FOUNDATION

OPC Complex Data Specification

Version 1.00

Released

December 10, 2003

Specification Type	Industry Standard Specification		
Title:	**OPC Complex Data Specification**	Date:	December 10, 2003
Version:	Version 1.00	Soft Source:	MS-Word OpcCmplxData
Author:	OPC Complex Data Working Group	Status:	Released

Synopsis:

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

Required Runtime Environment:

This specification requires Windows 95, Windows NT 4.0 or later.

NON-EXCLUSIVE LICENSE AGREEMENT

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

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

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

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

LICENSE GRANT:

Subject to the terms and conditions of this Agreement, the OPC Foundation hereby grants to User a non-exclusive, royalty-free, limited license to use, copy, display and distribute the OPC Materials in order to make, use, sell or otherwise distribute any products and/or product literature that are compliant with the standards included in the OPC Materials. User may not distribute OPC Materials outside of the Active Member organization to which User belongs unless the OPC Foundation has explicitly designated the OPC Material for public use.

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

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

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

1. User is requested to acknowledge the use of the OPC Materials and provide a link to the OPC Foundation home page www.opcfoundation.org from the About box of the User’s or Active Member’s application(s).
2. User may include the source code, modified source code, built binaries or modified built binaries within User’s own applications for either personal or commercial use except for:
   a) The OPC Foundation software source code or binaries cannot be sold as is, either individually or together.
   b) The OPC Foundation software source code or binaries cannot be modified and then sold as a library component, either individually or together.

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

WARRANTY AND LIABILITY DISCLAIMERS:

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

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

GENERAL PROVISIONS:

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

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

The Software and Documentation are provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor/ manufacturer is the OPC Foundation, 16101 N 82nd Street Suite 3B, Scottsdale, AZ 85260-1830.

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

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

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

Version 1.00 Highlights

This revision is the initial version of the Complex Data specification. It specifies new OPC DA item properties that define the structure of complex data items. Complex data items are OPC items whose values are constructed. These properties can be used by any version of OPC DA, including OPC XML DA.

Table of Contents

1 Introduction

1.1 Overview

This specification describes how to represent and access complex data within the existing OPC Data Access (DA) framework. It also describes how existing complex type systems such as the eXtensible Markup Language (XML) Schema can be used to describe complex data.

1.2 Audience

This specification is intended as reference material for developers of OPC compliant Client and Server applications. It is assumed that the reader is familiar with the OPC Data Access specifications, Microsoft COM/DCOM technology, XML schemas, and the needs of the Process Control industry.

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

1.3 Deliverables

The deliverables from the OPC Foundation with respect to the OPC Complex Data Project include the OPC Complex Data 1.00 Specification and OPC Complex Data 1.00 sample code, available from the OPC Foundation Web Site (members only).

1.4 Errata

Any clarifications or corrections to this specification found after publication will be posted to the following web page:

http://www.opcfoundation.org/forum/viewtopic.php?t=346

2 OPC Complex Data Fundamentals

Complex data is a term that describes OPC Data Access (DA) items whose values are constructed. OPC Complex Data provides a mechanism for OPC DA clients to discover the structure of the data values. This section introduces OPC Data Access and how OPC Complex Data relates to it.

2.1 OPC Overview

An OPC DA Client can connect to one or more OPC DA Servers provided by one or more vendors.

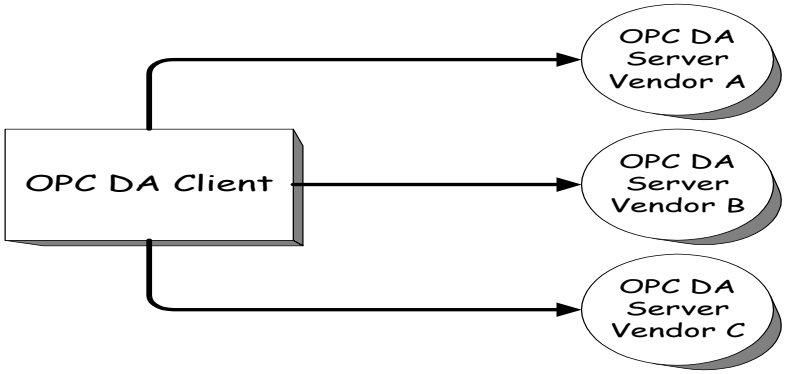


Figure 1 - OPC DA Client

Different vendors may provide OPC DA Servers. Vendor supplied code determines the devices and data to which each server has access, the data names, and the details about how the server physically accesses that data, and how it makes this data available to OPC DA clients, including its structure. Specifics on naming conventions are supplied in a subsequent section. Each OPC DA server allows OPC DA clients supplied by different vendors to simultaneously access its data.

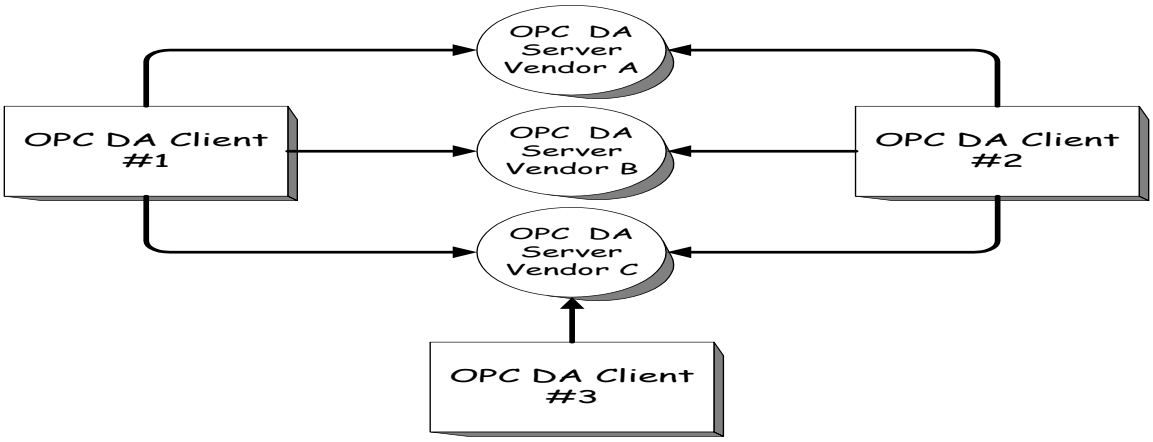


Figure 2 - OPC DA Client/Server Relationship

The data exposed by an OPC DA server is usually stored in an underlying physical system such as a programmable logic controller (PLC). The server represents each unit of data as a DA item which has an associated unique identifier called an Item ID. The server organizes these DA items into a hierarchical namespace where related items (i.e. I/O points connected to a single piece of equipment)

are stored in a single branch. Each DA item has associated metadata (called Properties) that describe what the item is and how it may be accessed.

OPC DA clients locate DA items of interest by browsing the DA server namespace. The client must obtain the Item ID for an item from server before it can access the item. The client uses the Item ID to perform direct I/O operations such as reading and writing the current value, quality or timestamp. A client may also establish subscriptions that require the OPC DA server to notify the client when the value or quality of an item changes.

2.2 Where OPC Complex Data Fits

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

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

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

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

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

OPC Complex Data solves these problems by:

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

3 Complex Data Model

3.1 Definition

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

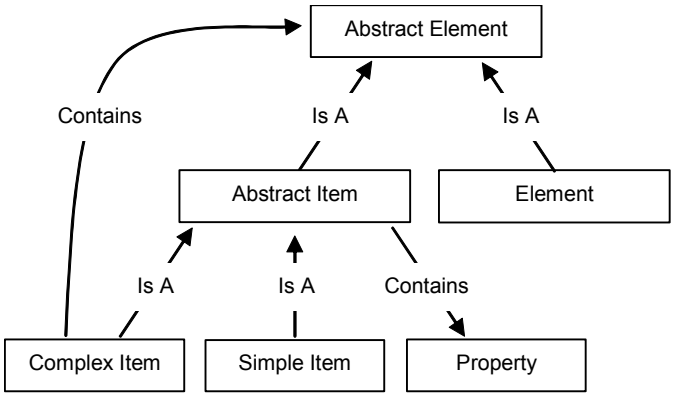


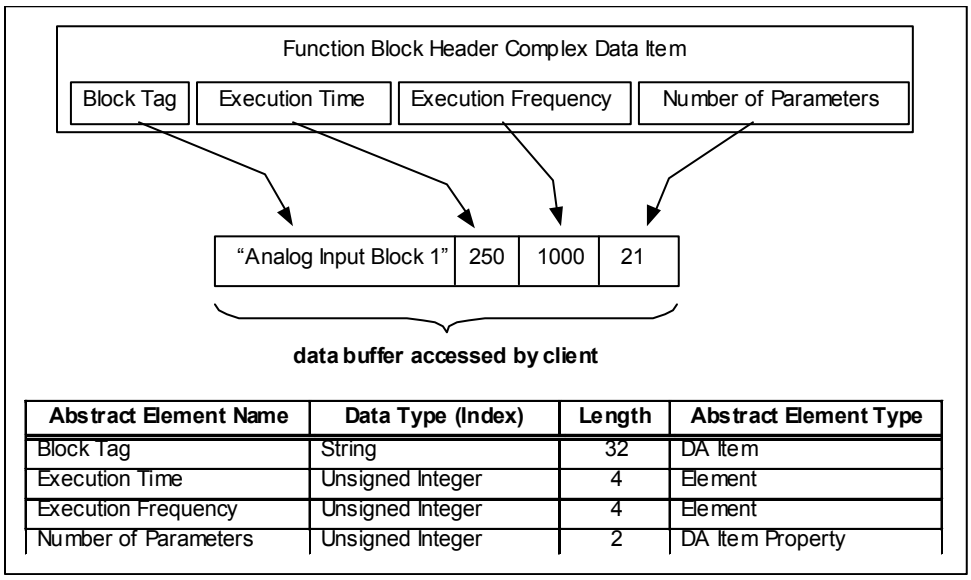
Figure 3 - OPC DA Client

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

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

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

Figure 4 below provides an example of a complex data item that exposes some abstract elements as "DA Items", "Item Properties", and "Elements". In this example, the Block Tag is exposed by the server as a DA Item. Therefore, it is accessible through the Function Block Header complex item, but also as a DA Item of its own. The Number of Parameters can also be accessed both as part of the Function Block Header complex item, and separately as a property of that complex item. Finally the Execution Time and Execution Frequency can only be accessed as part of the Function Block Header complex item.



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

Figure 4 - Example Complex Data Item

3.2 Complex Data Type Descriptions

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

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

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

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

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

3.3 Complex Data Type Item Properties

Complex Data Type Descriptions are provided through DA Item Properties, as shown in Figure 5 below.

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

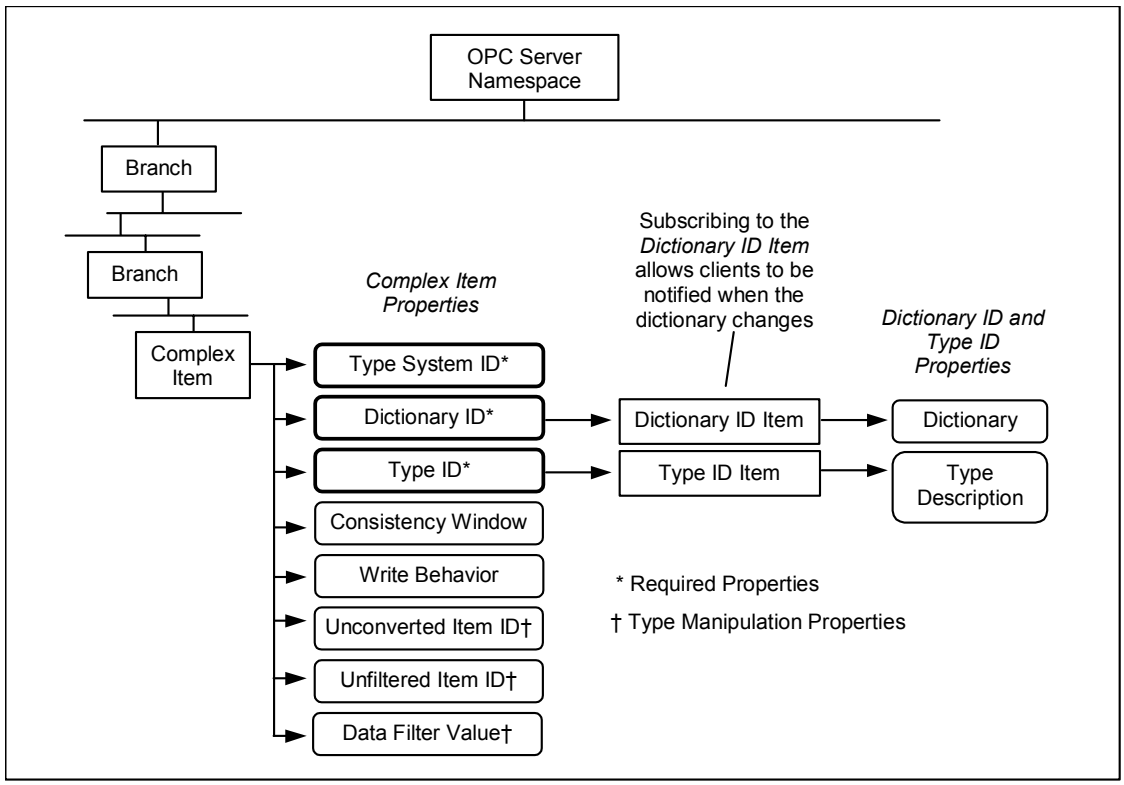


Figure 5 – Complex Data Properties

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

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

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

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

The Item ID for the Dictionary ID Item or the Type ID Item should never be the same as the Dictionary ID or the Type ID. In addition, a Dictionary ID Item or the Type ID Item may be part of the Complex Data Namespace (see Section 3.4). The value of the Dictionary ID Item is the same as the value of the Dictionary ID property for a complex data item. This allows clients to subscribe to it to detect changes in the dictionary. Similarily, the value of the Type ID Item is the Type ID, however, it is not allowed to change, therefore, DA clients have no need to subscribe to it.

Servers associate a Dictionary ID Item, as shown in Figure 5, with a given ictionary. The value of the Dictionary ID Item is the Dictionary ID, allowing clients to subscribe to it to detect changes in the

dictionary. This item also provides client access to the Dictionary property. This property contains the dictionary. Standard OPC DA interfaces can be used to obtain the Item ID of the Dictionary ID Item from the Dictionary ID property.

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

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

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

The following table summarizes the Item Properties defined to support complex data. XML qualified names are created for these properties by removing the spaces and making the first character lower case. For example, the XML qualified name for Type System ID is "typeSystemID". The namespace for these XML qualified names is http//opcfoundation.org/webservices/XMLDA/1.0/.

Prop ID	Prop Name	Data Type	Usage
600	Type System ID	string	Mandatory
601	Dictionary ID	string	Mandatory
602	Type ID	string	Mandatory
603	Dictionary	BLOB[1]	Optional
604	Type Description	BLOB[1]	Optional[2]
605	Consistency Window	string	Optional
606	Write Behavior	string	Defautls to "All or Nothing" if the complex data item is writable. Not used for Read-Only items.
607	Unconverted Item ID	string	The ID of the item that exposes the same complex data value in its native format. This property is mandatory for items that implement complex data type conversions (see Section 7).
608	Unfiltered Item ID	string	The ID the item that exposes the same complex data value without any data filter or with the default query applied to it. It is mandatory for items that implement complex data filters or queries (see Section 8).
609	Data Filter Value	string	The value of the filter that is currently applied to the item. It is mandatory for items that implement complex data filters or queries (see Section 8).

NOTES:
1 The term BLOB is used here, and below, to represent a sequence of bytes whose data type depends on the Type System.
2 Some servers may not have access to the type descriptions. In these cases, the client needs to be able to determine the structure of the data based on the Type ID.

3.3.1 Type System ID Property

This property is identifies the Type System. If the server supports the CPX namespace (see Section 3.4), then this identifier is used as the name of the Type System branch in the namespace. A DA client must recognize the type system in order to make use of any of the type description information.

The Type System ID is assigned by the DA server vendor, the entity responsible for the Type System, or by the OPC Foundation. The following Type Systems are defined by this specification.

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

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

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

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

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

3.3.2 Dictionary ID Property

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

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

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

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

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

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

3.3.3 Type ID Property

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

3.3.4 Dictionary Property

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

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

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

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

3.3.5 Type Description Property

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

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

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

3.3.6 Consistency Window Property

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

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

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

3.3.7 Write Behavior Property

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

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

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

3.4 Complex Data Namespace

In addition to providing support for the Complex Data Item Properties just described, OPC Servers may also extend their namespace to allow clients to browse for supported complex data descriptions. The "CPX" branch is reserved for this purpose. Its relationship to Complex Data Item properties is shown in Figure 6.

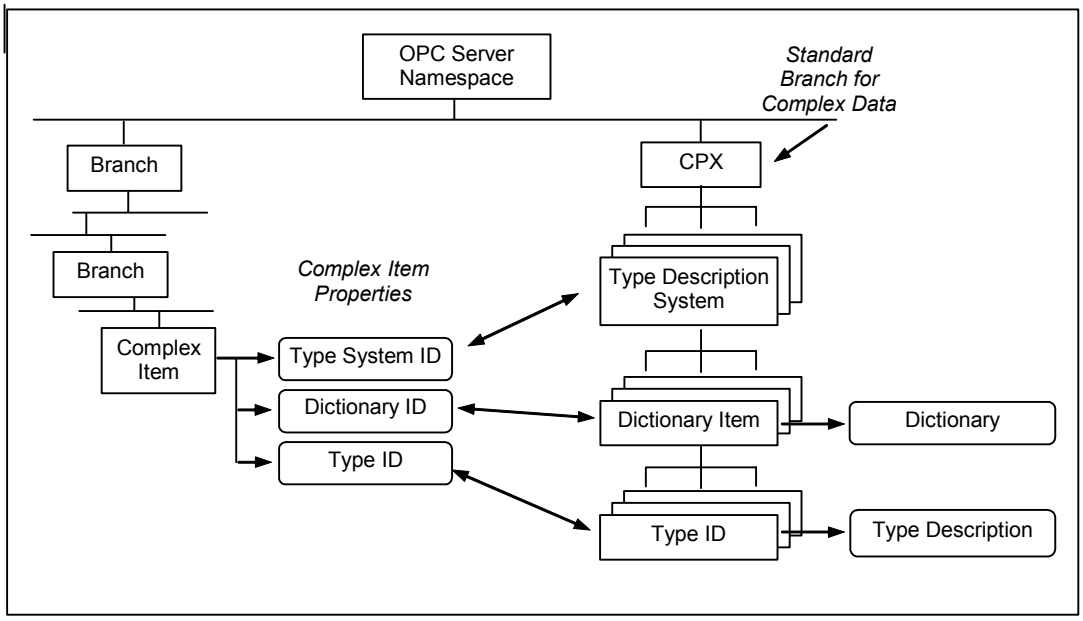


Figure 6 – Complex Data Namespace

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

Item/Branch	Description
Type System ID	Used to identify the type systems supported by the OPC Server. The name of the type system is used as the name of this branch and also as the value of the corresponding type system property (see Section 3.3.1).
Dictionary ID	Used to identify the dictionaries supported for each type system. The name of the branch identifies the dictionary independent of its version. The value of this item identifies the dictionary and its current version. This value is used as the value of the Dictionary ID Item Property (see Section 0). Clients may subscribe to this item to detect changes to the dictionary. The associated Dictionary property (see Section 3.3.3) contains the latest copy of the dictionary.
Type ID	Used to identify the type descriptions within each dictionary. The Type ID is used as the name of this branch and also as the value of the corresponding Type ID Item Property (see Section 3.3.3). If the type system provides both a Type Name and a Type ID, the Type Name is used for the value of this item. Otherwise, the Type ID is used.

4 Complex Data Behavior

Type Conversion :

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

OnDataChange :

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

Quality :

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

Timestamp :

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

Time Consistency Window :

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

Write :

Complex data items may be written by clients. When written, not all elements of the complex data item may be writable. In this case, if the client supplies the entire buffer in a write request, the server ignores the values of non-writable elements.

In the case where the value for one or more of the writable elements is invalid, or cannot be applied, the server rejects the entire write request if the “Write Behavior” Item Property indicates “All or Nothing”. If this property indicates “Best Effort”, then the write succeeds if one element is successfully written by the server.

5 Complex Data Examples

5.1 XML Schema

The elements of a structure are described in the following table:

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

An XML schema that describes this structure is:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
  xmlns=http://opcfoundation.org/ComplexData/Sample1.xsd
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  targetNamespace=http://opcfoundation.org/ComplexData/Sample1.xsd
  elementFormDefault="qualified"
>
<xs:schema>
  <xs:element name="Connection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DeviceName" type="xs:string" />
        <xs:element name="DeviceSettings" type="ArrayOfString" />
        <xs:element name="WaitTime" type="xs:unsignedInt" />
        <xs:element name="Status”>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ConnectState" type="xs:boolean" />
              <xs:element name="LastConnectTime" type="xs:dateTime" />
              <xs:element name="ConnectFailCount" type="xs:unsignedInt" />
              <xs:element name="IsConnected" type="xs:boolean" />
            </xs:sequence>
           </xs:complexType>
         </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

This type can be exposed as the following three DA items in the DA server namespace

Item	Item Data Type
Connection	The entire structure as an XML document as a string
Status	The status elements in an XML document as a string
Connect State	Boolean

The items must have the following properties:

Property	Value
Connection	
Data Type	string
Access Rights	readOnly
Type System ID	XMLSchema
Dictionary ID	http://opcfoundation.org/ComplexData/Sample1.xsd
Type ID	Connection
Consistency Window	Unknown
Write Behavior	Best Effort
Status	
Data Type	string
Access Rights	readWritable
Type System ID	XMLSchema
Dictionary ID	http://opcfoundation.org/ComplexData/Sample1.xsd
Type ID	Connection/Status
Consistency Window	Unknown
Write Behavior	Best Effort
Connect State	
Data Type	boolean
Access Rights	writable

In addition the their values Dictionary ID and Type ID properties have associated item ids that reference items in the Complex Data Namespace defined above. The client must access these items in order fetch the Dictionary and/or Type Descriptions associated with these complex data items.

The items in the Complex Data Namespace must have the following properties:

Property	Value
CPX/XMLSchema/Sample1	
Data Type	string
Value	http://opcfoundation.org/ComplexData/Sample1.xsd
Access Rights	readOnly
Dictionary	The entire XML schema shown above.
CPX/XMLSchema/Sample1/Connection	
Data Type	string
Value	Connection
Access Rights	readOnly
Type Description	An XML document containing only those elements that match when XPath "/*/xs:element[@name='Connection']" is applied to the XML schema shown above.
CPX/XMLSchema/Sample1/Connection/Status	
Data Type	string
Value	Connection/Status
Access Rights	readOnly
Type Description	An XML document containing only the element that matches when XPath "/*/xs: element[@name='Connection']//xs:element[@name='Status']" is applied to the XML schema shown above.

The Connection and Status items have the same Dictionary ID (The URI for the XML Schema) and the same Dictionary item. In each case, the Type ID property specifies the name of the complex type defined within the schema. This property is required since it tells the DA client that which subset of the types described in the schema are exposed by the item.

The syntax of the Type ID is defined by the XML Schema type system. In this case the Type ID is the value of the 'name' attribute of a 'element' element within the Schema. Nested complex types have a Type ID which is the value of the 'name' attribute for all containing 'element' elements separated by the '/' character.

The Connect State item does not have the complex data properties since it is not exposing a complex type.

The Consistency Window for the Connection and Status items is 'Unknown' which indicates all contained elements are consistent with each other however the server does not have an absolute measure of the window size.

The Status item is writable however not all of the individual elements are writeable. This is not a problem since the write behavior is considered to be 'Best Effort'. This means the client can write an XML document that contains read only elements, however, the server will only write the values for the writable elements. It is worth noting that XML allows elements to be omitted, as a result, is easy for a client to construct an XML document for writing that only contains the elements the client wishes to change.

This example returns all values as part of the Connection item. In some cases, the server may wish only return the configuration related values as part of this item since they change less frequently than the status elements. This is not possible to do with the existing schema since it only describes two types: the entire connection or only the status elements. The DA server can handle this situation using a slightly different XML schema which is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
  xmlns=http://opcfoundation.org/ComplexData/Sample2.xsd
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  targetNamespace=http://opcfoundation.org/ComplexData/Sample2.xsd
  elementFormDefault="qualified"
>
  <xs:element name="Connection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Config”>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="DeviceName" type="xs:string" />
              <xs:element name="DeviceSettings" type="ArrayOfString" />
              <xs:element name="WaitTime" type="xs:unsignedInt" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Status”>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ConnectState" type="xs:boolean" />
              <xs:element name="LastConnectTime" type="xs:dateTime" />
              <xs:element name="ConnectFailCount" type="xs:unsignedInt" />
              <xs:element name="IsConnected" type="xs:boolean" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The properties for the Connection item would now become:

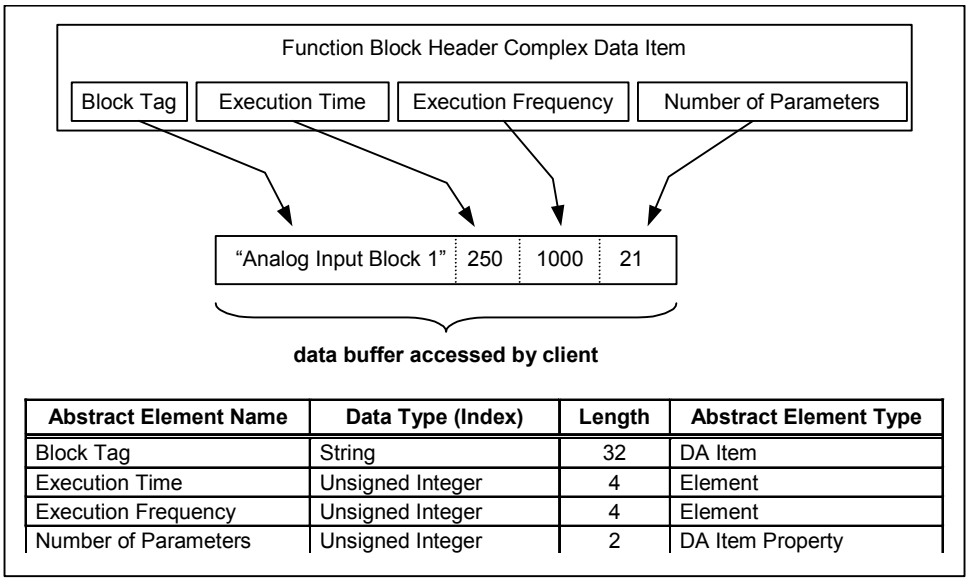
Property	Value
Connection	
Type System ID	XMLSchema
Dictionary ID	http://opcfoundation.org/ComplexData/Sample2.xsd
Type ID	Connection/Config

The Dictionary ID and Type ID properties would also reference different items in the Complex Data Namespace (i.e. in /CPX/XMLSchema/Sample2 branch).

This example illustrates a situation where the XML schema exposed by a DA server may be different than the XML schema that describes the underlying data. The decision to use a different schema is made by the designers of the DA server who are best able to evaluate the needs of the DA clients any potential performance issues.

5.2 Function Block Example

The figure is an example of a Function Block complex data type:



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

In this example, the Block Tag is exposed by the server as a DA item. Therefore, it is accessible through the Function Block Header complex item, but also as a DA item of its own. The Number of Parameters can also be accessed both as part of the Function Block Header complex item, and separately as a property of that complex item. Finally the Execution Time and Execution Frequency can only be accessed as part of the Function Block Header complex item

Server Example

This section provides an example for the Block Header complex data item illustrated above. Its abstract definition is repeated below, followed by an example of how it would be represented as a complex data item. In this example, the Block Header is defined by the hypothetical "Fieldbus" type system.

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

The table below shows the complex data item properties for the Function Block Header when described by the "Fieldbus" type system.

Property Name	Property Value	Comment
Type System ID	FieldbusOD	The name of the type system that is used to describe the complex data.
Dictionary ID	TempTransmitter102 2.01.1234	This identifies the device and firmware version that is the source of the data exposed by this item.
Type ID	"122"	This is the index of the structure definition within the device's object dictionary.

The Dictionary ID and Type ID properties have associated items.

The item for the Dictionary ID would have the following properties:

Property Name	Property Value	Comment
Value	TempTransmitter102 2.01.1234	This identifies the device and firmware version that is the source of the data exposed by this item.
Dictionary		This could conatin the entire Fieldbus object dictionary for the device, however, performance concerns may make it impractical to fetch from the device.

The item for the Type ID would have the following properties:

Property Name	Property Value	Comment
Value	"122"	This is the index of the structure definition within the device's object dictionary.
Type Description	04 01 20 07 04 07 04 07 02	In this example, the "Fieldbus Consortium" binary type description is show in hexadecimal, with the following meaning: 04 = the number of elements in the structure 01 = string element 20 = the length of the string is 32 bytes 07 = Unsigned Integer element 04 = the length of the Unsigned Integer is 4 bytes 07 = Unsigned Integer element 04 = the length of the Unsigned Integer is 4 bytes 07 = Unsigned Integer element 02 = the length of the Unsigned Integer is 2 bytes

The Function Block Header could also be described by the "OPC Binary" type system. The table below shows the complex data item properties for the Function Block Header:

Property Name	Property Value	Comment
Type System ID	OPCBinary	The name of the type system that is used to describe the complex data.
Dictionary ID	http://opcfoundation.org/ComplexData/Sample3/	The URI for the Type Dictionary.
Type ID	FunctionBlockHeader	"FunctionBlockHeader" is the TypeID for the Type Description within the OPC Binary Type Dictionary.

The Dictionary ID and Type ID would have different associated item ids.

The Dictionary property of the Dictionary ID item would now be:

```
<?xml version="1.0" encoding="utf-8" ?>
<TypeDictionary
  xmlns="http://opcfoundation.org/OPCBinary/1.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  DefaultBigEndian="true"
>
  <TypeDescription TypeID="FunctionBlockHeader"
    <CharString Name="Block Tag" xsi:type="Ascii" Length="32" />
    <Integer Name="Execution Time" xsi:type="Int32" />
    <Integer Name="Execution Frequency" xsi:type="Int32" />
    <Integer Name="Number of Parameters" xsi:type="Int16" />
  </TypeDescription>
</TypeDictionary>
```

The Type Description property of the Type ID item would now be:

```
<?xml version="1.0" encoding="utf-8" ?>
<TypeDescription
  xmlns="http://opcfoundation.org/OPCBinary/1.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  TypeID="FunctionBlockHeader"
>
  <CharString Name="Block Tag" xsi:type="Ascii" Length="32" />
  <Integer Name="Execution Time" xsi:type="Int32" />
  <Integer Name="Execution Frequency" xsi:type="Int32" />
  <Integer Name="Number of Parameters" xsi:type="Int16" />
</TypeDescription>
```

Client Example

Clients typically browse the items provided by a server to locate an item of interest. When an item is located that the client wishes to access, the client can determine which properties the item supports by calling QueryAvailableProperties.

If the Complex Data Item Properties are supported, then the client uses them to be able to interpret the value of complex data item. It first examines the "Type System" Item Property to determine which type system the server is using to describe the data. If the client is capable of using the type system, it identifies the structure of the data using the "Dictionary ID" and the "Type ID" Item Properties.

The client can then determine whether it has cached the definition of the specified version of this complex type and is prepared to interpret the value. If not, it can access the type definition contained in the "Dictionary" and/or "Type Description" Item Property. This property describes how the server has encoded the data value and provides adequate information for the client to be able to interpret it.

Note that if two separate complex data items have the same "Dictionary ID", then the server must provide the same "Dictionary" for each. However, servers generally will maintain a single copy of a "Dictionary" and provide access to it through the Dictionary Item Properties of the complex data items that use it.

After acquiring the type description, the client uses it to interpret values from one or more complex data items that all share the same dictionary.

6 OPC Binary Type System

6.1 Concepts

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

Figure 7 below illustrates the structure of the dictionary.

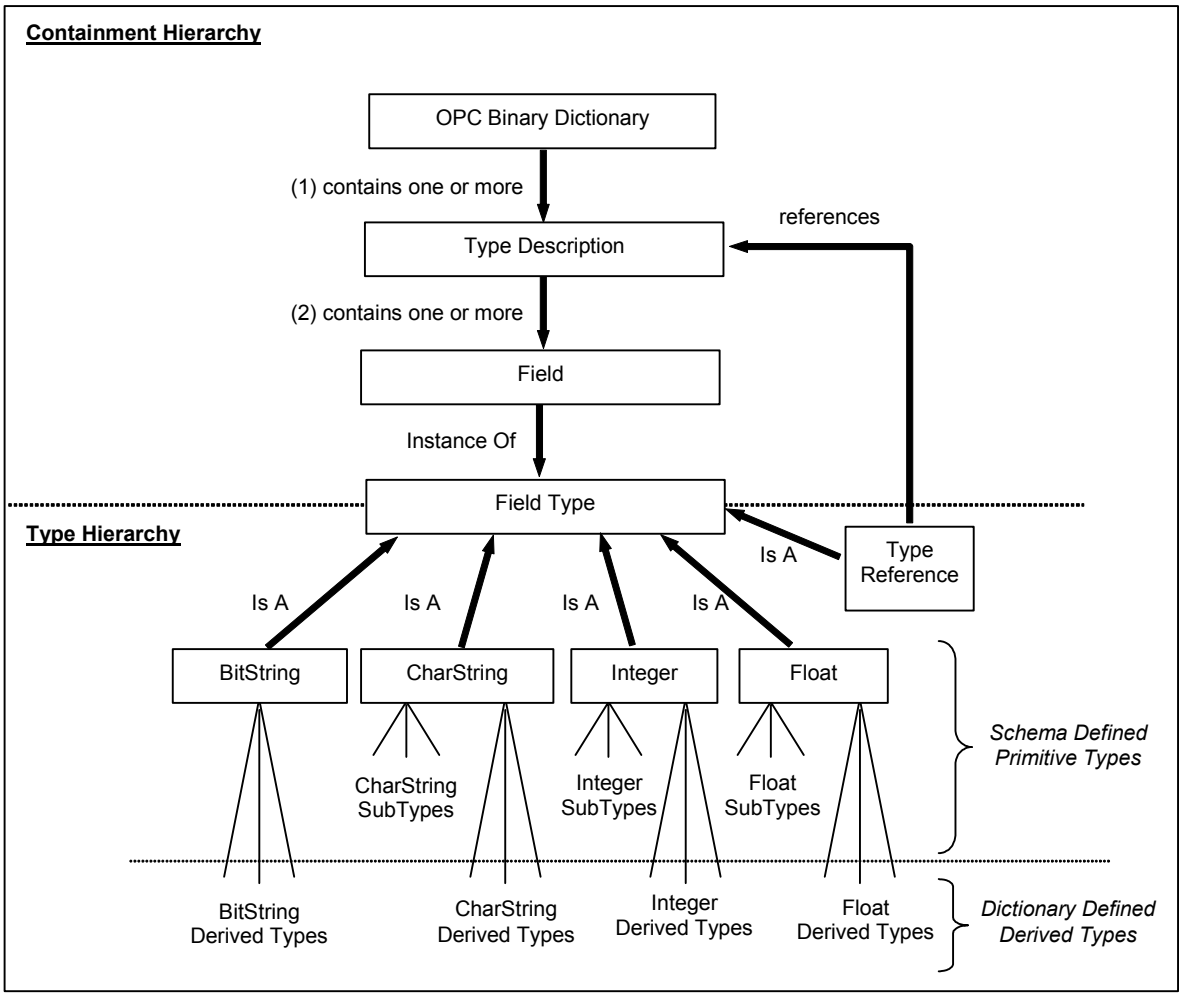


Figure 7 – OPC Binary Dictionary Structure

6.1.1 Dictionary Composition

The Containment Hierarchy portion of Figure 7 above shows two containment relationships. The first illustrates that OPC Binary Dictionaries are composed of a set of Type Descriptions. This relationship does not imply that the dictionary is a data structure.

These Type Descriptions are used to define constructed types or primitive types. Each is described below.

6.1.2 Defining Constructed Types

The second containment relationship in the Containment Hierarchy portion of Figure 7 shows that constructed types, defined as TypeDescriptions (see Section 6.2.2), are composed of a set of fields. Fields are defined as instance of a FieldType, defined in Section 6.2.3.

Fields may be constructed or they may be primitive. Field definitions may reference other constructed type definitions, or they may be either “primitive” or “derived”. Primitive Types are defined in this specification in Section 6.2.4.2 while Derived Types are defined by the containing dictionary (see Section 6.1.3).

6.1.3 Defining Derived Types

The Type Hierarchy portion of the figure shows the derivation of Primitive and Derived Types. Primitive Types are extensions (sub-types) of the Field Type in the OPC Binary XML Schema. Derived Types, on the other hand, are defined in a dictionary as XML instances of one of the Primitive Types. XML instances are defined by providing values for attributes.

The remainder of this section describes the XML Schema for the OPC Binary Dictionary.

6.2 Schema Description

6.2.1 OPC Binary Dictionary

The TypeDictionary element is the root element of an OPC Binary dictionary. The XML Schema definition for the TypeDictionary is shown below.

```
<xs:element name="TypeDictionary">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TypeDescription" type="TypeDescription"
                  minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="DefaultBigEndian" type="xs:boolean" default="true" />
    <xs:attribute name="DefaultStringEncoding" type="xs:string" default="UCS-2" />
    <xs:attribute name="DefaultCharWidth" type="xs:unsignedInt" default="2" />
    <xs:attribute name="DefaultFloatFormat" type="xs:string" default="IEEE-754" />
  </xs:complexType>
</xs:element>
```

The TypeDictionary contains a sequence of TypeDescriptions and has the following attributes:

Attribute	Description
DefaultBigEndian	This attribute defines the default byte order for all integers and all multibyte characters defined in the dictionary. Individual Type Descriptions may override this default. When the value of this attribute is “true”, Big Endian byte order (MSB first) is used. When “false”, Little Endian (LSB) byte order is used.
DefaultStringEncoding	This attribute defines the default encoding used for all character strings defined in the dictionary. Individual Type Descriptions may override this default. The standard encodings defined for this specification are described following this table.
DefaultCharWidth	This attribute defines the default character width (in bytes) used for all character strings contained in the dictionary. Individual Type Descriptions may override this default. A DA client may not recognize all string encodings. In these cases, the DA client must treat a character as an unsigned integer with a size equal to the character size for the encoding. For this reason, the character width must always be specified in addition to the encoding even though character width is determined by the encoding.
DefaultFloatFormat	This attribute defines the default format used for all floating point numbers defined in the dictionary. Individual Type Descriptions may override this default. See Section 6.2.4.2.4 below for the definiition of the Float Format attribute.

This specification defines a set of character encodings, although others may be used. The encoding determines the mapping between a lexical character and a sequence of bits.

Depending on the encoding, the length of each character may be fixed, or variable. The ASCII encoding uses 7 bits per character, and the UCS-2 and UTF-16 encodings use 2 bytes per character. The UTF-8 and UTF-16 encodings allow for multiple bytes per lexical character. The following table lists the well-known encodings defined in this specification:

Encoding	Description
ASCII	A 7-bit character encoding stored in an 8-bit character.
UCS-2	The un-encoded 16-bit Unicode character set.
UCS-4	The un-encoded 32-bit Unicode character set.
UTF-8	The Unicode character set encoded with a nominal 8 bit character size.
UTF-16	The Unicode character set encoded with a nominal 16 bit character size.

6.2.2 Type Description

The Type Description is used to define a constructed type. The XML Schema definition for the TypeDescription is shown below.

```
<xs:complexType name="TypeDescription">
  <xs:sequence>
    <xs:element name="Field" type="FieldType"
                minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="TypeID" type="xs:string" use="required" />
  <xs:attribute name="DefaultBigEndian" type="xs:boolean" use="optional" />
  <xs:attribute name="DefaultStringEncoding" type="xs:string" use="optional" />
  <xs:attribute name="DefaultCharWidth" type="xs:unsignedInt" use="optional" />
  <xs:attribute name="DefaultFloatFormat" type="xs:string" use="optional" />
</xs:complexType>
```

Constructed type definitions contain one or more fields. The type of a field may be a contructed type, a primitive type, or a derived type. When a TypeDescription contains a field that is constructed, the type of the field is a reference to another TypeDescription, and that TypeDescription is said to be “nested”.!!

Each Type Description has the following attributes:

Attribute	Description
TypeID	This attribute defines the Type ID (see Section 3.3.3) for this Type Description.
DefaultBigEndian	This attribute defines the default byte order for all integers and all multibyte characters contained by this Type Description, including those contained in nested Type Descriptions[1]. When the value of this attribute is “true”, Big Endian byte order (MSB first) is used. When “false”, Little Endian (LSB) byte order is used.
DefaultStringEncoding	This attribute defines the default encoding used for all character strings contained by this Type Description, including those contained in nested Type Descriptions[1]. See Section 6.2.1 above for the encodings defined by this specifications.!!
DefaultCharWidth	This attribute defines the default character width (in bytes) used for all character strings contained by this Type Description, including those contained in nested Type Descriptions[1]. A DA client may not recognize all string encodings. In these cases, the DA client must treat a character as an unsigned integer with a size equal to the character size for the encoding. For this reason, the character width must always be specified in addition to the encoding even though character width is determined by the encoding.
DefaultFloatFormat	This attribute defines the default format used for all floating point numbers contained by this Type Description, including those contained in nested Type Descriptions[1]. Only the ‘IEEE-754’ format is defined in this specification.
NOTES: 1 – The default attributes for Type Descriptions contained by this Type Description override the default values specified for this Type Description. Therefore, if a field has an attribute value, such as its byte order, that is different than that defined for the Type Description that contains it, then a constructed type that explicitly overrides the byte order should be defined and referenced by the field. In addition, if the default attribute value is not specified for this Type Description, then the value of this attribute is inherited from the containing structure (either the Dictionary or a containing Type Description). If this attribute value is not specified for a Type Description, and the Type Description is used in more than one containing structure, then the value is inherited independently for each containment.	

6.2.3 Field Type

Field Types are used to define the data types used for fields of constructed types. The XML Schema definition for FieldType is shown below.

```
<xs:complexType name="FieldType">
  <xs:attribute name="Name" type="xs:string" use="optional" />
  <xs:attribute name="Format" type="xs:string" use="optional" />
  <xs:attribute name="Length" type="xs:unsignedInt" use="optional" />
  <xs:attribute name="ElementCount" type="xs:unsignedInt" use="optional" />
  <xs:attribute name="ElementCountRef" type="xs:string" use="optional" />
  <xs:attribute name="FieldTerminator" type="xs:string" use="optional" />
</xs:complexType>
```

Standard Field Types are defined in the next section as XML Schema extensions of FieldType (sub-types of Field Type).

Each FieldType has the following attributes:

Attribute	Description
Name	Used only to define the name of a field. It is not required, but when present, it must be unique within the containing Type Description.
Format	The format provides additional semantic information about the field. The format is not required to parse complex data types, however, DA clients that wish to manipulate the value of the field will likely need to understand the format. This specification defines the following standard format strings. All others are specific to the containing dictionary and may not be interpretable by all DA clients. FILETIME — A WIN32 FILETIME value which is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601. time_t — An ANSI C time value which is a 32-bit value representing the number of seconds elapsed since midnight (00:00:00), January 1, 1970 HexString — A sequence of bytes encoded as hexadecimal digits in a string. Each byte is represented as two hexadecimal digits. Each byte may be separated by whitespace characters. For example: “9D A3” is the hex string encoding for the Big Endian Unsigned16 value = 40355. OnesComplement — This format is used only for bitstring to provide for the support of one’s complement encodings.
Length	The number of bits in a bitstring, or the length in bytes for all other types.
ElementCount	Used to define fields that are fixed length arrays of the referenced type. This attribute defines the number of elements in the array, and may not used when ElementCountRef, or FieldTerminator is present.
ElementCountRef	Used only to define fields that are variable length arrays. This attriibute contains the name of preceding field in the same Type Description. The referenced field must be an integer type that contains the number of elements in the array. This attribute may not used when ElementCount, or FieldTerminator is present.
FieldTerminator	This is a valid instance of the Field Type that indicates the end of the field. This attribute is used to describe fields containing an arbitrary sequence of values. This attribute may not used when ElementCount, or ElementCountRef is present. The value for the field terminator must be a sequence of bytes represented as a string formatted with the XML ‘hexBinary’ notation. This sequence of bytes must exactly match the sequence of bytes that would appear in the binary data buffer. This implies that the terminator must take into account the value of the DefaultBigEndian attribute for the containing Type Description (or the referenced Type Description in the case of Type Reference fields). An OPC Binary schema should always explicitly specify a value for the DefaultBigEndian for all Type Decriptions that either contain fields that use a field terminator or are referenced by fields that use a field terminator. Examples: Field Data Type — Terminator — Byte Order — Hexidecimal String ASCII string: — tab character — not applicable — 09 Unicode string: — tab character — Big Endian — 0009 Unicode string: — tab character — Little Endian — 0900 Int16 — 1 — Big Endian — 0001 Int16 — 1 — Little Endian — 0100

6.2.4 Standard Field Types

This specification defines the following standard field types.

Standard Field Type	Description
TypeReference	A reference to another type defined within the dictionary.
Primitive Type	
BitString	One or more consecutive binary digits. Used to define bitstrings and bitfields.
CharString	An ordered sequence of text characters with a specific encoding.
Ascii	A character string with ASCII encoding as defined in Section 6.2.1
Unicode	A character string with UCS-2 encoding as defined in Section 6.2.1
Integer	A decimal whole number.
Int8	A signed 8-bit integer.
Int16	A signed 16-bit integer.
Int32	A signed 32-bit integer.
Int64	A signed 64-bit integer.
UInt8	An unsigned 8-bit integer.
UInt16	An unsigned 16-bit integer.
UInt32	An unsigned 32-bit integer.
UInt32	An unsigned 64-bit integer.
FloatingPoint	A decimal number with a limited precision fractional component..
Single	A 32-bit fFloating point number
Double	A 64-bit floating point number

The first standard field type in the table is the TypeReference. The TypeReference allows the type of a field to be defined as another Type Description. This allows for nesting of constructed types.

All other standard Field Types are referred to as “Primitive Types”. Dictionaries may extend the list of primitive types by deriving their own from them. Derived types are defined in the dictionary’s XML by defining instances of Primitive Types.

6.2.4.1 Type Reference

Type References are used only to define fields. The XML Schema definition for TypeReference is shown below.

```
<xs:complexType name="TypeReference">
  <xs:complexContent>
    <xs:extension base="FieldType">
      <xs:attribute name="TypeID" type="xs:string" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The Type Reference of a field identifies the Type Description that defines the data type for the field. The Type Reference is the TypeID of the referenced Type Description.

This XML Schema component description contains one attribute:

Attribute	Description
TypeID	The TypeID of another type description that provides the type for this field.

6.2.4.2 Primitive Data Types

Primitive data types are defined for fields with unstructured, or *primitive*, values. A field containing a primitive value that cannot be fully described by one of the primitive types must be described by either deriving a dictionary specific primitive type, or by defining the field in a Type Description to be a primitive type, and by further specifying the appropriate attriibute values for it. In this case, only the Format, ElementCount, and Terminator Field Type Attributes are used (see Section 6.2.3).

For example, an unsigned 128-bit integer can be derived from the Integer primitive type by specifying values for the “Length” and “Signed” attributes, while a null terminated string type can be derived by specifying the Length, CharWidth, Encoding, and Terminator attributes. Alternatively, the field defintion in the TypeDescription can supply these attribute values without having to specifically define a new derived type.

6.2.4.2.1 Bit String Type

This primitive type is used to define bit strings. The XML Schema definition for BitString is shown below.

```
<xs:complexType name="BitString">
  <xs:complexContent>
    <xs:extension base="FieldType"></xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A bit string is a sequence of bits. A bit string may have any length. The start and end of an individual bit string does not have to align with a byte boundary, however, all non-bit string fields must begin on a byte boundary. In addition, the total length of all fields in a Type Description must be an integer multiple of 8 bits.

Therefore, the DA client must always assume unused bit padding exists if the end of the a bit string field is not on a byte boundary and one of the following conditions is true:

- The next field is not a bit string;
- The current bit string field is the last field in a Type Description;

This type defines no BitString specific attributes:

6.2.4.2.2 Char String Type

This primitive type is used to define character strings. The XML Schema definition for CharString is shown below.

```
<xs:complexType name="CharString">
  <xs:complexContent>
    <xs:extension base="FieldType">
      <xs:attribute name="CharWidth" type="xs:unsignedInt" />
      <xs:attribute name="StringEncoding" type="xs:string" />
      <xs:attribute name="CharCountRef" type="xs:string" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A character string is a sequence of text characters with a specific encoding. The size of a text character is defined using the CharWidth attribute.

A DA client may not recognize all string encodings. In these cases, the DA client must treat a character as an unsigned integer with a size equal to the character size for the encoding. For this reason, the character width must always be specified in addition to the encoding even though character width is determined by the encoding. The null terminator for all encodings is always a zero value with a size equal to the character size for the encoding.

This type has the following attributes:

Attrbute	Description
CharWidth	See DefaultCharWidth in Section 6.2.1
StringEncoding	See DefaultStringEncoding in Section 6.2.1
CharCountRef	The name of preceding field within the containing Type Description that contains the length, in characters, of the CharString field. This attribute may not be used in conjunction with the ElementCount, or ElementCountRef attributes. Null terminators in the string should be ignored if this attrrbute exists.

This specification defines the following CharString sub-types:

Type	CharWidth	Encoding
Ascii	1	“ASCII”
Unicode	2	“UCS-2”

6.2.4.2.3 Integer

This primitive type is used to define an integer or an integer array. The XML Schema definition for Integer is shown below.

```
<xs:complexType name="Integer">
  <xs:complexContent>
    <xs:extension base="FieldType">
      <xs:attribute name="Signed" type="xs:boolean" default="true" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

An integer is a whole decimal number.

This type has the following attribute:

Attribute	Description
Signed	Indicates, when “true”, that the integer is a two’s complement signed integer, and when “false” that the integer is unsigned. Dictionaries may define one’s complement encoding by defining a bitstring with the Base Type Attribute Format=”OnesComplement”.

This specification defines the following Integer sub-types:

Type	Length	Signed
Int8	1	true
UInt8	1	false
Int16	2	true
Uint16	2	false
Int32	4	true
Uint32	4	false
Int64	8	true
Uint64	8	false

6.2.4.2.4 Floating Point

This primitive type is used to define a floating point number, or an array of floating point numbers. The XML Schema definition for FloatingPoint is shown below.

```
<xs:complexType name="FloatingPoint">
  <xs:complexContent>
    <xs:extension base="FieldType">
      <xs:attribute name="FloatFormat" type="xs:string" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A floating point number is a limited precision representation of a decimal number. The floating point representation is specified by the Format attribute. If the Format attribute is not specified for a field then the value is determined from the Default Float Format attribute specified at the Type Description or Type Dictionary level. Only the 'IEEE-754' format is defined in this specification; a DA client that does not recognize the Format must treat the Floating Point as a sequence of bytes.

This XML Schema component description contains two attributes:

Attribute	Description
FloatFormat	See DefaulFloatFormat in Section 6.2.1

This specification defines the following FloatingPoint sub-types:

Type	Length	DefaultFloatFormat
Float	4	"IEEE-754"
Double	8	"IEEE-754"

6.3 Type Description Examples

1) Definition of a bitfield structure that includes explicit padding.

```
<TypeDescription TypeID="BitFields">
  <BitString Name="Data" Length="22" />
  <BitString Name="Padding" Length="2" />
</TypeDescription>
```

2) Definition of a 128-bit signed integer.

```
<TypeDescription TypeID="Int128">
  <Integer Length="16" Signed="true" />
</TypeDescription>
```

3) Divides a 16 bit word into several bit fields.

```
<TypeDescription TypeID="Quality">
  <BitString Name="LimitBits" Length="2" />
  <BitString Name="QualityBits" Length="6" />
  <BitString Name="VendorBits" Length="8" />
</TypeDescription>
```

4) Illustrates a type with several different field types.

```
<TypeDescription TypeID="ValueQT">
  <Integer Name="vt" xsi:type="Int16" />
  <Integer Name="reserved" xsi:type="Int16" />
  <BitString Name="value" Length="96" />
  <TypeReference Name="quality" TypeID="Quality" />
  <Integer Name="timestamp" xsi:type="Int64" Format="FILETIME" />
</TypeDescription>
```

5) Illustrates a structure with a fixed length character string field.

```
<TypeDescription TypeID="BlockHeader" DefaultBigEndian = "true">
  <CharString Name="Block Tag" xsi:type="Ansi" Length="32" />
  <Integer Name="Execution Time" xsi:type="Int32" />
  <Integer Name="Execution Frequency" xsi:type="Int32" />
  <Integer Name="Number of Parameters" xsi:type="Int16" />
</TypeDescription>
```

6) Illustrates arrays with a length fixed in the type description.

```
<TypeDescription TypeID="FixedLengthArrays">
  <BitString Name="Bits" Length="96" ElementCount="10" />
  <Integer Name="Shorts" xsi:type="Int16" ElementCount="10" />
  <CharString Name="Strings" CharWidth="2" Encoding="UCS-2" ElementCount="10" />
</TypeDescription>
```

7) Illustrates two variable length arrays with defined terminators (-1 and “9999” respectively).

```
<TypeDescription TypeID="TerminatedGroups" DefaultBigEndian="false"  >
  <Integer Name="Ints" xsi:type="Int32" FieldTerminator="FFFFFFFF" />
  <CharString Name="Dates" xsi:type="Unicode" FieldTerminator="3900390039003900" />
</TypeDescription>
```

8) Illustrates a variable length array with a length that must be determined at runtime.

```
<TypeDescription TypeID="FunctionBlockGroup">
  <Integer Name="Count" xsi:type="UInt32" />
  <TypeReference Name="Headers" Ref="BlockHeader" ElementCountRef="Count" />
</TypeDescription>
```

7 Type Conversions

7.1 Purpose

DA servers that support complex data usually expose this data in its native format. However, this format may not be useful to all DA clients. For example, a DA server may wish to support clients dedicated to either process control or configuration and maintenance. The process control clients likely understand the native format and wish to use that format for efficiency. On the other hand, the configuration clients usually must interface with many different DA servers and would prefer to access this data in more general format such as XML. The DA server can support the needs of both clients by representing the complex data item in its native format and in one or more alternate formats using the mechanisms described in this section.

For this reason, this specification defines type conversion mechanism for complex data items. This mechanism allows DA clients to browse the set of complex type conversions that are available for an individual DA complex data item and to them read from or write to the DA item in any one of the available formats.

7.2 Representation

Servers that support type conversions would represent the set of available conversions for a specific complex data item by creating a branch named ‘CPX’ under the complex data item within the DA server’s address namespace. This branch would contains one or more items that would expose the complex value same underlying datasource in different formats. This structure is shown in Figure 8.

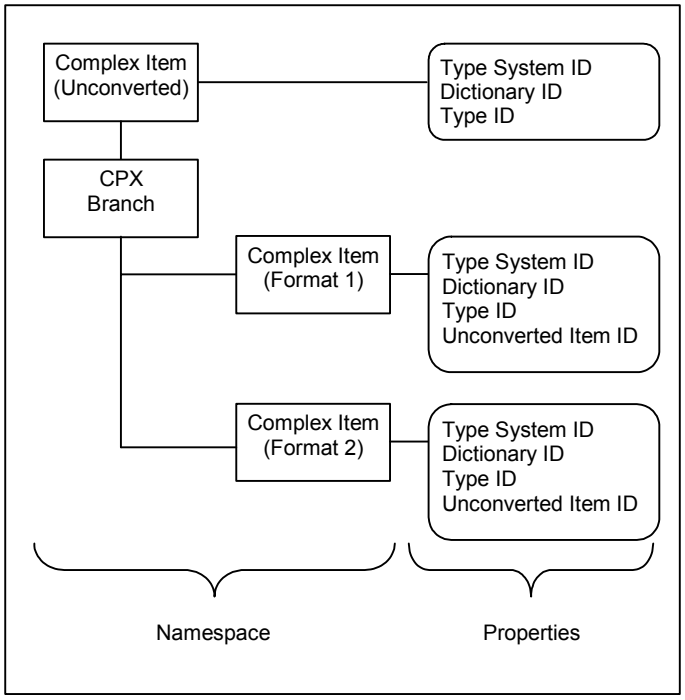


Figure 8 – Representing a Complex Data Type Conversion.

A DA client would be able to determine the set of available type conversions by browsing the items in the ‘CPX’ branch. A DA client would then be able to read and write data in alternate formats by accessing the items contained in this CPX branch instead of the base native complex data item. These

alternate items would also support all the complex data properties that are necessary for the DA client to read the Type Description information for the alternate formats.

DA servers that support complex type conversions should choose names for alternate items that are descriptive enough to allow a DA client to present the names in a list to the end user. For example, an item that exposes a binary data source as an XML document could be named 'XML' (assuming only one XML format is supported for the particular item). In addition, if there is more than one instance of a complex type in the server namespace, then the names of its conversions items should be the same for all instances of the same complex type server address space. Note that the next section reserves the name 'DataFilters' which is an item that also resides in the 'CPX' branch. DA clients must always ignore this item when browsing for available type conversions for a complex data item.

For an example of a complex data type conversion consider the binary data structure described in its native format with the following 'OPC Binary' type description:

```
<TypeDictionary
  xmlns="http://opcfoundation.org/OPCBinary/1.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  DefaultBigEndian="false"
>
  <TypeDescription ID="ConnectionStatus">
      <Integer Name="ConnectState" Format="BOOL" xsi:type="Int32" />
      <Integer Name="LastConnectTime" Format="FILETIME" xsi:type="UInt32" />
      <Integer Name="ConnectFailCount" xsi:type="UInt32" />
      <Integer Name="IsConnected" Format="BOOL" xsi:type="Int32" />
  </TypeDescription>
</TypeDictionary>
```

The complex data properties for this item would be:

Property	Value
Item ID	/Sample/Connections/Device00
Access Rights	readWritable
Type System ID	OPCBinary
Dictionary ID	http://opcfoundation.org/ComplexData/Sample2/Native.xml
Type ID	ConnectionStatus

The following XML schema describes the same complex data item:

```
<xsd:schema
  xmlns=http://opcfoundation.org/ComplexData/Sample2/XML.xsd
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  targetNamespace=http://opcfoundation.org/ComplexData/Sample2/XML.xsd
  elementFormDefault="qualified"
>
<xs:schema>
  <xs:complexType name="ConnectionStatus">
    <xs:sequence>
      <xs:element name="ConnectState" type="xs:boolean" />
      <xs:element name="LastConnectTime" type="xs:dateTime" />
      <xs:element name="ConnectFailCount" type="xs:unsignedInt" />
      <xs:element name="IsConnected" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

An item that provides access to the same data in the XML format would have the following properties:

Property	Value
Item ID	/Sample/Connections/Device00/CPX/XML
Access Rights	readOnly
Type System ID	XMLSchema
Dictionary ID	http://opcfoundation.org/ComplexData/Sample2/XML.xsd
Type ID	ConnectionStatus
Unconverted Item ID	/Sample/Connections/Device00

The ‘Unconverted Item ID’ property is necessary because it tells DA clients that an individualthis complex data item is part of a group of items that provide access to a single underlying data sourcis a type conversion item for the specified native iteme. The value of this property in XML-DA is the Item Name. This specification requires that all items in the ‘CPX’ branch under a complex data item must have the same Item Path.

It is up to the server to decide whether it will support writes to the item in the alternate formats.

In this example the DA server supports an Item ID syntax where the branch and leaf name are part of the Item ID. However, DA servers are allowed to use any appropriate syntax for the actual Item IDs as long as the names that appear in the DA server address space follow the conventions defined above.

8 Data Filters and Queries

8.1 Purpose

Servers that support complex data items may also wish to support more sophisticated queries and/or filters that can control what the the DA server returns to the client. Applications for filters or queries range from a simple mask where the client tells the server to ignore some fields when testing whether a data change update is requiredhas occurred to complex queries that the server uses to extract the data from an underlying data source such as a database.

For this reason, this specification defines a general filter/query specification mechanism for complex data items. This mechanism allows DA clients to construct complex queries using a syntax defined by the server implementation and to tell the server to use these queries each time the client reads data from the server.

It is expected that most server implementations of data filters and queries will fall into the following three categories:

- **Filters -** the client specifies a filter that excludes one or more fields from the value of complex data item. This capability is most useful for data representations, such as XML, that allow missing elements in a data value.
- **Masks** – the client specifies a mask that specifies which fields it is interested in. The server uses this mask to determine which fields to check for data changes. The actual complex data value returned to the client remains the same.
- **Queries** – the client specifies a query that is used to fetch data from an underlying data source. The DA server caches this query and uses it to extract data whenever it processes a read request.

8.2 Representation

Unlike the Type Conversions described in the previous section, Data Filters require the DA server to use information provided by the DA client. In addition, different clients may wish to establish different filters or queries for the same item. The DA server supports this behavior by creating a write only branch/item labeled ‘DataFilters’ in a branch labeled ‘CPX’ under the complex data item within the DA server’s address space. The same ‘DataFilters’ branch may also exist under any Type Conversion Items that are available. Figure 9 illustrates the complex data item hierarchy with data filter items.

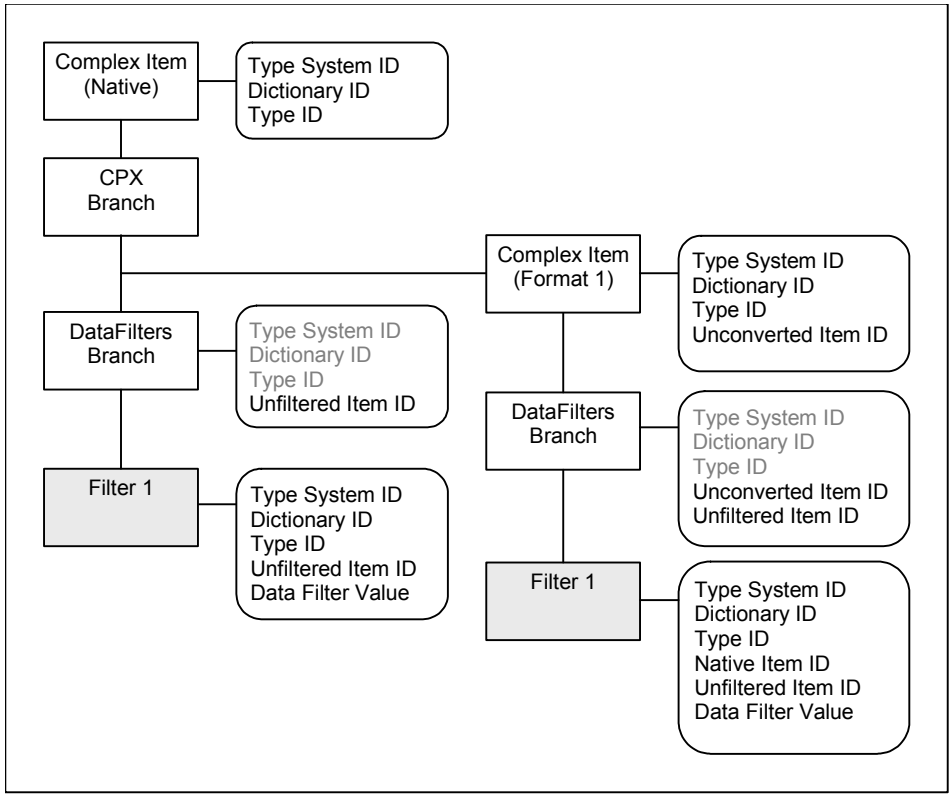


Figure 9 – Representing a Complex Data Filter

The complex data item properties on the Data Filters branch are used to describe the syntax of the filter/query value. These properties are optional because the syntax of the filter/query for a specific server may not have a formal type description or simply make use of some well known query language such as XPath or SQL.

A client that wishes to specify a filter/query must write the filter/query parameters to the Data Filters branch/item. The server validates the filter/query parameters and creates a new item under the Data Filters branch. The name of the new DataFilter item is one the of parameters provided by the client. The server should return 'E_FILTER_DUPLICATE' if an item with the same name already exists under the Data Filters branch.

After creating the new Data Filter item, and the DA client must read from or subscribe to the new Data Filter item int order to receive values of the the complex data item with the filter or query applied. The client must be able to find all existing Data Filter items by browsing the Data Filters branch. The filter value for each item is stored in the 'DataFilter' property. DA servers are not required to make query/filter items created by one client visible to other clients, however, they may choose to do so.

The filter parameters are passed as an XML document with the following schema:

```
<xsd:schema
  xmlns="http://opcfoundation.org/ComplexData/DataFilters/1.0/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://opcfoundation.org/ComplexData/DataFilters/1.0/"
  elementFormDefault="qualified"
>
<xs:schema>
  <xs:element name="DataFilter">
    <xs:complexType>
      <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The syntax of the filter/query value is defined by the server and my be described with the complex data properties associated with the Data Filters item. The filter/query value is always encoded as the text body of the DataFilters element in the parameters document even if the server uses XML for its filter format. The server should return 'E_FILTER_INVALID' if the syntax of the filter value is not valid.

A client may update the filter by writing only the filetr/query value to the Data Filter item as a string (In other words, the XML document described above is only used to create new data filter items). A client may delete a Data Filter item by writing an empty string (""). The DA server must remove the Data Filter item from its address space after a client writes a "" for a filter value. Any existing groups/subscriptions that contain the item should receive 'E_UNKNOWNITEMID' when the data filter item is removed. A DA server may prevent clients from removing a Data Filter but returning 'E_FILTER_INVALID' when a client attempts to write an empty filter value.

If the filter/query value is valid but results in an empty data set (the meaning of which is determined by the server). The server should return 'S_FILTER_NO_DATA' with a empty value and an appropriate quality and timestamp.

If the server encounters an error when applying the filter or issuing the query to the underlying data source then the server should return 'E_FILTER_ERROR'.

As mentioned above, the exact semanatics and syntax of the Data Filter property depend on the DA server implementation. That saidHowever, the XPath specification (http://www.w3.org/TR/xpath) provides a standard way to filter XML based data.

Consider the following complex data item value represented in XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ConnectionStatus xmlns="http://opcfoundation.org/ComplexData/Sample1.xsd">
  <ConnectState>true</ConnectState>
  <LastConnectTime>2003-05-27T06:16:31.6718750-07:00</LastConnectTime>
  <ConnectFailCount>1</ConnectFailCount>
  <IsConnected>true</IsConnected>>
</ConnectionStatus>
```

The complex data properties for this item would be:

Property	Value
Item ID	/Sample/Connections/Device00/CPX/XML
Access Rights	readOnly
Type System ID	XMLSchema
Dictionary ID	http://opcfoundation.org/ComplexData/Sample2/XML.xsd
Type ID	ConnectionStatus
Unconverted Item ID	/Sample/Connections/Device00

A DA client could specify the following XPath for the data filter.

/*/ConnectFailCount

Which would request that the server only return the following.

```
<?xml version="1.0" encoding="utf-8" ?>
<ConnectFailCount xmlns="http://opcfoundation.org/ComplexData/Sample1.xsd">
10
</ConnectFailCount>
```

The DA server would define the Data Filters item with these properties:

Property	Value
Item ID	/Sample/Connections/Device00/CPX/XML/DataFilters
Access Rights	writeOnly
Unconverted Item ID	/Sample/Connections/Device00
Unfiltered Item ID	/Sample/Connections/Device00/CPX/XML

The ‘Unfiltered Item ID’ property is necessary because it tells DA clients that an individualthis complex data item is a filter for the specified item. part of a group of items that provide access to a single underlying data source. The value of this property in XML-DA is the Item Name. This specification requires that all items in the ‘CPX’ branch under a complex data item must have the same Item Path.

The DA client would create a new filter item by writing this value to the Data Filters item.

```
<?xml version="1.0" encoding="utf-8" ?>
<DataFilter Name="Filter01">
  /*/ConnectFailCount
</ DataFilter >
```

The server would then create a new item with these properties:

Property	Value
Item ID	/Sample/Connections/Device00/CPX/XML/DataFilters/Filter01
Access Rights	readWritable
Type System ID	XMLSchema
Dictionary ID	http://opcfoundation.org/ComplexData/Sample2/XML.xsd
Type ID	ConnectionStatus
Unconverted Item ID	/Sample/Connections/Device00
Unfiltered Item ID	/Sample/Connections/Device00/CPX/XML
Data Fiter Value	/*/ConnectFailCount

The complex type description for the data filter item should be the same since the filter data values must be valid instances of the complex data item or, in the case of type description systems that support missing data, valid sub-sets of the complex data item.

The ‘Unconverted Item ID’ is only present when a data filter is applied to an item which is a type conversion of a complex data item. This property allows the DA client to recognize the data filter item a member of a group of items that expose the same underlying data source. The ‘Unfiltered Item ID’ refers to the item in the format that is actually being manipulated by the data filter.

9 Error Codes

This specification defines a number of complex data specific error codes.

For COM-DA these codes are added to the existing set of Data Access error codes declared in ‘opcerror.h’ but have the prefix ‘OPCCPX_’.

For XML-DA the error codes have the namespace “http://opcfoundation.org/ComplexData/1.0/”.

The codes are listed in the following table:

Name	Description
E_TYPE_CHANGED	The dictionary and/or type description for the item has changed.
E_FILTER_DUPLICATE	A data filter item with the specified name already exists.
E_FILTER_INVALID	The data filter value does not conform to the server's syntax.
E_FILTER_ERROR	An error occurred when the filter value was applied to the source data.
S_FILTER_NO_DATA	The item value is empty because the data filter has excluded all fields.

Appendix A. OPC Binary Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  id="OPCBinary"
  targetNamespace="http://opcfoundation.org/OPCBinary/1.0/"
  elementFormDefault="qualified"
  xmlns="http://opcfoundation.org/OPCBinary/1.0/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="TypeDictionary">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="TypeDescription" type="TypeDescription"
                    minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="DefaultBigEndian" type="xs:boolean" default="true" />
      <xs:attribute name="DefaultStringEncoding" type="xs:string" default="UCS-2" />
      <xs:attribute name="DefaultCharWidth" type="xs:unsignedInt" default="2" />
      <xs:attribute name="DefaultFloatFormat" type="xs:string" default="IEEE-754" />
    </xs:complexType>
  </xs:element>
  <xs:complexType name="TypeDescription">
    <xs:sequence>
      <xs:element name="Field" type="FieldType" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="TypeID" type="xs:string" use="required" />
    <xs:attribute name="DefaultBigEndian" type="xs:boolean" use="optional" />
    <xs:attribute name="DefaultStringEncoding" type="xs:string" use="optional" />
    <xs:attribute name="DefaultCharWidth" type="xs:unsignedInt" use="optional" />
    <xs:attribute name="DefaultFloatFormat" type="xs:string" use="optional" />
  </xs:complexType>
  <xs:complexType name="FieldType">
    <xs:attribute name="Name" type="xs:string" use="optional" />
    <xs:attribute name="Format" type="xs:string" use="optional" />
    <xs:attribute name="Length" type="xs:unsignedInt" use="optional" />
    <xs:attribute name="ElementCount" type="xs:unsignedInt" use="optional" />
    <xs:attribute name="ElementCountRef" type="xs:string" use="optional" />
    <xs:attribute name="FieldTerminator" type="xs:string" use="optional" />
  </xs:complexType>
  <xs:complexType name="BitString">
    <xs:complexContent>
      <xs:extension base="FieldType"></xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Integer">
    <xs:complexContent>
      <xs:extension base="FieldType">
        <xs:attribute name="Signed" type="xs:boolean" default="true" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="FloatingPoint">
    <xs:complexContent>
      <xs:extension base="FieldType">
        <xs:attribute name="FloatFormat" type="xs:string" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="CharString">
    <xs:complexContent>
      <xs:extension base="FieldType">
        <xs:attribute name="CharWidth" type="xs:unsignedInt" />
        <xs:attribute name="StringEncoding" type="xs:string" />
        <xs:attribute name="CharCountRef" type="xs:string" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="TypeReference">
```

```
    <xs:complexContent>
      <xs:extension base="FieldType">
        <xs:attribute name="TypeID" type="xs:string" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Int8">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="1" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="true" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Int16">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="2" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="true" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Int32">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="4" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="true" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Int64">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="8" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="true" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UInt8">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="1" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="false" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UInt16">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="2" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="false" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UInt32">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="4" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="false" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UInt64">
    <xs:complexContent>
      <xs:restriction base="Integer">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="8" />
        <xs:attribute name="Signed" type="xs:boolean" fixed="false" />
      </xs:restriction>
    </xs:complexContent>
```

```
  </xs:complexType>
  <xs:complexType name="Single">
    <xs:complexContent>
      <xs:restriction base="FloatingPoint">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="4" />
        <xs:attribute name="FloatFormat" type="xs:string" fixed="IEEE-754" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Double">
    <xs:complexContent>
      <xs:restriction base="FloatingPoint">
        <xs:attribute name="Length" type="xs:unsignedInt" fixed="8" />
        <xs:attribute name="FloatFormat" type="xs:string" fixed="IEEE-754" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Ascii">
    <xs:complexContent>
      <xs:restriction base="CharString">
        <xs:attribute name="CharWidth" type="xs:unsignedInt" fixed="1" />
        <xs:attribute name="StringEncoding" type="xs:string" fixed="ASCII" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Unicode">
    <xs:complexContent>
      <xs:restriction base="CharString">
        <xs:attribute name="CharWidth" type="xs:unsignedInt" fixed="2" />
        <xs:attribute name="StringEncoding" type="xs:string" fixed="UCS-2" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
</xs:schema>
```