,
            OpcDxServerState_UNKNOWN
    }
    OpcDxServerState;

    // connection state enumeration.
    typedef enum tagOpcDxConnectionState
    {
            OpcDxConnectionState_INITIALIZING = 1,
            OpcDxConnectionState_OPERATIONAL,
            OpcDxConnectionState_DEACTIVATED,
            OpcDxConnectionState_SOURCE_SERVER_NOT_CONNECTED,
            OpcDxConnectionState_SUBSCRIPTION_FAILED,
            OpcDxConnectionState_TARGET_ITEM_NOT_FOUND
    }
    OpcDxConnectionState;

    // source server connect status enumeration.
    typedef enum tagOpcDxConnectStatus
    {
            OpcDxConnectStatus_CONNECTED = 1,
            OpcDxConnectStatus_DISCONNECTED,
            OpcDxConnectStatus_CONNECTING,
            OpcDxConnectStatus_FAILED
    }
    OpcDxConnectStatus;

    // bit masks for optional fields in source server or connection structures.
typedef enum tagOpcDxMask
{
            OpcDxMask_None                          = 0x0,
            OpcDxMask_ItemPath                      = 0x1,
            OpcDxMask_ItemName                      = 0x2,
            OpcDxMask_Version                       = 0x4,
            OpcDxMask_BrowsePaths                   = 0x8,
            OpcDxMask_Name                          = 0x10,
            OpcDxMask_Description                   = 0x20,
            OpcDxMask_Keyword                       = 0x40,
            OpcDxMask_DefaultSourceItemConnected    = 0x80,
            OpcDxMask_DefaultTargetItemConnected    = 0x100,
            OpcDxMask_DefaultOverridden             = 0x200,
            OpcDxMask_DefaultOverrideValue          = 0x400,
            OpcDxMask_SubstituteValue               = 0x800,
            OpcDxMask_EnableSubstituteValue         = 0x1000,
            OpcDxMask_TargetItemPath                = 0x2000,
            OpcDxMask_TargetItemName                = 0x4000,
            OpcDxMask_SourceServerName              = 0x8000,
            OpcDxMask_SourceItemPath                = 0x10000,
            OpcDxMask_SourceItemName                = 0x20000,
            OpcDxMask_SourceItemQueueSize           = 0x40000,
                OpcDxMask_UpdateRate                   = 0x80000,
                OpcDxMask_DeadBand                     = 0x100000,
                OpcDxMask_VendorData                   = 0x200000,
                OpcDxMask_ServerType                   = 0x400000,
                OpcDxMask_ServerURL                    = 0x800000,
                OpcDxMask_DefaultSourceServerConnected = 0x1000000,
        OpcDxMask_All                          = 0x7FFFFFFF
    }
    OpcDxMask;
};
```