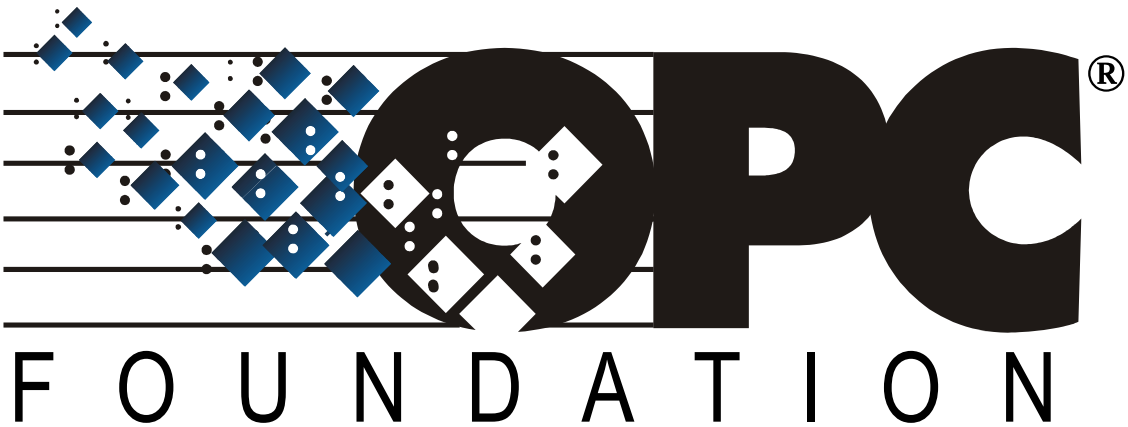

OPC Unified Architecture

Specification

Part 5: Information Model

Release 1.04

November 22, 2017

Specification Type:	Industry Standard Specification	Comments:	Report or view errata: http://www.opcfoundation.org/errata
Title:	OPC Unified Architecture Part 5 :Information Model	Date:	November 22, 2017
Version:	Release 1.04	Software:	MS-Word
		Source:	OPC UA Part 5 - Information Model Release 1.04 Specification.docx
Author:	OPC Foundation	Status:	Release

CONTENTS

FIGURES

TABLES

OPC FOUNDATION

_____

UNIFIED ARCHITECTURE –

FOREWORD

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

AGREEMENT OF USE

GENERAL PROVISIONS

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, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: http://www.opcfoundation.org/errata

Revision 1.04 Highlights

The following table includes the Mantis issues resolved with this revision.

Mantis ID	Summary	Resolution
3120	Handling of DataType Encoding Information	Added new DataTypes defined in Part 3 to handle new attribute on DataType NodeClass. Added Annex D describing the old approach of providing encoding information previously defined in Part 3. Moved definitions from this part to the annex. Moved information in 8.2.9 about DataTypeDictionaries to Annex D. Moved annex on OPC Binary from Part 3 to Annex E.
3219	Clarify what diagnostics nodes have to exist if diagnostics is turned off.	Changed description is 6.3.3 explaining what nodes have to exist and what nodes not.
3307	Remove reference to DataType Time	Table 115 had reference to DataType Time which does not exist, only DateTime and UtcTime. Removed entry.
3192	Missing description for Method	In 6.3.1 the description for Method ResendData was missing. Added description to paragraph.
2928 3146	Temporary File Transfer	Added concept allowing to temporary create file to write to or read from server in C.4.
3170	Clarification of 'serverHandles' parameter	In 9.1 corrected the Description of Argument 'serverHandles' to improve readability.
3349	Clarification of ResendData method	In 9.2 added additional description of functionality.
3469	Server's current Time Zone	In 6.3.1 added optional property CurrentTimeZone to ServerType
3162	SetTriggering Service maximum items	In 6.3.11 added definition for SetTriggering Service.
3636	new property on DataTypeDictionaryType	In D.5.2 added "Deprecated" property.
3642	New Variable type for Selection List	In 7.18 added SelectionListType Variable type
3623	Inconsistency in SessionDiagnosticsDataType and SessionDiagnosticsVariableType	Removed currentPublishTimerExpiration from Table 146 – SessionDiagnosticsDataType Structure
3189	*OptionSetType DataType* should be "OptionSet"	In 7.17 *OptionSetType* added example explaining why *OptionSet DataType* can be used
3474	SourceNode term clarification	Added the term "Property" when SourceNode is used to reference the Property of the BaseEventType in various sections.
3670	AuditSecurityEventType doesn't return actual occurring error	Added new property StatusCodeId to 6.4.4 AuditsecurityEventType.
3717	User Authentication	Added Annex F on UserAuthentication

Mantis ID	Summary	Resolution
3710	Allow Statemachine to expose currently available states and transitions	Added optional properties to FiniteStateMachineType and an example use of optional states B.4.5
3714	Add KeyValuePair DataType	Added KeyValuePair DataType as 12.21
3734	Decimal DataType	Added Decimal as a Part 3 DataType in Table 115 and Table 120
3703	FileTransferStateMachineType modelling rule	Changed states and transitions to have a no modelling rule in C.4.6.
3781	Add AudioVariableType	Added VariableType AudioVariableType in 7.19.
3755	OperationLimits interpretation of value=0	Clarification added to 6.3.11.
3750	Standard Type Definitions do not match model compiler	Removed WideChar, CharArray, and WideCharArray. Updated String and WideString in E.3.
3790	SessionDiagnosticsVariableType names contradict generated code	Changed names in Table 73 to match what is used by generated code.
2323	Clarification of ServerShutdown definition and behavior	Clarification added to Table 136 and Table 144.
3826	Add UrisVersion Property to Server Object	Added UrisVersion Property to ServerType in 6.3.1
3851	MaxStringLength Clarify	Added clarification of MaxStringLength and MaxByteStringLength Properties in 6.3.2

OPC Unified Architecture Specification

Part 5: Information Model

1 Scope

This specification defines the Information Model of the OPC Unified Architecture. The Information Model describes standardised *Nodes* of a *Server*'s *AddressSpace*. These *Nodes* are standardised types as well as standardised instances used for diagnostics or as entry points to server-specific *Nodes*. Thus, the Information Model defines the *AddressSpace* of an empty OPC UA *Server*. However, it is not expected that all *Servers* will provide all of these *Nodes*.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Part 1: OPC UA Specification: Part 1 – Concepts
http://www.opcfoundation.org/UA/Part1/

Part 3: OPC UA Specification: Part 3 – Address Space Model
http://www.opcfoundation.org/UA/Part3/

Part 4: OPC UA Specification: Part 4 – Services
http://www.opcfoundation.org/UA/Part4/

Part 6: OPC UA Specification: Part 6 – Mappings
http://www.opcfoundation.org/UA/Part6/

Part 7: OPC UA Specification: Part 7 – Profiles
http://www.opcfoundation.org/UA/Part7/

Part 9: OPC UA Specification: Part 9 – Alarms and conditions
http://www.opcfoundation.org/UA/Part9/

Part 10: OPC UA Specification: Part 10 – Programs
http://www.opcfoundation.org/UA/Part10/

Part 11: OPC UA Specification: Part 11 – Historical Access
http://www.opcfoundation.org/UA/Part11/

RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One:
http://www.ietf.org/rfc/rfc2045.txt

RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two:
https://www.ietf.org/rfc/rfc2046.txt

RFC 2047: Multipurpose Internet Mail Extensions (MIME) Part Three:
http://www.ietf.org/rfc/rfc2047.txt

ISO/IEC/IEEE 60559:2011 : Information technology – Microprocessor Systems – Floating-Point arithmetic

https://www.iso.org/standard/57469.html

XML Schema Part 1: Structures

http://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes

http://www.w3.org/TR/xmlschema-2/

XPATH: XML Path Language

http://www.w3.org/TR/xpath/

UTF-8: UTF-8, a transformation format of ISO 10646

http://www.ietf.org/rfc/rfc3629.txt

3 Terms, definitions and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in Part 1 and Part 3, as well as the following apply.

3.1.1
ClientUserId

String that identifies the user of the client requesting an action

Note 1 to entry: The *ClientUserId* is obtained directly or indirectly from the *UserIdentityToken* passed by the *Client* in the *ActivateSession Service* call. See 6.4.3 for details.

3.2 Abbreviations and symbols

UA Unified Architecture

XML Extensible Markup Language

3.3 Conventions for Node descriptions

Node definitions are specified using tables (see Table 2).

Attributes are defined by providing the *Attribute* name and a value, or a description of the value.

References are defined by providing the *ReferenceType* name, the *BrowseName* of the *TargetNode* and its *NodeClass*.

- If the *TargetNode* is a component of the *Node* being defined in the table the *Attributes* of the composed *Node* are defined in the same row of the table.
- The *DataType* is only specified for *Variables*; "[<number>]" indicates a single-dimensional array, for multi-dimensional arrays the expression is repeated for each dimension (e.g. [2][3] for a two-dimensional array). For all arrays the *ArrayDimensions* is set as identified by <number> values. If no <number> is set, the corresponding dimension is set to 0, indicating an unknown size. If no number is provided at all the *ArrayDimensions* can be omitted. If no brackets are provided, it identifies a scalar *DataType* and the *ValueRank* is set to the corresponding value (see Part 3). In addition, *ArrayDimensions* is set to null or is omitted. If it can be Any or ScalarOrOneDimension, the value is put into "{<value>}", so either "{Any}" or "{ScalarOrOneDimension}" and the *ValueRank* is set to the corresponding value (see Part 3) and the *ArrayDimensions* is set to null or is omitted. Examples are given in Table 1.

Table 1 – Examples of DataTypes

Notation	Data-Type	Value-Rank	Array-Dimensions	Description
Int32	Int32	−1	omitted or null	A scalar Int32.
Int32[]	Int32	1	omitted or {0}	Single-dimensional array of Int32 with an unknown size.
Int32[][]	Int32	2	omitted or {0,0}	Two-dimensional array of Int32 with unknown sizes for both dimensions.
Int32[3][]	Int32	2	{3,0}	Two-dimensional array of Int32 with a size of 3 for the first dimension and an unknown size for the second dimension.
Int32[5][3]	Int32	2	{5,3}	Two-dimensional array of Int32 with a size of 5 for the first dimension and a size of 3 for the second dimension.
Int32{Any}	Int32	−2	omitted or null	An Int32 where it is unknown if it is scalar or array with any number of dimensions.
Int32{ScalarOrOneDimension}	Int32	−3	omitted or null	An Int32 where it is either a single-dimensional array or a scalar.

- The TypeDefinition is specified for *Objects* and *Variables*.
- The TypeDefinition column specifies a symbolic name for a *NodeId*, i.e. the specified *Node* points with a *HasTypeDefinition Reference* to the corresponding *Node*.
- The *ModellingRule* of the referenced component is provided by specifying the symbolic name of the rule in the *ModellingRule* column. In the *AddressSpace*, the *Node* shall use a *HasModellingRule Reference* to point to the corresponding *ModellingRule Object*.

If the *NodeId* of a *DataType* is provided, the symbolic name of the *Node* representing the *DataType* shall be used.

Nodes of all other *NodeClasses* cannot be defined in the same table; therefore only the used *ReferenceType*, their *NodeClass* and their *BrowseName* are specified. A reference to another part of this document points to their definition.

Table 2 illustrates the table. If no components are provided, the DataType, TypeDefinition and ModellingRule columns may be omitted and only a Comment column is introduced to point to the *Node* definition.

Table 2 – Type Definition Table

Attribute	**Value**				
Attribute name	Attribute value. If it is an optional Attribute that is not set "--" will be used.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
ReferenceType name	*NodeClass* of the Target*Node*.	*BrowseName* of the target *Node*. If the *Reference* is to be instantiated by the server, then the value of the target Node's BrowseName is "--".	*DataType* of the referenced *Node*, only applicable for *Variables*.	*TypeDefinition* of the referenced *Node*, only applicable for *Variables* and *Objects*.	Referenced *ModellingRule* of the referenced *Object*.
NOTE Notes referencing footnotes of the table content.					

Components of *Nodes* can be complex that is containing components by themselves. The *TypeDefinition*, *NodeClass*, *DataType* and *ModellingRule* can be derived from the type definitions, and the symbolic name can be created as defined in 4.1. Therefore those containing components are not explicitly specified; they are implicitly specified by the type definitions.

4 NodeIds and BrowseNames

4.1 NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names. Part 6 defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this standard is its *BrowseName*, or, when it is part of another *Node*, the *BrowseName* of the other *Node*, a ".", and the *BrowseName* of itself. In this case "part of" means that the whole has a *HasProperty* or *HasComponent Reference* to its part. Since all *Nodes* not being part of another *Node* have a unique name in this standard, the symbolic name is unique. For example, the *ServerType* defined in 6.3.1 has the symbolic name "ServerType". One of its *InstanceDeclarations* would be identified as "ServerType.ServerCapabilities". Since this *Object* is complex, another *InstanceDeclaration* of the *ServerType* is "ServerType.ServerCapabilities.MinSupportedSampleRate". The *Server Object* defined in 8.3.2 is based on the ServerType and has the symbolic name "Server". Therefore, the instance based on the *InstanceDeclaration* described above has the symbolic name "Server.ServerCapabilities.MinSupportedSampleRate".

The NamespaceIndex for all *NodeIds* defined in this standard is 0. The namespace for this NamespaceIndex is specified in Part 3.

Note that this standard not only defines concrete *Nodes*, but also requires that some Nodes have to be generated, for example one for each Session running on the *Server*. The *NodeIds* of those *Nodes* are server-specific, including the Namespace. However the NamespaceIndex of those *Nodes* cannot be the NamespaceIndex 0, because they are not defined by the OPC Foundation but generated by the *Server*.

4.2 BrowseNames

The text part of the *BrowseNames* for all *Nodes* defined in this standard is specified in the tables defining the *Nodes*. The NamespaceIndex for all *BrowseNames* defined in this standard is 0.

5 Common Attributes

5.1 General

For all *Nodes* specified in this standard, the *Attributes* named in Table 3 shall be set as specified in Table 3.

Table 3 – Common Node Attributes

Attribute	Value
DisplayName	The *DisplayName* is a *LocalizedText*. Each server shall provide the *DisplayName* identical to the *BrowseName* of the *Node* for the LocaleId "en". Whether the server provides translated names for other LocaleIds is server-specific.
Description	Optionally a server-specific description is provided.
NodeClass	Shall reflect the *NodeClass* of the *Node.*
NodeId	The *NodeId* is described by *BrowseNames* as defined in 4.1 and defined in Part 6.
WriteMask	Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all non-server-specific *Attributes* to not writable. For example, the *Description Attribute* may be set to writable since a *Server* may provide a server-specific description for the *Node*. The *NodeId* shall not be writable, because it is defined for each *Node* in this standard.
UserWriteMask	Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply.
RolePermissions	Optionally server-specific role permissions can be provided.
UserRolePermissions	Optionally the role permissions of the current Session can be provided. The value is server-specifc and depend on the *RolePermissions Attribute* (if provided) and the current *Session*.
AccessRestrictions	Optionally server-specific access restrictions can be provided.

5.2 Objects

For all *Objects* specified in this standard, the *Attributes* named in Table 4 shall be set as specified in Table 4.

Table 4 – Common Object Attributes

Attribute	Value
EventNotifier	Whether the *Node* can be used to subscribe to *Events* or not is server-specific.

5.3 Variables

For all *Variables* specified in this standard, the *Attributes* named in Table 5 shall be set as specified in Table 5.

Table 5 – Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a server-specific minimum sampling interval is provided.
AccessLevel	The access level for *Variables* used for type definitions is server-specific, for all other *Variables* defined in this standard, the access level shall allow reading; other settings are server-specific.
UserAccessLevel	The value for the *UserAccessLevel Attribute* is server-specific. It is assumed that all *Variables* can be accessed by at least one user.
Value	For *Variables* used as *InstanceDeclarations,* the value is server-specific; otherwise it shall represent the value described in the text.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is server-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *Variable*.
Historizing	The value for the *Historizing Attribute* is server-specific.
AccessLevelEx	If the *AccessLevelEx Attribute* is provided, it shall have the bits 8, 9, and 10 set to 0, meaning that read and write operations on an individual *Variable* are atomic, and arrays can be partly written.

5.4 VariableTypes

For all *VariableTypes* specified in this standard, the *Attributes* named in Table 6 shall be set as specified in Table 6.

Table 6 – Common VariableType Attributes

Attributes	Value
Value	Optionally a server-specific default value can be provided.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is server-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *VariableType*.

5.5 Methods

For all *Methods* specified in this standard, the *Attributes* named in Table 7 shall be set as specified in Table 7.

Table 7 – Common Method Attributes

Attributes	Value
Executable	All *Methods* defined in this specification shall be executable (*Executable Attribute* set to “True”), unless it is defined differently in the *Method* definition.
UserExecutable	The value of the *UserExecutable Attribute* is server-specific. It is assumed that all *Methods* can be executed by at least one user.

6 Standard ObjectTypes

6.1 General

Typically, the components of an *ObjectType* are fixed and can be extended by subtyping. However, since each *Object* of an *ObjectType* can be extended with additional components, this standard allows extending the standard *ObjectTypes* defined in this document with additional components. Thereby, it is possible to express the additional information in the type definition that would already be contained in each *Object*. Some *ObjectTypes* already provide

entry points for server-specific extensions. However, it is not allowed to restrict the components of the standard *ObjectTypes* defined in this standard. An example of extending the *ObjectTypes* is putting the standard *Property NodeVersion* defined in Part 3 into the *BaseObjectType*, stating that each *Object* of the *Server* will provide a *NodeVersion*.

6.2 BaseObjectType

The *BaseObjectType* is used as type definition whenever there is an *Object* having no more concrete type definitions available. *Servers* should avoid using this *ObjectType* and use a more specific type, if possible. This *ObjectType* is the base *ObjectType* and all other *ObjectTypes* shall either directly or indirectly inherit from it. However, it might not be possible for *Servers* to provide all *HasSubtype References* from this *ObjectType* to its subtypes, and therefore it is not required to provide this information.

There are no *References* except for *HasSubtype References* specified for this *ObjectType*. It is formally defined in Table 8.

Table 8 – BaseObjectType Definition

Attribute	**Value**				
BrowseName	BaseObjectType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasSubtype	ObjectType	ServerType	Defined in 6.3.1		
HasSubtype	ObjectType	ServerCapabilitiesType	Defined in 6.3.2		
HasSubtype	ObjectType	ServerDiagnosticsType	Defined in 6.3.3		
HasSubtype	ObjectType	SessionsDiagnosticsSummaryType	Defined in 6.3.4		
HasSubtype	ObjectType	SessionDiagnosticsObjectType	Defined in 6.3.5		
HasSubtype	ObjectType	VendorServerInfoType	Defined in 6.3.6		
HasSubtype	ObjectType	ServerRedundancyType	Defined in 6.3.7		
HasSubtype	ObjectType	BaseEventType	Defined in 6.4.2		
HasSubtype	ObjectType	ModellingRuleType	Defined in 6.5		
HasSubtype	ObjectType	FolderType	Defined in 6.6		
HasSubtype	ObjectType	DataTypeEncodingType	Defined in 6.7		

6.3 ObjectTypes for the Server Object

6.3.1 ServerType

This *ObjectType* defines the capabilities supported by the OPC UA *Server*. It is formally defined in Table 9.

Table 9 – ServerType Definition

Attribute	Value			
BrowseName	ServerType			
IsAbstract	False			
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasProperty	Variable	ServerArray	String[] PropertyType	Mandatory
HasProperty	Variable	NamespaceArray	String[] PropertyType	Mandatory
HasProperty	Variable	UrisVersion	VersionTime PropertyType	Optional
HasComponent	Variable	ServerStatus[1]	ServerStatusDataType ServerStatusType	Mandatory
HasProperty	Variable	ServiceLevel	Byte PropertyType	Mandatory
HasProperty	Variable	Auditing	Boolean PropertyType	Mandatory
HasProperty	Variable	EstimatedReturnTime	DateTime PropertyType	Optional
HasProperty	Variable	LocalTime	TimeZoneDataType PropertyType	Optional
HasComponent	Object	ServerCapabilities[1]	- ServerCapabilitiesType	Mandatory
HasComponent	Object	ServerDiagnostics[1]	- ServerDiagnosticsType	Mandatory
HasComponent	Object	VendorServerInfo	- VendorServerInfoType	Mandatory
HasComponent	Object	ServerRedundancy[1]	- ServerRedundancyType	Mandatory
HasComponent	Object	Namespaces	- NamespacesType	Optional
HasComponent	Method	GetMonitoredItems	Defined in 9.1	Optional
HasComponent	Method	ResendData	Defined in 9.2	Optional
HasComponent	Method	SetSubscriptionDurable	Defined in 9.3	Optional
HasComponent	Method	RequestServerStateChange	Defined in 9.4	Optional
NOTE Containing *Objects* and *Variables* of these *Objects* and *Variables* are defined by their *BrowseName* defined in the corresponding *TypeDefinitionNode*. The *NodeId* is defined by the composed symbolic name described in 4.1.				

ServerArray defines an array of *Server* URIs. This *Variable* is also referred to as the *server table*. Each URI in this array represents a globally-unique logical name for a *Server* within the scope of the network in which it is installed. Each OPC UA *Server* instance has a single URI that is used in the *server table* of other OPC UA *Servers*. Index 0 is reserved for the URI of the local *Server*. Values above 0 are used to identify remote *Servers* and are specific to a *Server*. Part 4 describes discovery mechanism that can be used to resolve URIs into URLs. The *Server* URI is case sensitive.

The URI of the *ServerArray* with Index 0 shall be identical to the URI of the *NamespaceArray* with Index 1, since both represent the local *Server*.

The indexes into the *server table* are referred to as *server indexes* or *server names*. They are used in OPC UA *Services* to identify *TargetNodes* of *References* that reside in remote *Servers*. Clients may read the entire table or they may read individual entries in the table. The *Server* shall not modify or delete entries of this table while any client has an open session to the *Server*, because clients may cache the *server table*. A *Server* may add entries to the *server table* even if clients are connected to the *Server*.

NamespaceArray defines an array of namespace URIs. This *Variable* is also referred as *namespace table*. The indexes into the *namespace table* are referred to as *NamespaceIndexes*. *NamespaceIndexes* are used in *NodeIds* in OPC UA *Services*, rather than the longer namespace URI. Index 0 is reserved for the OPC UA namespace, and index 1 is reserved for the local *Server*. Clients may read the entire *namespace table* or they may read individual entries in the *namespace table*. The *Server* shall not modify or delete entries of the *namespace table* while any client has an open session to the *Server*, because clients may cache the

namespace table. A *Server* may add entries to the *namespace table* even if clients are connected to the *Server*. It is recommended that *Servers* not change the indexes of the *namespace table* but only add entries, because the client may cache *NodeIds* using the indexes. Nevertheless, it might not always be possible for *Servers* to avoid changing indexes in the *namespace table*. Clients that cache *NamespaceIndexes* of *NodeIds* should always check when starting a session to verify that the cached *NamespaceIndexes* have not changed.

UrisVersion defines the version of the *ServerArray* and the *NamespaceArray*. Everytime the *ServerArray* or the *NamespaceArray* is changed, the value of the *UrisVersion* shall be updated to a value greater than the previous value. The *UrisVersion Property* is used in combination with the *SessionlessInvoke Service* defined in Part 4. If a *Server* supports this *Service*, the *Server* shall support this *Property*. It is the responsibility of the *Server* to provide a consistent set of values for the *ServerArray*, *NamespaceArray* and the *UrisVersion Properties*. The *VersionTime DataType* is defined in Part 4.

ServerStatus contains elements that describe the status of the *Server*. See 12.10 for a description of its elements.

ServiceLevel describes the ability of the *Server* to provide its data to the client. The value range is from 0 to 255, where 0 indicates the worst and 255 indicates the best. Part 4 defines required sub-ranges for different scenarios. The intent is to provide the clients an indication of availability among redundant *Servers*.

Auditing is a Boolean specifying if the *Server* is currently generating audit events. It is set to TRUE if the *Server* generates audit events, otherwise to false. The *Profiles* defined in Part 7 specify what kind of audit events are generated by the *Server*.

EstimatedReturnTime indicates the time at which the *Server* is expected to have a *ServerStatus*.*State* of RUNNING_0. A *Client* that observes a shutdown or a *ServiceLevel* of 0 should either wait until after this time to attempt to reconnect to this *Server* or enter into slow retry logic. For example, most *Clients* will attempt to reconnect after a failure immediately and then progressively increase the delay between attempts until some maximum delay. This time can be used to trigger the *Client* to start its reconnect logic with some delay.

LocalTime is a structure containing the Offset and the DaylightSavingInOffset flag. The Offset specifies the time difference (in minutes) between the *Server* time in UTC and the local time at the *Server* location. If DaylightSavingInOffset is TRUE, then Standard/Daylight savings time (DST) at the *Server* location is in effect and Offset includes the DST correction. If FALSE then the Offset does not include DST correction and DST may or may not be in effect.

ServerCapabilities defines the capabilities supported by the OPC UA *Server*. See 6.3.2 for its description.

ServerDiagnostics defines diagnostic information about the OPC UA *Server*. See 6.3.3 for its description.

VendorServerInfo represents the browse entry point for vendor-defined *Server* information. This *Object* is required to be present even if there are no vendor-defined *Objects* beneath it. See 6.3.6 for its description.

ServerRedundancy describes the redundancy capabilities provided by the *Server*. This *Object* is required even if the *Server* does not provide any redundancy support. If the *Server* supports redundancy, then a subtype of *ServerRedundancyType* is used to describe its capabilities. Otherwise, it provides an *Object* of type *ServerRedundancyType* with the *Property* RedundancySupport set to none. See 6.3.7 for the description of *ServerRedundancyType*.

Namespaces provides a list of *NamespaceMetadataType Objects* with additional information about the namespaces used in the *Server*. See 6.3.14 for the description of *NamespaceMetadataType.*

The *GetMonitoredItems Method* is used to identify the *MonitoredItems* of a *Subscription*. It is defined in 9.1; the intended usage is defined in Part 4.

The *ResendData Method* is used to get the latest values of the data monitored items of a *Subscription*. It is defined in 9.2; the intended usage is defined in Part 4.

The *SetSubscriptionDurable Method* is used to set a *Subscription* into a mode where *MonitoredItem* data and event queues are stored and delivered even if an OPC UA *Client* was disconnected for a longer time or the OPC UA *Server* was restarted. It is defined in 9.3; the intended usage is defined in Part 4.

The *RequestServerStateChange Method* allows a *Client* to request a state change in the *Server*. It is defined in 9.4; the intended usage is defined in Part 4.

6.3.2 ServerCapabilitiesType

This *ObjectType* defines the capabilities supported by the OPC UA *Server*. It is formally defined in Table 10.

Table 10 – ServerCapabilitiesType Definition

Attribute	**Value**			
BrowseName	ServerCapabilitiesType			
IsAbstract	False			
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2				
HasProperty	Variable	ServerProfileArray	String[] PropertyType	Mandatory
HasProperty	Variable	LocaleIdArray	LocaleId[] PropertyType	Mandatory
HasProperty	Variable	MinSupportedSampleRate	Duration PropertyType	Mandatory
HasProperty	Variable	MaxBrowseContinuationPoints	UInt16 PropertyType	Mandatory
HasProperty	Variable	MaxQueryContinuationPoints	UInt16 PropertyType	Mandatory
HasProperty	Variable	MaxHistoryContinuationPoints	UInt16 PropertyType	Mandatory
HasProperty	Variable	SoftwareCertificates	SignedSoftwareCertificate[] PropertyType	Mandatory
HasProperty	Variable	MaxArrayLength	UInt32 PropertyType	Optional
HasProperty	Variable	MaxStringLength	UInt32 PropertyType	Optional
HasProperty	Variable	MaxByteStringLength	UInt32 PropertyType	Optional
HasComponent	Object	OperationLimits	-- OperationLimitsType	Optional
HasComponent	Object	ModellingRules	-- FolderType	Mandatory
HasComponent	Object	AggregateFunctions	-- FolderType	Mandatory
HasComponent	Object	RoleSet	RoleSetType	Optional

ServerProfileArray lists the *Profiles* that the *Server* supports. See Part 7 for the definitions of *Server Profiles*. This list should be limited to the *Profiles* the *Server* supports in its current configuration.

LocaleIdArray is an array of LocaleIds that are known to be supported by the *Server*. The *Server* might not be aware of all LocaleIds that it supports because it may provide access to underlying servers, systems or devices that do not report the LocaleIds that they support.

MinSupportedSampleRate defines the minimum supported sample rate, including 0, which is supported by the *Server*.

MaxBrowseContinuationPoints is an integer specifying the maximum number of parallel continuation points of the Browse *Service* that the *Server* can support per session. The value specifies the maximum the *Server* can support under normal circumstances, so there is no

guarantee the *Server* can always support the maximum. The client should not open more Browse calls with open continuation points than exposed in this *Variable*. The value 0 indicates that the *Server* does not restrict the number of parallel continuation points the client should use.

MaxQueryContinuationPoints is an integer specifying the maximum number of parallel continuation points of the QueryFirst *Services* that the *Server* can support per session. The value specifies the maximum the *Server* can support under normal circumstances, so there is no guarantee the *Server* can always support the maximum. The client should not open more QueryFirst calls with open continuation points than exposed in this *Variable*. The value 0 indicates that the *Server* does not restrict the number of parallel continuation points the client should use.

MaxHistoryContinuationPoints is an integer specifying the maximum number of parallel continuation points of the HistoryRead *Services* that the *Server* can support per session. The value specifies the maximum the *Server* can support under normal circumstances, so there is no guarantee the *Server* can always support the maximum. The client should not open more HistoryRead calls with open continuation points than exposed in this *Variable*. The value 0 indicates that the *Server* does not restrict the number of parallel continuation points the client should use.

SoftwareCertificates is an array of *SignedSoftwareCertificates* containing all *SoftwareCertificates* supported by the *Server*. A *SoftwareCertificate* identifies capabilities of the *Server*. It contains the list of *Profiles* supported by the *Server*. *Profiles* are described in Part 7.

The *MaxArrayLength Property* indicates the maximum length of a one or multidimensional array supported by Variables of the *Server*. In a multidimensional array it indicates the overall length. For example, a three-dimensional array of 2x3x10 has the array length of 60. The *Server* might further restrict the length for individual Variables without notice to the client. *Servers* may use the *Property MaxArrayLength* defined in Part 3 on individual *DataVariables* to specify the size on individual values. The individual *Property* may have a larger or smaller value than *MaxArrayLength*.

The *MaxStringLength Property* indicates the maximum number of bytes in *Strings* supported by *Variables* of the *Server*. *Servers* may override this setting by adding the *MaxStringLength Property* defined in Part 3 to an individual *DataVariable*. If a *Server* does not impose a maximum number of bytes or is not able to determine the maximum number of bytes this *Property* shall not be provided.

The *MaxByteStringLength Property* indicates the maximum number of bytes in a *ByteString* supported by *Variables* of the *Server*. It also specifies the default maximum size of a *FileType Object's* read and write buffers. *Servers* may override this setting by adding the *MaxByteStringLength Property* defined in Part 3 to an individual *DataVariable* or *FileType Object.* If a *Server* does not impose a maximum number of bytes or is not able to determine the maximum number of bytes this *Property* shall not be provided.

OperationLimits is an entry point to access information on operation limits of the *Server*, for example the maximum length of an array in a read *Service* call.

ModellingRules is an entry point to browse to all *ModellingRules* supported by the *Server*. All *ModellingRules* supported by the *Server* should be able to be browsed starting from this *Object*.

AggregateFunctions is an entry point to browse to all *AggregateFunctions* supported by the *Server*. All *AggregateFunctions* supported by the *Server* should be able to be browsed starting from this *Object*. AggregateFunctions are Objects of *AggregateFunctionType*.

The *RoleSet Object* is used to publish all *Roles* supported by the *Server*. The *RoleSetType* is specified in F.2

When vendors expose their own capabilities they should add additional *Nodes* to the standard *ServerCapabilities Object* instance.

6.3.3 ServerDiagnosticsType

This *ObjectType* defines diagnostic information about the OPC UA *Server*. This *ObjectType* is formally defined in Table 11.

Table 11 – ServerDiagnosticsType Definition

Attribute	Value			
BrowseName	ServerDiagnosticsType			
IsAbstract	False			
References	**Node Class**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasComponent	Variable	ServerDiagnosticsSummary	ServerDiagnosticsSummaryDataType ServerDiagnosticsSummaryType	Mandatory
HasComponent	Variable	SamplingIntervalDiagnosticsArray	SamplingIntervalDiagnosticsDataType[] SamplingIntervalDiagnosticsArrayType	Optional
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Mandatory
HasComponent	Object	SessionsDiagnosticsSummary	-- SessionsDiagnosticsSummaryType	Mandatory
HasProperty	Variable	EnabledFlag	Boolean PropertyType	Mandatory

ServerDiagnosticsSummary contains diagnostic summary information for the *Server*, as defined in 12.9.

SamplingIntervalDiagnosticsArray is an array of diagnostic information per sampling rate as defined in 12.8. There is one entry for each sampling rate currently used by the *Server*. Its *TypeDefinitionNode* is the *VariableType SamplingIntervalDiagnosticsArrayType*, providing a *Variable* for each entry in the array, as defined in 7.9.

The sampling interval diagnostics are only collected by *Servers* which use a fixed set of sampling intervals. In these cases, length of the array and the set of contained *Variables* will be determined by the *Server* configuration and the *NodeId* assigned to a given sampling interval diagnostics variable shall not change as long as the *Server* configuration does not change. A *Server* may not expose the SamplingIntervalDiagnosticsArray if it does not use fixed sampling rates.

SubscriptionDiagnosticsArray is an array of Subscription diagnostic information per subscription, as defined in 12.15. There is one entry for each Notification channel actually established in the *Server*. Its *TypeDefinitionNode* is the *VariableType* SubscriptionDiagnosticsArrayType, providing a *Variable* for each entry in the array as defined in 7.11. Those *Variables* are also used as *Variables* referenced by other *Variables*.

SessionsDiagnosticsSummary contains diagnostic information per session, as defined in 6.3.4.

EnabledFlag identifies whether or not diagnostic information is collected by the *Server*. It can also be used by a client to enable or disable the collection of diagnostic information of the *Server*. The following settings of the Boolean value apply: TRUE indicates that the *Server* collects diagnostic information, and setting the value to TRUE leads to resetting and enabling the collection. FALSE indicates that no diagnostic information is collected, and setting the value to FALSE disables the collection without resetting the diagnostic values.

When diagnostics are turned off, the *Server* can return Bad_NodeIdUnknown for all static diagnostic *Nodes* except the *EnabledFlag Property*. Dynamic diagnostic *Nodes* (such as the *Session Nodes*) will not appear in the *AddressSpace*.

If collection of diagnostic information is not supported at all, the EnabledFlag Property will be read only.

6.3.4 SessionsDiagnosticsSummaryType

This *ObjectType* defines diagnostic information about the sessions of the OPC UA *Server*. This *ObjectType* is formally defined in Table 12.

Table 12 – SessionsDiagnosticsSummaryType Definition

Attribute		Value		
BrowseName		SessionsDiagnosticsSummaryType		
IsAbstract		False		
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasComponent	Variable	SessionDiagnosticsArray	SessionDiagnosticsDataType[] SessionDiagnosticsArrayType	Mandatory
HasComponent	Variable	SessionSecurityDiagnosticsArray	SessionSecurityDiagnosticsDataType[] SessionSecurityDiagnosticsArrayType	Mandatory
HasComponent	Object	<ClientName>	-- SessionDiagnosticsObjectType	Optional Placeholder
NOTE This row represents no *Node* in the *AddressSpace*. It is a placeholder pointing out that instances of the *ObjectType* will have those *Objects*.				

SessionDiagnosticsArray provides an array with an entry for each session in the *Server* having general diagnostic information about a session.

SessionSecurityDiagnosticsArray provides an array with an entry for each active session in the *Server* having security-related diagnostic information about a session. Since this information is security-related, it should not be made accessible to all users, but only to authorised users.

For each session of the *Server*, this *Object* also provides an *Object* representing the session, indicated by <*ClientName*>. The BrowseName could be derived from the *sessionName* defined in the *CreateSession Service* (Part 4) or some other server-specific mechanisms. It is of the *ObjectType* SessionDiagnosticsObjectType, as defined in 6.3.5.

6.3.5 SessionDiagnosticsObjectType

This *ObjectType* defines diagnostic information about a session of the OPC UA *Server*. This *ObjectType* is formally defined in Table 13.

Table 13 – SessionDiagnosticsObjectType Definition

Attribute		Value		
BrowseName		SessionDiagnosticsObjectType		
IsAbstract		False		
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	Mandatory
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	Mandatory
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Mandatory

SessionDiagnostics contains general diagnostic information about the session; the *SessionSecurityDiagnostics Variable* contains security-related diagnostic information. Because the information of the second *Variable* is security-related, it should not be made accessible to all users, but only to authorised users.

SubscriptionDiagnosticsArray is an array of Subscription diagnostic information per opened subscription, as defined in 12.15. Its *TypeDefinitionNode* is the *VariableType* SubscriptionDiagnosticsArrayType providing a *Variable* for each entry in the array, as defined in 7.11.

6.3.6 VendorServerInfoType

This *ObjectType* defines a placeholder *Object* for vendor-specific information about the OPC UA *Server*. This *ObjectType* defines an empty *ObjectType* that has no components. It shall be subtyped by vendors to define their vendor-specific information. This *ObjectType* is formally defined in Table 14.

Table 14 – VendorServerInfoType Definition

Attribute	Value				
BrowseName	VendorServerInfoType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2					

6.3.7 ServerRedundancyType

This *ObjectType* defines the redundancy capabilities supported by the OPC UA *Server*. It is formally defined in Table 15.

Table 15 – ServerRedundancyType Definition

Attribute	Value				
BrowseName	ServerRedundancyType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**Type Definition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	RedundancySupport	RedundancySupport	PropertyType	Mandatory
HasSubtype	ObjectType	TransparentRedundancyType	Defined in 6.3.8		
HasSubtype	ObjectType	NonTransparentRedundancyType	Defined in 6.3.9		

RedundancySupport indicates what redundancy is supported by the *Server*. Its values are defined in 12.5. It shall be set to NONE_0 for all instances of the *ServerRedundancyType* using the *ObjectType* directly (no subtype).

6.3.8 TransparentRedundancyType

This *ObjectType* is a subtype of *ServerRedundancyType* and is used to identify the capabilities of the OPC UA *Server* for server-controlled redundancy with a transparent switchover for the client. It is formally defined in Table 16.

Table 16 – TransparentRedundancyType Definition

Attribute	Value				
BrowseName	TransparentRedundancyType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ServerRedundancyType defined in 6.3.7, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	CurrentServerId	String	PropertyType	Mandatory
HasProperty	Variable	RedundantServerArray	RedundantServerDataType[]	PropertyType	Mandatory

RedundancySupport is inherited from the *ServerRedundancyType*. It shall be set to TRANSPARENT_4 for all instances of the *TransparentRedundancyType*.

Although, in a transparent switchover scenario, all redundant *Servers* serve under the same URI to the *Client*, it may be required to track the exact data source on the *Client*. Therefore, *CurrentServerId* contains an identifier of the currently-used *Server* in the *Redundant Set*. This *Server* is valid only inside a *Session*; if a *Client* opens several *Sessions*, different *Servers* of the redundant set of *Servers* may serve it in different *Sessions*. The value of the *CurrentServerId* may change due to *Failover* or load balancing, so a *Client* that needs to track its data source shall subscribe to this *Variable*.

As diagnostic information, the *RedundantServerArray* contains an array of available *Servers* in the *Redundant Set*; including their service levels (see 12.7). This array may change during a *Session*.

6.3.9 NonTransparentRedundancyType

This *ObjectType* is a subtype of *ServerRedundancyType* and is used to identify the capabilities of the OPC UA *Server* for non-transparent redundancy. It is formally defined in Table 17.

Table 17 – NonTransparentRedundancyType Definition

Attribute	Value				
BrowseName	NonTransparentRedundancyType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ServerRedundancyType defined in 6.3.7, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ServerUriArray	String[]	PropertyType	Mandatory
HasSubtype	ObjectType	NonTransparentNetworkRedundancyType	Defined in 6.3.10		

ServerUriArray is an array with the URI of all redundant *Servers* of the OPC UA *Server*. See Part 4 for the definition of redundancy in this standard. In a non-transparent redundancy environment, the *Client* is responsible to subscribe to the redundant *Servers*. Therefore the *Client* might open a session to one or more redundant *Servers* of this array. The *ServerUriArray* shall contain the local *Server*.

RedundancySupport is inherited from the *ServerRedundancyType*. It shall be set to COLD_1, WARM_2, HOT_3 or HOT_AND_MIRRORED_5 for all instances of the *NonTransparentRedundancyType*. It defines the redundancy support provided by the *Server*. Its intended use is defined in Part 4.

6.3.10 NonTransparentNetworkRedundancyType

This *ObjectType* is a subtype of *NonTransparentRedundancyType* and is used to identify the capabilities of the OPC UA *Server* for non-transparent network redundancy. It is formally defined in Table 18.

Table 18 – NonTransparentNetworkRedundancyType Definition

Attribute	Value				
BrowseName	NonTransparentNetworkRedundancyType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the NonTransparentRedundancyType defined in 6.3.9, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ServerNetworkGroups	NetworkGroupDataType[]	PropertyType	Mandatory

Clients switching between network paths to the same *Server* behave the same as HotAndMirrored redundancy. *Server* and network redundancy can be combined. In the combined approach it is important for the *Client* to know which ServerUris belong to the same *Server* representing different network paths and which ServerUris represent different Servers. Therefore, a *Server* implementing non-transparent network redundancy shall use the *NonTransparentNetworkRedundancyType* to identify its redundancy support.

RedundancySupport is inherited from the *ServerRedundancyType*. It shall be set to COLD_1, WARM_2, HOT_3 or HOT_AND_MIRRORED_5 for all instances of the *NonTransparentNetworkRedundancyType*. If no *Server* redundancy is supported (the *ServerUriArray* only contains one entry), the *RedundancySupport* shall be set to HOT_AND_MIRRORED_5.

The *ServerNetworkGroups* contains an array of *NetworkGroupDataType*. The URIs of the *Servers* in that array (in the *serverUri* of the structure) shall be exactly the same as the ones provided in the *ServerUriArray*. However, the order might be different. Thus the array represents a list of HotAndMirrored redundant *Servers*. If a *Server* only supports network redundancy, it has only one entry in the *ServerNetworkGroups*. The *networkPaths* in the structure represents the redundant network paths for each of the *Servers*. The *networkPaths* describes the different paths (one entry for each path) ordered by priority. Each network path contains an *endpointUrlList* having an array of Strings each containing a URL of an *Endpoint.* This allows using different protocol options for the same network path.

The *Endpoints* provided shall match with the *Endpoints* provided by the *GetEndpoints Service* of the corresponding *Server*.

6.3.11 OperationLimitsType

This *ObjectType* is a subtype of *FolderType* and is used to identify the operation limits of the OPC UA *Server*. It is formally defined in Table 19.

Table 19 – OperationLimitsType Definition

Attribute	Value				
BrowseName	OperationLimitsType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the FolderType defined in 6.6, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	MaxNodesPerRead	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryReadData	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryReadEvents	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerWrite	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryUpdateData	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryUpdateEvents	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerMethodCall	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerBrowse	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerRegisterNodes	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerTranslateBrowsePaths ToNodeIds	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerNodeManagement	UInt32	PropertyType	Optional
HasProperty	Variable	MaxMonitoredItemsPerCall	UInt32	PropertyType	Optional

Any operational limits *Property* that is provided shall have a non zero value.

The *MaxNodesPerRead Property* indicates the maximum size of the nodesToRead array when a *Client* calls the Read *Service*.

The *MaxNodesPerHistoryReadData Property* indicates the maximum size of the nodesToRead array when a *Client* calls the HistoryRead *Service* using the historyReadDetails RAW, PROCESSED, MODIFIED or ATTIME.

The *MaxNodesPerHistoryReadEvents Property* indicates the maximum size of the nodesToRead array when a *Client* calls the HistoryRead *Service* using the historyReadDetails EVENTS.

The *MaxNodesPerWrite Property* indicates the maximum size of the nodesToWrite array when a *Client* calls the Write *Service*.

The *MaxNodesPerHistoryUpdateData Property* indicates the maximum size of the historyUpdateDetails array supported by the *Server* when a *Client* calls the HistoryUpdate *Service*.

The *MaxNodesPerHistoryUpdateEvents Property* indicates the maximum size of the historyUpdateDetails array when a *Client* calls the HistoryUpdate *Service*.

The *MaxNodesPerMethodCall Property* indicates the maximum size of the methodsToCall array when a *Client* calls the Call *Service*.

The *MaxNodesPerBrowse Property* indicates the maximum size of the nodesToBrowse array when calling the Browse *Service* or the continuationPoints array when a *Client* calls the BrowseNext *Service*.

The *MaxNodesPerRegisterNodes Property* indicates the maximum size of the nodesToRegister array when a *Client* calls the RegisterNodes *Service* and the maximum size of the nodesToUnregister when calling the UnregisterNodes *Service*.

The *MaxNodesPerTranslateBrowsePathsToNodeIds Property* indicates the maximum size of the browsePaths array when a *Client* calls the TranslateBrowsePathsToNodeIds *Service*.

The *MaxNodesPerNodeManagement Property* indicates the maximum size of the nodesToAdd array when a *Client* calls the AddNodes *Service*, the maximum size of the referencesToAdd

array when a *Client* calls the AddReferences *Service*, the maximum size of the nodesToDelete array when a *Client* calls the DeleteNodes *Service*, and the maximum size of the referencesToDelete array when a *Client* calls the DeleteReferences *Service*.

The *MaxMonitoredItemsPerCall Property* indicates

- the maximum size of the itemsToCreate array when a *Client* calls the CreateMonitoredItems *Service,*
- the maximum size of the itemsToModify array when a *Client* calls the ModifyMonitoredItems *Service*,
- the maximum size of the monitoredItemIds array when a *Client* calls the SetMonitoringMode *Service* or the DeleteMonitoredItems *Service*,
- the maximum size of the sum of the linksToAdd and linksToRemove arrays when a *Client* calls the SetTriggering *Service*.

6.3.12 AddressSpaceFileType

This *ObjectType* defines the file for a namespace provided by the OPC UA *Server*. It is formally defined in Table 20. It represents an XML address space file using the XML schema defined in Part 6.

Table 20 – AddressSpaceFileType Definition

Attribute	**Value**				
BrowseName	AddressSpaceFileType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the FileType defined in C.2					
HasComponent	Method	ExportNamespace	The method has no parameters.		Optional

The *ExportNamespace Method* provides a way to export the namespace from the *Server AddressSpace* to the XML file represented by the *AddressSpaceFileType*. *Value Attributes* are only exported if they represent static configuration information. The client is expected to call the *ExportNamespace Method* first to update the XML file and then access the file with the *Methods* defined in the *FileType*.

Servers might provide some vendor-specific mechanisms importing parts of an address space as subtype of this *ObjectType*, for example by defining appropriate *Methods*.

6.3.13 NamespaceMetadataType

This *ObjectType* defines the metadata for a namespace provided by the *Server*. It is formally defined in Table 21.

Instances of this *Object* allow *Servers* to provide more information like version information in addition to the namespace URI. Important information for aggregating *Servers* is provided by the *StaticNodeIdTypes, StaticNumericNodeIdRange and StaticStringNodeIdPattern Properties*.

Table 21 – NamespaceMetadataType Definition

Attribute	Value				
BrowseName	NamespaceMetadataType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	NamespaceUri	String	PropertyType	Mandatory
HasProperty	Variable	NamespaceVersion	String	PropertyType	Mandatory
HasProperty	Variable	NamespacePublicationDate	DateTime	PropertyType	Mandatory
HasProperty	Variable	IsNamespaceSubset	Boolean	PropertyType	Mandatory
HasProperty	Variable	StaticNodeIdTypes	IdType[]	PropertyType	Mandatory
HasProperty	Variable	StaticNumericNodeIdRange	NumericRange[]	PropertyType	Mandatory
HasProperty	Variable	StaticStringNodeIdPattern	String	PropertyType	Mandatory
HasComponent	Object	NamespaceFile	-	AddressSpaceFileType	Optional
HasProperty	Variable	DefaultRolePermissions	RolePermission Type[]	PropertyType	Optional
HasProperty	Variable	DefaultUserRolePermissions	RolePermission Type[]	PropretyType	Optional
HasProperty	Variable	DefaultAccessRestrictions	Uint16	PropertyType	Optional

The *BrowseName* of instances of this type shall be derived from the represented namespace. This can, for example, be done by using the index of the namespace in the *NamespaceArray* as *namespaceIndex* of the *QualifiedName* and the namespace URI as *name* of the *QualifiedName*.

The *NamespaceUri Property* contains the namespace represented by an instance of the MetaDataType.

The *NamespaceVersion Property* provides version information for the namespace. It is intended for display purposes and shall not be used to programmatically identify the latest version. If there is no formal version defined for the namespace this *Property* shall be set to a null *String*.

The *NamespacePublicationDate* Property provides the publication date of the namespace version. This *Property* value can be used by *Clients* to determine the latest version if different versions are provided by different *Servers*. If there is no formal publication date defined for the namespace this *Property* shall be set to a null *DateTime*.

The *IsNamespaceSubset Property* defines whether all *Nodes* of the namespace are accessible in the *Server* or only a subset. It is set to FALSE if the full namespace is provided and TRUE if not. If the completeness is unknown then this *Property* shall be set to TRUE.

Static *Nodes* are identical for all *Attributes* in all *Servers*, including the *Value Attribute*. For *TypeDefinitionNodes*, also the *InstanceDeclarations* shall be identical. That means that for static *Nodes* the semantic is always the same. Namespaces with static *Nodes* are for example namespaces defined by standard bodies like the OPC Foundation. This is important information for aggregating *Servers.* If the namespace is dynamic and used in several *Servers* the aggregating *Server* needs to distinguish the namespace for each aggregated *Server*. The static *Nodes* of a namespace only need to be handled once, even if they are used by several aggregated *Servers*.

The *StaticNodeIdTypes Property* provides a list of *IdTypes* used for static *Nodes*. All *Nodes* in the *AddressSpace* of the namespace using one of the *IdTypes* in the array shall be static *Nodes*.

The *StaticNumericNodeIdRange Property* provides a list of *NumericRanges* used for numeric *NodeIds* of static *Nodes*. If the *StaticNodeIdTypes Property* contains an entry for numeric NodeIds then this Property is ignored.

The *StaticStringNodeIdPattern Property* provides a regular expression as defined for the *Like Operator* defined in Part 4 to filter for string *NodeIds* of static *Nodes*. If the *StaticNodeIdTypes Property* contains an entry for string *NodeIds* then this *Property* is ignored.

The *Object NamespaceFile* contains all *Nodes* and *References* of the namespace in an XML file where the Information Model XML Schema is defined in Part 6. The XML file is provided through an *AddressSpaceFileType Object*.

The *DefaultRolePermissions Property* provides the default permissions if a *Server* supports *RolePermissions* for the *Namespace.* A *Node* in the *Namespace* overrides this default by adding a *RolePermissions Attribute* to the *Node*. If a *Server* implements a vendor-specific *RolePermissions* model for a *Namespace,* it does not add the *DefaultRolePermissions Property* to the *NamespaceMetadata Object.*

The *DefaultUserRolePermissions Property* provides the default user permissions if a *Server* supports *UserRolePermissions* for the *Namespace.* A *Node* in the *Namespace* overrides this default by adding a *UserRolePermissions Attribute* to the *Node*. If a *Server* implements a vendor-specific *UserRolePermissions* model for a *Namespace,* it does not add the *DefaultUserRolePermissions Property* to the *NamespaceMetadata Object.*

The *DefaultAccessRestrictions Property* is present if a *Server* supports *AccessRestrictions* for the *Namespace* and provides the defaults*. A Node* in the *Namespace* overrides this default by adding a *AccessRestrictions Attribute* to the *Node.* If a *Server* implements a vendor-specific *AccessRestriction* model for a *Namespace,* it does not add the *DefaultAccessRestrictions Property* to the *NamespaceMetadata Object.*

6.3.14 NamespacesType

This *ObjectType* defines a list of *NamespaceMetadataType Objects* provided by the *Server*. It is formally defined in Table 22.

Table 22 – NamespacesType Definition

Attribute	**Value**				
BrowseName	NamespacesType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasComponent	Object	<NamespaceIdentifier>	-	NamespaceMetadataType	OptionalPlaceholder

The *ObjectType* contains a list of *NamespaceMetadataType Objects* representing the namespaces in the *Server*. The *BrowseName* of an *Object* shall be derived from the namespace represented by the *Object*. This can, for example, be done by using the index of the namespace in the *NamespaceArray* as *namespaceIndex* of the *QualifiedName* and the namespace URI as *name* of the *QualifiedName*. *Clients* should not assume that all namespaces provided by a *Server* are present in this list as a namespace may not provide the information necessary to fill all mandatory *Properties* of the *NamespaceMetadataType*.

6.4 ObjectTypes used as EventTypes

6.4.1 General

This International Standard defines standard *EventTypes*. They are represented in the *AddressSpace* as *ObjectTypes*. The *EventTypes* are already defined in Part 3. The following subclauses specify their representation in the *AddressSpace*.

6.4.2 BaseEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 23.

Table 23 – BaseEventType Definition

Attribute	Value				
BrowseName	BaseEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in 6.2					
HasSubtype	ObjectType	AuditEventType	Defined in 6.4.3		
HasSubtype	ObjectType	SystemEventType	Defined in 6.4.28		
HasSubtype	ObjectType	BaseModelChangeEventType	Defined in 6.4.31		
HasSubtype	ObjectType	SemanticChangeEventType	Defined in 6.4.33		
HasSubtype	ObjectType	EventQueueOverflowEventType	Defined in 6.4.34		
HasSubtype	ObjectType	ProgressEventType	Defined in 6.4.35		
HasProperty	Variable	EventId	ByteString	PropertyType	Mandatory
HasProperty	Variable	EventType	NodeId	PropertyType	Mandatory
HasProperty	Variable	SourceNode	NodeId	PropertyType	Mandatory
HasProperty	Variable	SourceName	String	PropertyType	Mandatory
HasProperty	Variable	Time	UtcTime	PropertyType	Mandatory
HasProperty	Variable	ReceiveTime	UtcTime	PropertyType	Mandatory
HasProperty	Variable	LocalTime	TimeZoneDataType	PropertyType	Optional
HasProperty	Variable	Message	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	Severity	UInt16	PropertyType	Mandatory

EventId is generated by the *Server* to uniquely identify a particular *Event Notification*. The *Server* is responsible to ensure that each *Event* has its unique *EventId*. It may do this, for example, by putting GUIDs into the ByteString. Clients can use the *EventId* to assist in minimizing or eliminating gaps and overlaps that may occur during a redundancy failover. The *EventId* shall always be returned as value and the *Server* is not allowed to return a *StatusCode* for the *EventId* indicating an error.

EventType describes the specific type of *Event*. The *EventType* shall always be returned as value and the *Server* is not allowed to return a *StatusCode* for the *EventType* indicating an error.

The *SourceNode Property* identifies the *Node* that the *Event* originated from. If the *Event* is not specific to a *Node* the *NodeId* is set to null. Some subtypes of this *BaseEventType* may define additional rules for the *SourceNode Property*.

SourceName provides a description of the source of the *Event*. This could be the string-part of the *DisplayName* of the *Event* source using the default locale of the server, if the *Event* is specific to a *Node,* or some server-specific notation.

Time provides the time the *Event* occurred. This value is set as close to the event generator as possible. It often comes from the underlying system or device. Once set, intermediate OPC UA *Servers* shall not alter the value.

ReceiveTime provides the time the OPC UA *Server* received the *Event* from the underlying device of another *Server*. *ReceiveTime* is analogous to *ServerTimestamp* defined in Part 4, i.e. in the case where the OPC UA *Server* gets an *Event* from another OPC UA *Server*, each *Server* applies its own *ReceiveTime*. That implies that a *Client* may get the same *Event,* having the same *EventId*, from different *Servers* having different values of the *ReceiveTime*. The *ReceiveTime* shall always be returned as value and the *Server* is not allowed to return a *StatusCode* for the *ReceiveTime* indicating an error.

LocalTime is a structure containing the Offset and the DaylightSavingInOffset flag. The Offset specifies the time difference (in minutes) between the *Time Property* and the time at the location in which the event was issued. If DaylightSavingInOffset is TRUE, then Standard/Daylight savings time (DST) at the originating location is in effect and Offset includes the DST correction. If FALSE then the Offset does not include DST correction and DST may or may not have been in effect.

Message provides a human-readable and localizable text description of the *Event*. The *Server* may return any appropriate text to describe the *Event*. A null string is not a valid value; if the

Server does not have a description, it shall return the string part of the *BrowseName* of the *Node* associated with the *Event*.

Severity is an indication of the urgency of the *Event*. This is also commonly called “priority”. Values will range from 1 to 1 000, with 1 being the lowest severity and 1 000 being the highest. Typically, a severity of 1 would indicate an *Event* which is informational in nature, while a value of 1 000 would indicate an *Event* of catastrophic nature, which could potentially result in severe financial loss or loss of life.

It is expected that very few *Server* implementations will support 1 000 distinct severity levels. Therefore, *Server* developers are responsible for distributing their severity levels across the 1 to 1 000 range in such a manner that clients can assume a linear distribution. For example, a client wishing to present five severity levels to a user should be able to do the following mapping:

Client Severity	OPC Severity
HIGH	801 – 1 000
MEDIUM HIGH	601 – 800
MEDIUM	401 – 600
MEDIUM LOW	201 – 400
LOW	1 – 200

In many cases a strict linear mapping of underlying source severities to the OPC Severity range is not appropriate. The *Server* developer will instead intelligently map the underlying source severities to the 1 to 1 000 OPC Severity range in some other fashion. In particular, it is recommended that *Server* developers map *Events* of high urgency into the OPC severity range of 667 to 1 000, *Events* of medium urgency into the OPC severity range of 334 to 666 and *Events* of low urgency into OPC severities of 1 to 333.

For example, if a source supports 16 severity levels that are clustered such that severities 0 to 2 are considered to be LOW, 3 to 7 are MEDIUM and 8 to 15 are HIGH, then an appropriate mapping might be as follows:

OPC Range	Source Severity	OPC Severity
HIGH (667 – 1 000)	15	1 000
	14	955
	13	910
	12	865
	11	820
	10	775
	9	730
	8	685
MEDIUM (334 – 666)	7	650
	6	575
	5	500
	4	425
	3	350
LOW (1 – 333)	2	300
	1	150
	0	1

Some *Servers* might not support any *Events* which are catastrophic in nature, so they may choose to map all of their severities into a subset of the 1 to 1 000 range (for example, 1 to 666). Other *Servers* might not support any *Events* which are merely informational, so they may choose to map all of their severities into a different subset of the 1 to 1 000 range (for example, 334 to 1 000).

The purpose of this approach is to allow clients to use severity values from multiple *Servers* from different vendors in a consistent manner. Additional discussions of severity can be found in Part 9.

6.4.3 AuditEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 24.

Table 24 – AuditEventType Definition

Attribute		Value			
BrowseName		AuditEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditSecurityEventType	Defined in 6.4.4		
HasSubtype	ObjectType	AuditNodeManagementEventType	Defined in 6.4.19		
HasSubtype	ObjectType	AuditUpdateEventType	Defined in 6.4.24		
HasSubtype	ObjectType	AuditUpdateMethodEventType	Defined in 6.4.27		
HasProperty	Variable	ActionTimeStamp	UtcTime	PropertyType	Mandatory
HasProperty	Variable	Status	Boolean	PropertyType	Mandatory
HasProperty	Variable	ServerId	String	PropertyType	Mandatory
HasProperty	Variable	ClientAuditEntryId	String	PropertyType	Mandatory
HasProperty	Variable	ClientUserId	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2.

ActionTimeStamp identifies the time the user initiated the action that resulted in the *AuditEvent* being generated. It differs from the *Time Property* because this is the time the server generated the *AuditEvent* documenting the action.

Status identifies whether the requested action could be performed (set *Status* to TRUE) or not (set *Status* to FALSE).

ServerId uniquely identifies the *Server* generating the *Event*. It identifies the *Server* uniquely even in a server-controlled transparent redundancy scenario where several *Servers* may use the same URI.

ClientAuditEntryId contains the human-readable *AuditEntryId* defined in Part 3.

The *ClientUserId* identifies the user of the client requesting an action. The *ClientUserId* can be obtained from the *UserIdentityToken* passed in the *ActivateSession* call. If the *UserIdentityToken* is a *UserNameIdentityToken* then the *ClientUserId* is the *UserName.* If the *UserIdentityToken* is an *X509IdentityToken* then the *ClientUserId* is the X509 Subject Name of the *Certificate*. If the *UserIdentityToken* is an *IssuedIdentityToken* then the *ClientUserId* shall be a string that represents the owner of the token. The best choice for the string depends on the type of *IssuedIdentityToken.* If an *AnonymousIdentityToken* was used, the value is null.

6.4.4 AuditSecurityEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 25.

Table 25 – AuditSecurityEventType Definition

Attribute		Value			
BrowseName		AuditSecurityEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the AuditEventType defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditChannelEventType	Defined in 6.4.5		
HasSubtype	ObjectType	AuditSessionEventType	Defined in 6.4.7		
HasSubtype	ObjectType	AuditCertificateEventType	Defined in 6.4.12		
HasProperty	Variable	StatusCodeId	StatusCode	PropertyType	Optional

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. There are no additional *Properties* defined for this *EventType*.

The optional StatusCodeId *Property* provides the exact security error responsible for producing the *Event.*

6.4.5 AuditChannelEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 26.

Table 26 – AuditChannelEventType Definition

Attribute	Value				
BrowseName	AuditChannelEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the AuditSecurityEventType defined in 6.4.4, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditOpenSecureChannelEventType	Defined in 6.4.6		
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSecurityEventType*. Their semantic is defined in 6.4.4. The *SourceNode Property* for *Events* of this type shall be assigned to the *Server Object*. The *SourceName* for *Events* of this type shall be “SecureChannel/” and the *Service* that generates the *Event* (e.g. SecureChannel/*OpenSecureChannel* or SecureChannel/*CloseSecureChannel*). If the *ClientUserId* is not available for a *CloseSecureChannel* call, then this parameter shall be set to ”System/CloseSecureChannel”.

The *SecureChannelId* shall uniquely identify the SecureChannel. The application shall use the same identifier in all *AuditEvents* related to the Session Service Set (AuditCreateSessionEventType, AuditActivateSessionEventType and their subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

6.4.6 AuditOpenSecureChannelEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 27.

Table 27 – AuditOpenSecureChannelEventType Definition

Attribute	Value				
BrowseName	AuditOpenSecureChannelEventType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditChannelEventType* defined in 6.4.5, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ClientCertificate	ByteString	PropertyType	Mandatory
HasProperty	Variable	ClientCertificateThumbprint	String	PropertyType	Mandatory
HasProperty	Variable	RequestType	SecurityTokenRequestType	PropertyType	Mandatory
HasProperty	Variable	SecurityPolicyUri	String	PropertyType	Mandatory
HasProperty	Variable	SecurityMode	MessageSecurityMode	PropertyType	Mandatory
HasProperty	Variable	RequestedLifetime	Duration	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditChannelEventType*. Their semantic is defined in 6.4.5. The *SourceName* for *Events* of this type shall be “SecureChannel/OpenSecureChannel”. The *ClientUserId* is not available for this call, thus this parameter shall be set to ”System/OpenSecureChannel”.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ClientCertificate is the clientCertificate parameter of the OpenSecureChannel *Service* call.

ClientCertificateThumbprint is a thumbprint of the *ClientCertificate.* See Part 6 for details on thumbprints.

RequestType is the requestType parameter of the OpenSecureChannel *Service* call.

SecurityPolicyUri is the securityPolicyUri parameter of the OpenSecureChannel *Service* call.

SecurityMode is the securityMode parameter of the OpenSecureChannel *Service* call.

RequestedLifetime is the requestedLifetime parameter of the OpenSecureChannel *Service* call.

6.4.7 AuditSessionEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 28.

Table 28 – AuditSessionEventType Definition

Attribute	**Value**				
BrowseName	AuditSessionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditSecurityEventType* defined in 6.4.4, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditCreateSessionEventType	Defined in 6.4.8		
HasSubtype	ObjectType	AuditActivateSessionEventType	Defined in 6.4.10		
HasSubtype	ObjectType	AuditCancelEventType	Defined in 6.4.11		
HasProperty	Variable	SessionId	NodeId	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSecurityEventType*. Their semantic is defined in 6.4.4.

If the *Event* is generated by a *TransferSubscriptions Service* call, the *SourceNode Property* shall be assigned to the *SessionDiagnostics Object* that represents the session. The *SourceName* for *Events* of this type shall be “Session/TransferSubscriptions”.

Otherwise, the *SourceNode Property* for *Events* of this type shall be assigned to the *Server Object*. The *SourceName* for *Events* of this type shall be “Session/” and the *Service* or cause that generates the *Event* (e.g. *CreateSession*, *ActivateSession* or *CloseSession*).

The *SessionId* shall contain the *SessionId* of the session that the *Service* call was issued on In the *CreateSession Service* this shall be set to the newly created *SessionId*. If no session context exists (e.g. for a failed *CreateSession Service* call) the *SessionId* shall be null.

6.4.8 AuditCreateSessionEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 29.

Table 29 – AuditCreateSessionEventType Definition

Attribute	**Value**				
BrowseName	AuditCreateSessionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditSessionEventType* defined in 6.4.7, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditUrlMismatchEventType	Defined in 6.4.9		
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory
HasProperty	Variable	ClientCertificate	ByteString	PropertyType	Mandatory
HasProperty	Variable	ClientCertificateThumbprint	String	PropertyType	Mandatory
HasProperty	Variable	RevisedSessionTimeout	Duration	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.7. The *SourceName* for *Events* of this type shall be “Session/CreateSession”. The *ClientUserId* is not available for this call thus this parameter shall be set to the ”System/CreateSession”.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

SecureChannelId shall uniquely identify the SecureChannel. The application shall use the same identifier in all *AuditEvents* related to the Session Service Set (AuditCreateSessionEventType, AuditActivateSessionEventType and their subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

ClientCertificate is the clientCertificate parameter of the CreateSession *Service* call.

ClientCertificateThumbprint is a thumbprint of the *ClientCertificate.* See Part 6 for details on thumbprints.

RevisedSessionTimeout is the returned revisedSessionTimeout parameter of the CreateSession *Service* call.

6.4.9 AuditUrlMismatchEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 30.

Table 30 – AuditUrlMismatchEventType Definition

Attribute	Value				
BrowseName	AuditUrlMismatchEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the AuditCreateSessionEventType defined in 6.4.8 which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	EndpointUrl	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.8.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

EndpointUrl is the endpointUrl parameter of the CreateSession *Service* call.

6.4.10 AuditActivateSessionEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 31.

Table 31 – AuditActivateSessionEventType Definition

Attribute	Value				
BrowseName	AuditActivateSessionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditSessionEventType* defined in 6.4.7, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ClientSoftwareCertificates	SignedSoftwareCertificate[]	PropertyType	Mandatory
HasProperty	Variable	UserIdentityToken	UserIdentityToken	PropertyType	Mandatory
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.7. The *SourceName* for *Events* of this type shall be “Session/ActivateSession”.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ClientSoftwareCertificates is the clientSoftwareCertificates parameter of the ActivateSession *Service* call.

UserIdentityToken reflects the userIdentityToken parameter of the ActivateSession *Service* call. For Username/Password tokens the password shall not be included.

SecureChannelId shall uniquely identify the SecureChannel. The application shall use the same identifier in all *AuditEvents* related to the Session Service Set (AuditCreateSessionEventType, AuditActivateSessionEventType and their subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

6.4.11 AuditCancelEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 32.

Table 32 – AuditCancelEventType Definition

Attribute		**Value**			
BrowseName		AuditCancelEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditSessionEventType* defined in 6.4.7, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	RequestHandle	UInt32	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.7. The *SourceName* for *Events* of this type shall be “Session/Cancel”.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

RequestHandle is the requestHandle parameter of the Cancel *Service* call.

6.4.12 AuditCertificateEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 33.

Table 33 – AuditCertificateEventType Definition

Attribute		**Value**			
BrowseName		AuditCertificateEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditSecurityEventType* defined in 6.4.7, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditCertificateDataMismatchEventType	Defined in 6.4.13		
HasSubtype	ObjectType	AuditCertificateExpiredEventType	Defined in 6.4.14		
HasSubtype	ObjectType	AuditCertificateInvalidEventType	Defined in 6.4.15		
HasSubtype	ObjectType	AuditCertificateUntrustedEventType	Defined in 6.4.16		
HasSubtype	ObjectType	AuditCertificateRevokedEventType	Defined in 6.4.17		
HasSubtype	ObjectType	AuditCertificateMismatchEventType	Defined in 6.4.18		
HasProperty	Variable	Certificate	ByteString	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSecurityEventType*. Their semantic is defined in 6.4.4. The *SourceName* for *Events* of this type shall be “Security/Certificate”.

Certificate is the certificate that encountered a validation issue. Additional subtypes of this *EventType* will be defined representing the individual validation errors. This certificate can be matched to the *Service* that passed it (Session or SecureChannel Service Set) since the *AuditEvents* for these *Services* also included the Certificate.

6.4.13 AuditCertificateDataMismatchEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 34.

Table 34 – AuditCertificateDataMismatchEventType Definition

Attribute		Value			
BrowseName		AuditCertificateDataMismatchEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the AuditCertificateEventType defined in 6.4.12, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	InvalidHostname	String	PropertyType	Mandatory
HasProperty	Variable	InvalidUri	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type shall be “Security/Certificate”.

InvalidHostname is the string that represents the host name passed in as part of the URL that is found to be invalid. If the host name was not invalid it can be null.

InvalidUri is the URI that was passed in and found to not match what is contained in the certificate. If the URI was not invalid it can be null.

Either the *InvalidHostname* or *InvalidUri* shall be provided.

6.4.14 AuditCertificateExpiredEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 35.

Table 35 – AuditCertificateExpiredEventType Definition

Attribute		Value			
BrowseName		AuditCertificateExpiredEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type shall be “Security/Certificate”. The *Message Variable* shall include a description of why the certificate was expired (i.e. time before start or time after end). There are no additional *Properties* defined for this *EventType*.

6.4.15 AuditCertificateInvalidEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 36.

Table 36 – AuditCertificateInvalidEventType Definition

Attribute		Value			
BrowseName		AuditCertificateInvalidEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type shall be “Security/Certificate”. The *Message* shall include a description of why the certificate is invalid. There are no additional *Properties* defined for this *EventType*.

6.4.16 AuditCertificateUntrustedEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 37.

Table 37 – AuditCertificateUntrustedEventType Definition

Attribute		Value			
BrowseName		AuditCertificateUntrustedEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type shall be “Security/Certificate”. The *Message Variable* shall include a description of why the certificate is not trusted. If a trust chain is involved then the certificate that failed in the trust chain should be described. There are no additional *Properties* defined for this *EventType*.

6.4.17 AuditCertificateRevokedEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 38.

Table 38 – AuditCertificateRevokedEventType Definition

Attribute		Value			
BrowseName		AuditCertificateRevokedEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type shall be “Security/Certificate”. The *Message Variable* shall include a description of why the certificate is revoked (was the revocation list unavailable or was the certificate on the list). There are no additional *Properties* defined for this *EventType*.

6.4.18 AuditCertificateMismatchEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 39.

Table 39 – AuditCertificateMismatchEventType Definition

Attribute		Value			
BrowseName		AuditCertificateMismatchEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type shall be “Security/Certificate”. The *Message Variable* shall include a description of misuse of the certificate. There are no additional *Properties* defined for this *EventType*.

6.4.19 AuditNodeManagementEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 40.

Table 40 – AuditNodeManagementEventType Definition

Attribute	Value				
BrowseName	AuditNodeManagementEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditAddNodesEventType			
HasSubtype	ObjectType	AuditDeleteNodesEventType			
HasSubtype	ObjectType	AuditAddReferencesEventType			
HasSubtype	ObjectType	AuditDeleteReferencesEventType			

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. There are no additional *Properties* defined for this *EventType*. The *SourceNode Property* for *Events* of this type shall be assigned to the *Server Object*. The *SourceName* for *Events* of this type shall be "NodeManagement/" and the *Service* that generates the *Event* (e.g. *AddNodes*, *AddReferences*, *DeleteNodes*, *DeleteReferences*).

6.4.20 AuditAddNodesEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 41.

Table 41 – AuditAddNodesEventType Definition

Attribute	Value				
BrowseName	AuditAddNodesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	NodesToAdd	AddNodesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type shall be "NodeManagement/AddNodes".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

NodesToAdd is the NodesToAdd parameter of the AddNodes *Service* call.

6.4.21 AuditDeleteNodesEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 42.

Table 42 – AuditDeleteNodesEventType Definition

Attribute	Value				
BrowseName	AuditDeleteNodesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	NodesToDelete	DeleteNodesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type shall be "NodeManagement/DeleteNodes".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

NodesToDelete is the nodesToDelete parameter of the DeleteNodes *Service* call.

6.4.22 AuditAddReferencesEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 43.

Table 43 – AuditAddReferencesEventType Definition

Attribute	Value				
BrowseName	AuditAddReferencesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ReferencesToAdd	AddReferencesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type shall be "NodeManagement/AddReferences".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ReferencesToAdd is the referencesToAdd parameter of the AddReferences *Service* call.

6.4.23 AuditDeleteReferencesEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 44.

Table 44 – AuditDeleteReferencesEventType Definition

Attribute	Value				
BrowseName	AuditDeleteReferencesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ReferencesToDelete	DeleteReferencesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type shall be "NodeManagement/DeleteReferences".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ReferencesToDelete is the referencesToDelete parameter of the DeleteReferences *Service* call.

6.4.24 AuditUpdateEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 45.

Table 45 – AuditUpdateEventType Definition

Attribute	Value				
BrowseName	AuditUpdateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditWriteUpdateEventType	Defined in 6.4.25		
HasSubtype	ObjectType	AuditHistoryUpdateEventType	Defined in 6.4.26		

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. The *SourceNode Property* for *Events* of this type shall be assigned to the *NodeId* that was

changed. The *SourceName* for *Events* of this type shall be “Attribute/” and the *Service* that generated the event (e.g. *Write*, *HistoryUpdate*). Note that one *Service* call may generate several *Events* of this type, one per changed value.

6.4.25 AuditWriteUpdateEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 46.

Table 46 – AuditWriteUpdateEventType Definition

Attribute	**Value**				
BrowseName	AuditWriteUpdateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateEventType* defined in 6.4.24, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	AttributeId	UInt32	PropertyType	Mandatory
HasProperty	Variable	IndexRange	NumericRange	PropertyType	Mandatory
HasProperty	Variable	NewValue	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	OldValue	BaseDataType	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateEventType*. The *SourceName* for *Events* of this type shall be “Attribute/Write”. Their semantic is defined in 6.4.24.

AttributeId identifies the *Attribute* that was written. The *SourceNode Property* identifies the *Node* that was written.

IndexRange identifies the index range of the written *Attribute* if the *Attribute* is an array. If the *Attribute* is not an array or the whole array was written, the *IndexRange* is set to null.

NewValue identifies the value that was written. If the *IndexRange* is provided, only the values in the provided range are shown.

OldValue identifies the value that the *Attribute* contained before the write. If the *IndexRange* is provided, only the value of that range is shown. It is acceptable for a *Server* that does not have this information to report a null value.

Both the *NewValue* and the *OldValue* will contain a value in the *DataType* and encoding used for writing the value.

6.4.26 AuditHistoryUpdateEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 47.

Table 47 – AuditHistoryUpdateEventType Definition

Attribute	**Value**				
BrowseName	AuditHistoryUpdateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateEventType* defined in 6.4.24, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ParameterDataTypeId	NodeId	PropertyType	New

This *EventType* inherits all *Properties* of the *AuditUpdateEventType*. Their semantic is defined in 6.4.24.

The *ParameterDataTypeId* identifies the *DataTypeId* for the extensible parameter used by the HistoryUpdate. This parameter indicates the type of HistoryUpdate being performed.

Subtypes of this *EventType* are defined in Part 11 representing the different possibilities to manipulate historical data.

6.4.27 AuditUpdateMethodEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 48.

Table 48 – AuditUpdateMethodEventType Definition

Attribute	Value				
BrowseName	AuditUpdateMethodEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	MethodId	NodeId	PropertyType	Mandatory
HasProperty	Variable	InputArguments	BaseDataType[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. The *SourceNode Property* for *Events* of this type shall be assigned to the *NodeId* of the *Object* that the *Method* resides on. The *SourceName* for *Events* of this type shall be “Attribute/Call”. Note that one *Service* call may generate several *Events* of this type, one per method called. This *EventType* should be further subtyped to better reflect the functionality of the method and to reflect changes to the address space or updated values triggered by the method.

MethodId identifies the method that was called.

InputArguments identifies the input Arguments for the method. This parameter can be null if no input arguments where provided.

6.4.28 SystemEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 49.

Table 49 – SystemEventType Definition

Attribute	Value				
BrowseName	SystemEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	ObjectType	DeviceFailureEventType	Defined in 6.4.29		
HasSubtype	ObjectType	SystemStatusChangeEventType	Defined in 6.4.30		
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. There are no additional *Properties* defined for this *EventType*.

6.4.29 DeviceFailureEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 50.

Table 50 – DeviceFailureEventType Definition

Attribute	Value				
BrowseName	DeviceFailureEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *SystemEventType* defined in 6.4.28, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *SystemEventType*. Their semantic is defined in 6.4.28. There are no additional *Properties* defined for this *EventType*.

6.4.30 SystemStatusChangeEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 51.

Table 51 – SystemStatusChangeEventType Definition

Attribute		Value			
BrowseName		SystemStatusChangeEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *SystemEventType* defined in 6.4.28, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	SystemState	ServerState	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *SystemEventType*. Their semantic is defined in 6.4.28. The *SourceNode Property* and the *SourceName* shall identify the system. The system can be the *Server* itself or some underlying system.

The *SystemState* specifies the current state of the system. Changes to the *ServerState* of the system shall trigger a *SystemStatusChangeEvent*, when the event is supported by the system.

6.4.31 BaseModelChangeEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 52.

Table 52 – BaseModelChangeEventType Definition

Attribute	Value				
BrowseName	BaseModelChangeEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	GeneralModelChangeEventType	Defined in 6.4.32		

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. There are no additional *Properties* defined for this *EventType*. The *SourceNode Property* for Events of this type shall be the *Node* of the *View* that gives the context of the changes. If the whole *AddressSpace* is the context, the *SourceNode Property* is set to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type shall be the *String* part of the *BrowseName* of the *View*; for the whole *AddressSpace* it shall be “Server”.

6.4.32 GeneralModelChangeEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 53.

Table 53 – GeneralModelChangeEventType Definition

Attribute	Value				
BrowseName	GeneralModelChangeEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseModelChangeEventType* defined in 6.4.31, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	Changes	ModelChangeStructureDataType[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseModelChangeEventType*. Their semantic is defined in 6.4.31.

The additional *Property* defined for this *EventType* reflects the changes that issued the *ModelChangeEvent*. It shall contain at least one entry in its array. Its structure is defined in 12.16.

6.4.33 SemanticChangeEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 54.

Table 54 – SemanticChangeEventType Definition

Attribute	Value				
BrowseName	SemanticChangeEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	Changes	SemanticChangeStructureDataType[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. There are no additional *Properties* defined for this *EventType*. The *SourceNode Property* for Events of this type shall be the *Node* of the *View* that gives the context of the changes. If the whole *AddressSpace* is the context, the *SourceNode Property* is set to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type shall be the *String* part of the *BrowseName* of the *View*, for the whole *AddressSpace* it shall be “Server”.

The additional *Property* defined for this *EventType* reflects the changes that issued the *SemanticChangeEvent*. Its structure is defined in 12.17.

6.4.34 EventQueueOverflowEventType

EventQueueOverflow Events are generated when an internal queue of a MonitoredItem subscribing for *Events* in the *Server* overflows. Part 4 defines when the internal EventQueueOverflow *Events* shall be generated.

The *EventType* for *EventQueueOverflow Events* is formally defined in Table 55.

Table 55 – EventQueueOverflowEventType Definition

Attribute	Value				
BrowseName	EventQueueOverflowEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. The *SourceNode Property* for *Events* of this type shall be assigned to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type shall be “Internal/EventQueueOverflow”.

6.4.35 ProgressEventType

ProgressEvents are generated to identify the progress of an operation. An operation can be a *Service* call or something application specific like a program execution.

The *EventType* for *Progress Events* is formally defined in Table 56.

Table 56 – ProgressEventType Definition

Attribute	Value				
BrowseName	ProgressEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	Context	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Progress	UInt16	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. The *SourceNode Property* for *Events* of this type shall be assigned to the *NodeId* of the *Session Object* where the operation was initiated. The *SourceName* for *Events* of this type shall be “Service/<Service Name as defined in Part 4>” when the progress of a *Service* call is exposed.

The additional *Property Context* contains context information about what operation progress is reported. In the case of *Service* calls it shall be a UInt32 containing the *requestHandle* of the *RequestHeader* of the *Service* call.

The additional *Property Progress* contains the percentage completed of the progress. The value shall be between 0 and 100, where 100 identifies that the operation has been finished.

It is recommended that *Servers* only expose *ProgressEvents* for *Service* calls to the *Session* that invoked the *Service*.

6.5 ModellingRuleType

ModellingRules are defined in Part 3. This *ObjectType* is used as the type for the *ModellingRules*. It is formally defined in Table 57.

Table 57 – ModellingRuleType Definition

Attribute		Value			
BrowseName		ModellingRuleType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	NamingRule	NamingRuleType	PropertyType	Mandatory

The *Property NamingRule* identifies the *NamingRule* of a *ModellingRule* as defined in Part 3.

6.6 FolderType

Instances of this *ObjectType* are used to organise the *AddressSpace* into a hierarchy of *Nodes*. They represent the root *Node* of a subtree, and have no other semantics associated with them. However, the *DisplayName* of an instance of the *FolderType*, such as “ObjectTypes”, should imply the semantics associated with the use of it. There are no References specified for this *ObjectType*. It is formally defined in Table 58.

Table 58 – FolderType Definition

Attribute		Value			
BrowseName		FolderType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

6.7 DataTypeEncodingType

DataTypeEncodings are defined in Part 3. This *ObjectType* is used as type for the *DataTypeEncodings*. There are no *References* specified for this *ObjectType*. It is formally defined in Table 59.

Table 59 – DataTypeEncodingType Definition

Attribute		Value			
BrowseName		DataTypeEncodingType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

6.8 AggregateFunctionType

This *ObjectType* defines an *AggregateFunction* supported by a UA *Server*. It is formally defined in Table 60.

Table 60 – AggregateFunctionType Definition

Attribute		Value			
BrowseName		AggregateFunctionType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

For the *AggregateFunctionType*, the *Description Attribute* is mandatory. The *Description Attribute* provides a localized description of the *AggregateFunction.* Specific *AggregateFunctions* may be defined in further parts of this series of standards.

7 Standard VariableTypes

7.1 General

Typically, the components of a complex *VariableType* are fixed and can be extended by subtyping. However, because each *Variable* of a *VariableType* can be extended with additional components this standard allows the extension of the standard *VariableTypes* defined in this document with additional components. This allows the expression of additional information in the type definition that would be contained in each *Variable* anyway. However, it is not allowed to restrict the components of the standard *VariableTypes* defined in this International Standard. An example of extending *VariableTypes* would be putting the standard *Property NodeVersion*, defined in Part 3, into the *BaseDataVariableType*, stating that each *DataVariable* of the *Server* will provide a *NodeVersion*.

7.2 BaseVariableType

The *BaseVariableType* is the abstract base type for all other *VariableTypes*. However, only the *PropertyType* and the *BaseDataVariableType* directly inherit from this type.

There are no *References*, except for *HasSubtype References*, specified for this *VariableType*. It is formally defined in Table 61.

Table 61 – BaseVariableType Definition

Attribute		Value			
BrowseName		BaseVariableType			
IsAbstract		True			
ValueRank		−2 (−2 = Any)			
DataType		BaseDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	VariableType	PropertyType	Defined in 7.3		
HasSubtype	VariableType	BaseDataVariableType	Defined in 7.4		

7.3 PropertyType

The *PropertyType* is a subtype of the *BaseVariableType*. It is used as the type definition for all *Properties*. *Properties* are defined by their *BrowseName* and therefore they do not need a specialised type definition. It is not allowed to subtype this *VariableType*.

There are no *References* specified for this *VariableType*. It is formally defined in Table 62.

Table 62 – PropertyType Definition

Attribute		Value			
BrowseName		PropertyType			
IsAbstract		False			
ValueRank		−2 (−2 = Any)			
DataType		BaseDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseVariableType defined in 7.2.					

7.4 BaseDataVariableType

The *BaseDataVariableType* is a subtype of the *BaseVariableType*. It is used as the type definition whenever there is a *DataVariable* having no more concrete type definition available. This *VariableType* is the base *VariableType* for *VariableTypes* of *DataVariables*, and all other *VariableTypes* of *DataVariables* shall either directly or indirectly inherit from it. However, it might not be possible for *Servers* to provide all *HasSubtype References* from this *VariableType* to its subtypes, and therefore it is not required to provide this information.

There are no *References* except for *HasSubtype References* specified for this *VariableType*. It is formally defined in Table 63.

Table 63 – BaseDataVariableType Definition

Attribute		Value	
BrowseName		BaseDataVariableType	
IsAbstract		False	
ValueRank		−2 (−2 = Any)	
DataType		BaseDataType	
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of the BaseVariableType defined in 7.2.			
HasSubtype	VariableType	ServerVendorCapabilityType	Defined in 7.5
HasSubtype	VariableType	ServerStatusType	Defined in 7.6
HasSubtype	VariableType	BuildInfoType	Defined in 7.7
HasSubtype	VariableType	ServerDiagnosticsSummaryType	Defined in 7.8
HasSubtype	VariableType	SamplingIntervalDiagnosticsArrayType	Defined in 7.9
HasSubtype	VariableType	SamplingIntervalDiagnosticsType	Defined in 7.10
HasSubtype	VariableType	SubscriptionDiagnosticsArrayType	Defined in 7.11
HasSubtype	VariableType	SubscriptionDiagnosticsType	Defined in 7.12
HasSubtype	VariableType	SessionDiagnosticsArrayType	Defined in 7.13
HasSubtype	VariableType	SessionDiagnosticsVariableType	Defined in 7.14
HasSubtype	VariableType	SessionSecurityDiagnosticsArrayType	Defined in 7.15
HasSubtype	VariableType	SessionSecurityDiagnosticsType	Defined in 7.16
HasSubtype	VariableType	OptionSetType	Defined in 7.17

7.5 ServerVendorCapabilityType

This *VariableType* is an abstract type whose subtypes define capabilities of the *Server*. Vendors may define subtypes of this type. This *VariableType* is formally defined in Table 64.

Table 64 – ServerVendorCapabilityType Definition

Attribute		Value			
BrowseName		ServerVendorCapabilityType			
IsAbstract		True			
ValueRank		−1 (−1 = Scalar)			
DataType		BaseDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					

7.6 ServerStatusType

This complex *VariableType* is used for information about the *Server* status. Its *DataVariables* reflect its *DataType* having the same semantic defined in 12.10. The *VariableType* is formally defined in Table 65.

Table 65 – ServerStatusType Definition

Attribute		Value			
BrowseName		ServerStatusType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		ServerStatusDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	StartTime	UtcTime	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentTime	UtcTime	BaseDataVariableType	Mandatory
HasComponent	Variable	State	ServerState	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildInfo[1]	BuildInfo	BuildInfoType	Mandatory
HasComponent	Variable	SecondsTillShutdown	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	ShutdownReason	LocalizedText	BaseDataVariableType	Mandatory
NOTE Containing *Objects* and *Variables* of these *Objects* and *Variables* are defined by their *BrowseName* defined in the corresponding *TypeDefinitionNode*. The *NodeId* is defined by the composed symbolic name described in 4.1.					

7.7 BuildInfoType

This complex *VariableType* is used for information about the *Server* status. Its *DataVariables* reflect its *DataType* having the same semantic defined in 12.4. The *VariableType* is formally defined in Table 66.

Table 66 – BuildInfoType Definition

Attribute		Value			
BrowseName		BuildInfoType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		BuildInfo			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	ProductUri	String	BaseDataVariableType	Mandatory
HasComponent	Variable	ManufacturerName	String	BaseDataVariableType	Mandatory
HasComponent	Variable	ProductName	String	BaseDataVariableType	Mandatory
HasComponent	Variable	SoftwareVersion	String	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildNumber	String	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildDate	UtcTime	BaseDataVariableType	Mandatory

7.8 ServerDiagnosticsSummaryType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType* having the same semantic defined in 12.9. The *VariableType* is formally defined in Table 67.

Table 67 – ServerDiagnosticsSummaryType Definition

Attribute		Value			
BrowseName		ServerDiagnosticsSummaryType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		ServerDiagnosticsSummaryDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	ServerViewCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CumulatedSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityRejectedSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RejectedSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SessionTimeoutCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SessionAbortCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishingIntervalCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentSubscriptionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CumulatedSubscriptionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityRejectedRequestsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RejectedRequestsCount	UInt32	BaseDataVariableType	Mandatory

7.9 SamplingIntervalDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array, instances of this type will provide a *Variable* of the SamplingIntervalDiagnosticsType *VariableType* having the sampling rate as *BrowseName*. The *VariableType* is formally defined in Table 68.

Table 68 – SamplingIntervalDiagnosticsArrayType Definition

Attribute		Value		
BrowseName		SamplingIntervalDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SamplingIntervalDiagnosticsDataType		
References	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SamplingIntervalDiagnostics	SamplingIntervalDiagnosticsDataType SamplingIntervalDiagnosticsType	ExposesItsArray

7.10 SamplingIntervalDiagnosticsType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.8. The *VariableType* is formally defined in Table 69.

Table 69 – SamplingIntervalDiagnosticsType Definition

Attribute			Value		
BrowseName			SamplingIntervalDiagnosticsType		
IsAbstract			False		
ValueRank			−1 (−1 = Scalar)		
DataType			SamplingIntervalDiagnosticsDataType		
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	SamplingInterval	Duration	BaseDataVariableType	Mandatory
HasComponent	Variable	SampledMonitoredItemsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxSampledMonitoredItemsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DisabledMonitoredItemsSamplingCount	UInt32	BaseDataVariableType	Mandatory

7.11 SubscriptionDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array, instances of this type will provide a *Variable* of the SubscriptionDiagnosticsType *VariableType* having the SubscriptionId as *BrowseName*. The *VariableType* is formally defined in Table 70.

Table 70 – SubscriptionDiagnosticsArrayType Definition

Attribute		Value		
BrowseName		SubscriptionDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SubscriptionDiagnosticsDataType		
References	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SubscriptionDiagnostics	SubscriptionDiagnosticsDataType SubscriptionDiagnosticsType	ExposesItsArray

7.12 SubscriptionDiagnosticsType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.15. The *VariableType* is formally defined in Table 71.

Table 71 – SubscriptionDiagnosticsType Definition

Attribute	Value				
BrowseName	SubscriptionDiagnosticsType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	SubscriptionDiagnosticsDataType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	SessionId	NodeId	BaseDataVariableType	Mandatory
HasComponent	Variable	SubscriptionId	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	Priority	Byte	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishingInterval	Duration	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxKeepAliveCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxLifetimeCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxNotificationsPerPublish	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishingEnabled	Boolean	BaseDataVariableType	Mandatory
HasComponent	Variable	ModifyCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	EnableCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DisableCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishMessageRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishMessageCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	TransferRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	TransferredToAltClientCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	TransferredToSameClientCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DataChangeNotificationsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	EventNotificationsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	NotificationsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	LatePublishRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentKeepAliveCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentLifetimeCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	UnacknowledgedMessageCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DiscardedMessageCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MonitoredItemCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DisabledMonitoredItemCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MonitoringQueueOverflowCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	NextSequenceNumber	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	EventQueueOverflowCount	UInt32	BaseDataVariableType	Mandatory

7.13 SessionDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array instances of this type will provide a *Variable* of the SessionDiagnosticsVariableType *VariableType*, having the SessionDiagnostics as *BrowseName*. Those *Variables* will also be referenced by the SessionDiagnostics *Objects* defined by their type in 6.3.5. The *VariableType* is formally defined in Table 72.

Table 72 – SessionDiagnosticsArrayType Definition

Attribute	Value			
BrowseName	SessionDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionDiagnosticsDataType			
References	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	ExposesItsArray

7.14 SessionDiagnosticsVariableType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.11. The *VariableType* is formally defined in Table 73.

Table 73 – SessionDiagnosticsVariableType Definition

Attribute	Value			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
References	**Node Class**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SessionId	NodeId BaseDataVariableType	Mandatory
HasComponent	Variable	SessionName	String BaseDataVariableType	Mandatory
HasComponent	Variable	ClientDescription	ApplicationDescription BaseDataVariableType	Mandatory
HasComponent	Variable	ServerUri	String BaseDataVariableType	Mandatory
HasComponent	Variable	EndpointUrl	String BaseDataVariableType	Mandatory
HasComponent	Variable	LocaleIds	LocaleId[] BaseDataVariableType	Mandatory
HasComponent	Variable	MaxResponseMessageSize	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	ActualSessionTimeout	Duration BaseDataVariableType	Mandatory
HasComponent	Variable	ClientConnectionTime	UtcTime BaseDataVariableType	Mandatory
HasComponent	Variable	ClientLastContactTime	UtcTime BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentSubscriptionsCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentMonitoredItemsCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentPublishRequestsInQueue	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	TotalRequestCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	UnauthorizedRequestCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	ReadCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	HistoryReadCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	WriteCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	HistoryUpdateCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	CallCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	CreateMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	ModifyMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	SetMonitoringModeCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	SetTriggeringCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	CreateSubscriptionCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	ModifySubscriptionCount	ServiceCounterDataType	Mandatory

			BaseDataVariableType	
HasComponent	Variable	SetPublishingModeCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	PublishCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	TransferSubscriptionsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteSubscriptionsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	AddNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	AddReferencesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteReferencesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	BrowseCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	BrowseNextCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	TranslateBrowsePathsToNodeIdsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	QueryFirstCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	QueryNextCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	RegisterNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	UnregisterNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory

7.15 SessionSecurityDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array instances of this type will provide a *Variable* of the SessionSecurityDiagnosticsType *VariableType*, having the SessionSecurityDiagnostics as *BrowseName*. Those *Variables* will also be referenced by the SessionDiagnostics *Objects* defined by their type in 6.3.5. The *VariableType* is formally defined in Table 74. Since this information is security related, it should not be made accessible to all users, but only to authorised users.

Table 74 – SessionSecurityDiagnosticsArrayType Definition

Attribute	**Value**			
BrowseName	SessionSecurityDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionSecurityDiagnosticsDataType			
References	**Node Class**	**Browse Name**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	ExposesItsArray

7.16 SessionSecurityDiagnosticsType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.12. The *VariableType* is formally defined in Table 75. Since this information is security-related, it should not be made accessible to all users, but only to authorised users.

Table 75 – SessionSecurityDiagnosticsType Definition

Attribute	Value			
BrowseName	SessionSecurityDiagnosticsType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionSecurityDiagnosticsDataType			
References	**Node Class**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4				
HasComponent	Variable	SessionId	NodeId BaseDataVariableType	Mandatory
HasComponent	Variable	ClientUserIdOfSession	String BaseDataVariableType	Mandatory
HasComponent	Variable	ClientUserIdHistory	String[] BaseDataVariableType	Mandatory
HasComponent	Variable	AuthenticationMechanism	String BaseDataVariableType	Mandatory
HasComponent	Variable	Encoding	String BaseDataVariableType	Mandatory
HasComponent	Variable	TransportProtocol	String BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityMode	MessageSecurityMode BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityPolicyUri	String BaseDataVariableType	Mandatory
HasComponent	Variable	ClientCertificate	ByteString BaseDataVariableType	Mandatory

7.17 OptionSetType

The *OptionSetType VariableType* is used to represent a bit mask. Each array element of the *OptionSetValues Property* contains either the human-readable representation for the corresponding bit used in the option set or an empty *LocalizedText* for a bit that has no specific meaning. The order of the bits of the bit mask maps to a position of the array, i.e. the first bit (least significant bit) maps to the first entry in the array, etc.

In addition to this *VariableType*, the *DataType OptionSet* can alternatively be used to represent a bit mask. As a guideline the *DataType* would be used when the bit mask is fixed and applies to several *Variables.* The *VariableType* would be used when the bit mask is specific for only that *Variable*.

The *DataType* of this *VariableType* shall be capable of representing a bit mask. It shall be either a numeric *DataType* representing a signed or unsigned integer, or a *ByteString*. For example, it can be the *BitFieldMaskDataType.*

The optional BitMask *Property* provides the bit mask in an array of Booleans. This allows subscribing to individual entries of the bit mask. The order of the bits of the bit mask points to a position of the array, i.e. the first bit points to the first entry in the array, etc. The *VariableType* is formally defined in Table 74.

Table 76 – OptionSetType Definition

Attribute	Value			
BrowseName	OptionSetType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	BaseDataType			
References	**NodeClass**	**Browse Name**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4				
HasProperty	Variable	OptionSetValues	LocalizedText[] PropertyType	Mandatory
HasProperty	Variable	BitMask	Boolean[] PropertyType	Optional

7.18 SelectionListType

The *SelectionListType VariableType* is used for a *Variable* where the possible values are provided by a set of values.

The *Selections Property* contains an array of values which represent valid values for this *VariableType's* value.

The *DataType* of the *Selections Property* array shall be of the same *DataType* as this *VariableType.*

Each array element of the optional *SelectionDescriptions Property* contains a human-readable representation of the corresponding value in the *Selections Property* and shall be of the same array size as the *Selections Property*.

The value of this *VariableType* may be restricted to only the values defined in the *Selections Property* by setting the optional *RestrictToList Property* to a value of *True*. If the *RestrictToList Property* is not present or has a value of *False* then the value is not restricted to the set defined by the *Selections Property*.

The *VariableType* is formally defined in Table 77.

Table 77 – SelectionListType Definition

Attribute	**Value**			
BrowseName	SelectionListType			
IsAbstract	False			
ValueRank	−2 (−2 = Any)			
DataType	BaseDataType			
References	**NodeClass**	**Browse Name**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4				
HasProperty	Variable	Selections	BaseDataType[] PropertyType	Mandatory
HasProperty	Variable	SelectionDescriptions	LocalizedText[] PropertyType	Optional
HasProperty	Variable	RestrictToList	Boolean PropertyType	Optional

7.19 AudioVariableType

The *AudioVariableType VariableType* defines a Multipurpose Internet Mail Extensions (MIME) media type of the AudibleSound *Property*. This standard recommends use of text code defined in IETF RFC 2045, IETF RFC 2046 and IETF RFC 2047 for MIME types. The *AudioVariableType* references the Content-Type that is defined as part of the MIME type and commonly used as a reference to a specific MIME. The top-level media type is used to declare the general type of data, while the subtype specifies a specific format for that type of data. Thus, a media type of "audio /xyz" is a sufficient description for a user agent to determine the data is an audio file, even if the user agent has no knowledge of the specific audio format "xyz".

The *VariableType* is formally defined in Table 78.

Table 78 – AudioVariableType Definition

Attribute	Value				
BrowseName	AudioVariableType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ByteString				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4					
HasProperty	Variable	ListId	String	PropertyType	Optional
HasProperty	Variable	AgencyId	String	PropertyType	Optional
HasProperty	Variable	VersionId	String	PropertyType	Optional

8 Standard Objects and their Variables

8.1 General

Objects and *Variables* described in the following subclauses can be extended by additional *Properties* or *References* to other *Nodes*, except where it is stated in the text that it is restricted.

8.2 Objects used to organise the AddressSpace structure

8.2.1 Overview

To promote interoperability of clients and *Servers*, the OPC UA *AddressSpace* is structured as a hierarchy, with the top levels standardised for all *Servers*. Figure 1 illustrates the structure of the *AddressSpace*. All *Objects* in this figure are organised using *Organizes References* and have the *ObjectType FolderType* as type definition.

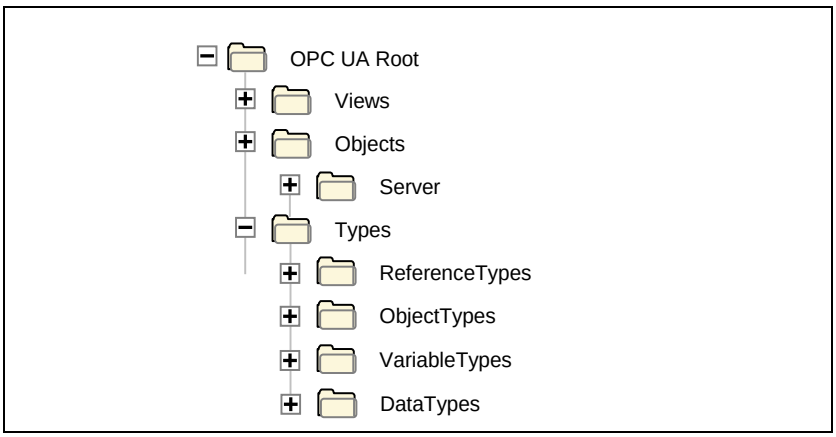

Figure 1 – Standard AddressSpace Structure

The remainder of this provides descriptions of these standard *Nodes* and the organization of *Nodes* beneath them. *Servers* typically implement a subset of these standard *Nodes*, depending on their capabilities.

8.2.2 Root

This standard *Object* is the browse entry point for the *AddressSpace*. It contains a set of *Organizes References* that point to the other standard *Objects*. The "*Root*" *Object* shall not reference any other *NodeClasses*. It is formally defined in Table 79.

Table 79 – Root Definition

Attribute	Value		
BrowseName	Root		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	Views	Defined in 8.2.3
Organizes	Object	Objects	Defined in 8.2.4
Organizes	Object	Types	Defined in 8.2.5

8.2.3 Views

This standard *Object* is the browse entry point for *Views*. Only *Organizes References* are used to relate *View Nodes* to the "*Views*" standard *Object*. All *View Nodes* in the *AddressSpace* shall be referenced by this *Node*, either directly or indirectly. That is, the "*Views*" *Object* may reference other *Objects* using *Organizes References.* Those *Objects* may reference additional *Views*. Figure 2 illustrates the Views Organization. The "*Views"* standard *Object* directly references the *Views* "View1" and "View2" and indirectly "View3" by referencing another *Object* called "Engineering".

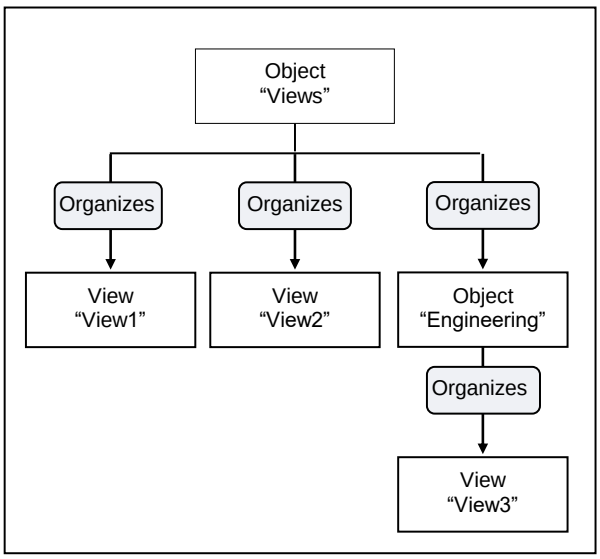

Figure 2 – Views Organization

The "*Views*" *Object* shall not reference any other *NodeClasses*. The "*Views*" *Object* is formally defined in Table 80.

Table 80 – Views Definition

Attribute	Value		
BrowseName	Views		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6

8.2.4 Objects

This standard *Object* is the browse entry point for *Object Nodes*. Figure 3 illustrates the structure beneath this *Node*. Only *Organizes References* are used to relate *Objects* to the "*Objects*" standard *Object*. A *View Node* can be used as entry point into a subset of the *AddressSpace* containing *Objects* and *Variables* and thus the "*Objects*" *Object* can also reference *View Nodes* using *Organizes References*. The intent of the "*Objects*" *Object* is that all *Objects* and *Variables* that are not used for type definitions or other organizational purposes (e.g. organizing the *Views*) are accessible through *hierarchical References* starting from this *Node*. However, this is not a requirement, because not all *Servers* may be able to support this. This *Object* references the standard *Server Object* defined in 8.3.2.

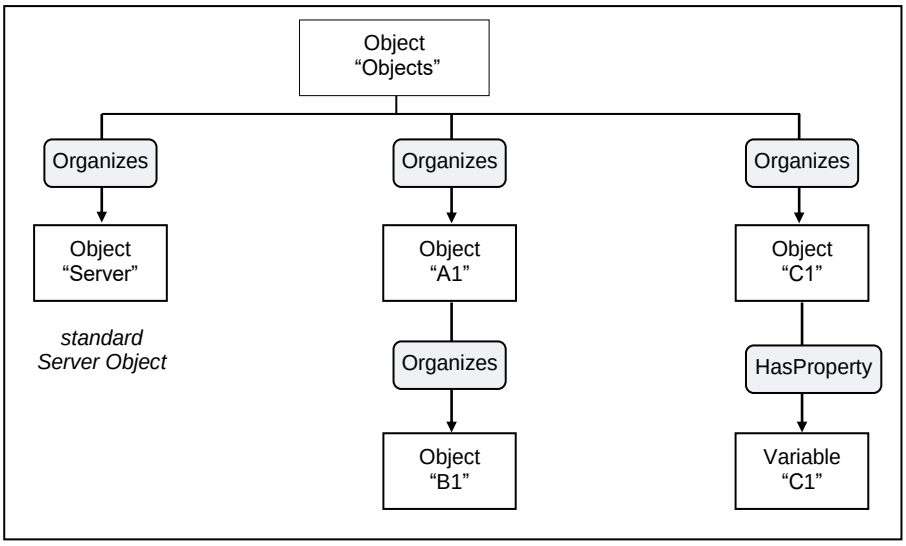

Figure 3 – Objects Organization

The "*Objects*" *Object* shall not reference any other *NodeClasses*. The "*Objects*" *Object* is formally defined in Table 81.

Table 81 – Objects Definition

Attribute	**Value**		
BrowseName	Objects		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	Server	Defined in 8.3.2

8.2.5 Types

This standard *Object Node* is the browse entry point for type *Nodes*. Figure 1 illustrates the structure beneath this *Node*. Only *Organizes References* are used to relate *Objects* to the "*Types*" standard *Object*. The "*Types*" *Object* shall not reference any other *NodeClasses*. It is formally defined in Table 82.

Table 82 – Types Definition

Attribute	**Value**		
BrowseName	Types		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	ObjectTypes	Defined in 8.2.6
Organizes	Object	VariableTypes	Defined in 8.2.7
Organizes	Object	ReferenceTypes	Defined in 8.2.8
Organizes	Object	DataTypes	Defined in 8.2.9
Organizes	Object	EventTypes	Defined in 8.2.10

8.2.6 ObjectTypes

This standard *Object Node* is the browse entry point for *ObjectType Nodes*. Figure 4 illustrates the structure beneath this *Node* showing some of the standard *ObjectTypes* defined in 6. Only *Organizes References* are used to relate *Objects* and *ObjectTypes* to the "*ObjectTypes*" standard *Object*. The "*ObjectTypes*" *Object* shall not reference any other *NodeClasses*.

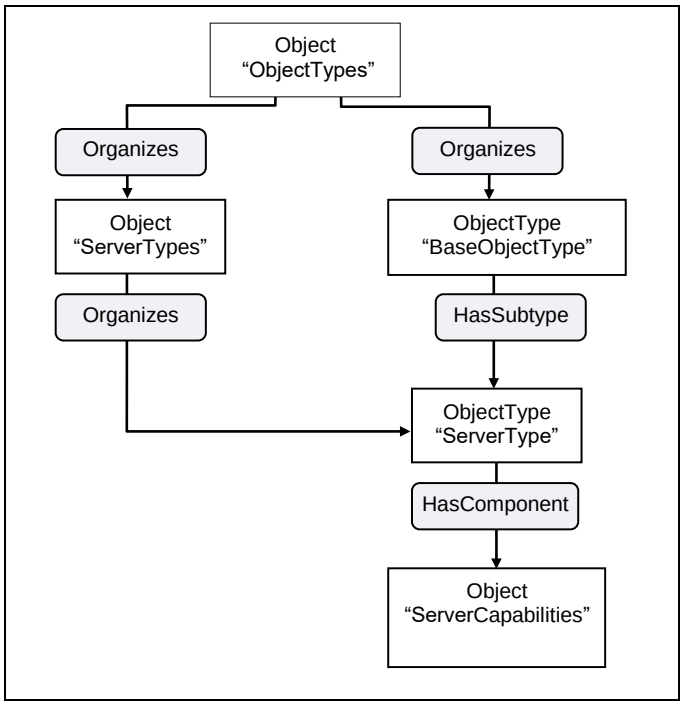

Figure 4 – ObjectTypes Organization

The intention of the “*ObjectTypes*” *Object* is that all *ObjectTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. However, this is not required and *Servers* might not provide some of their *ObjectTypes* because they may be well-known in the industry, such as the *ServerType* defined in 6.3.1.

This *Object* also indirectly references the *BaseEventType* defined in 6.4.2, which is the base type of all *EventTypes*. Thereby it is the entry point for all *EventTypes* provided by the *Server*. It is required that the *Server* expose all its *EventTypes*, so a client can usefully subscribe to *Events*.

The “*ObjectTypes*” *Object* is formally defined in Table 83.

Table 83 – ObjectTypes Definition

Attribute	**Value**		
BrowseName	ObjectTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ObjectType	BaseObjectType	Defined in 6.2

8.2.7 VariableTypes

This standard *Object* is the browse entry point for *VariableType Nodes*. Figure 5 illustrates the structure beneath this *Node*. Only *Organizes References* are used to relate *Objects* and *VariableTypes* to the “*VariableTypes*” standard *Object*. The “*VariableTypes*” *Object* shall not reference any other *NodeClasses*.

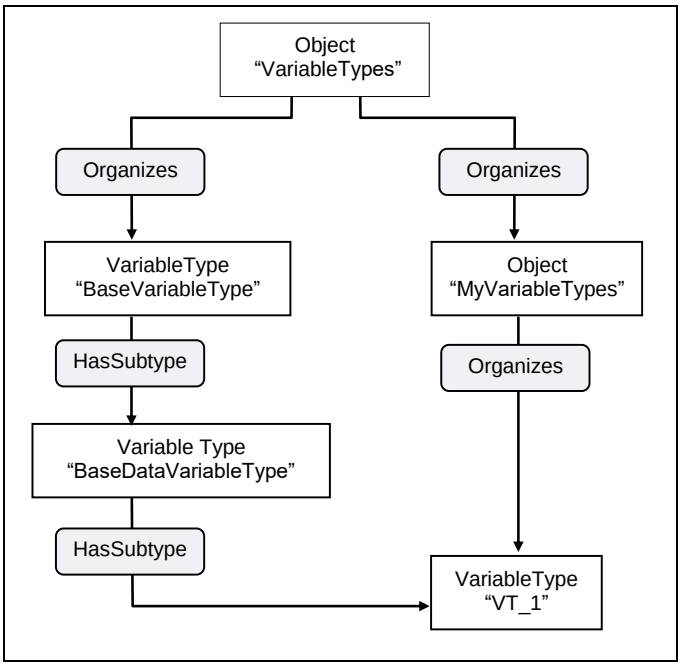

Figure 5 – VariableTypes Organization

The intent of the "*VariableTypes*" *Object* is that all *VariableTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. However, this is not required and *Servers* might not provide some of their *VariableTypes*, because they may be well-known in the industry, such as the "*BaseVariableType*" defined in 7.2.

The "*VariableTypes*" *Object* is formally defined in Table 84.

Table 84 – VariableTypes Definition

Attribute	**Value**		
BrowseName	VariableTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	VariableType	BaseVariableType	Defined in 7.2

8.2.8 ReferenceTypes

This standard *Object* is the browse entry point for *ReferenceType Nodes*. Figure 6 illustrates the organization of *ReferenceTypes*. *Organizes References* are used to define *ReferenceTypes* and *Objects* referenced by the "*ReferenceTypes*" *Object*. The "*ReferenceTypes*" *Object* shall not reference any other *NodeClasses*. See Clause 11 for a discussion of the standard *ReferenceTypes* that appear beneath the "*ReferenceTypes*" *Object*.

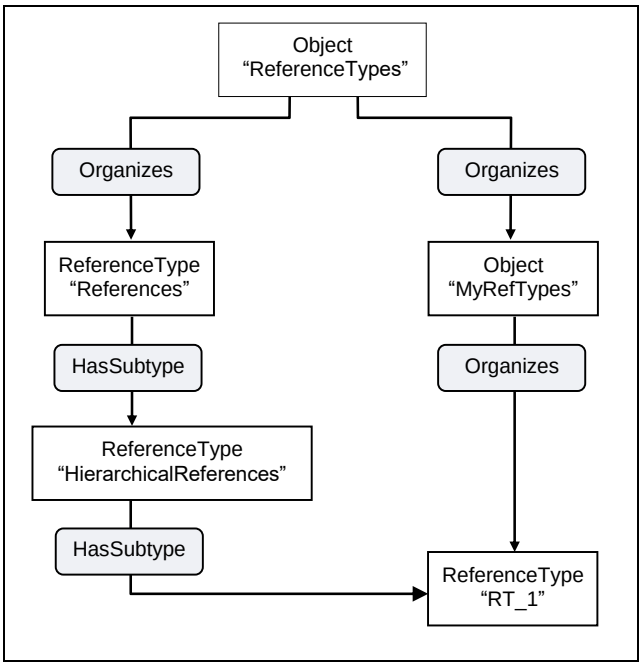

Figure 6 – ReferenceType Definitions

Since *ReferenceTypes* will be used as filters in the browse *Service* and in queries, the *Server* shall provide all its *ReferenceTypes*, directly or indirectly following *hierarchical References* starting from the “*ReferenceTypes*” *Object*. This means that, whenever the client follows a *Reference*, the *Server* shall expose the type of this Reference in the *ReferenceType* hierarchy. It shall provide all *ReferenceTypes* so that the client would be able, following the inverse subtype of *References*, to come to the base *References ReferenceType*. It does not mean that the *Server* shall expose the *ReferenceTypes* that the client has not used any *Reference* of.

The “*ReferenceTypes*” *Object* is formally defined in Table 85.

Table 85 – ReferenceTypes Definition

Attribute	**Value**		
BrowseName	ReferenceTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ReferenceType	References	Defined in 11.1

8.2.9 DataTypes

This standard *Object* is the browse entry point for *DataTypes* that the *Server* wishes to expose in the *AddressSpace*.

DataType Nodes should be made available using *Organizes References* pointing either directly from the “DataTypes” *Object* to the *DataType Nodes* or using additional *Folder Objects* for grouping purposes. The intent is that all *DataTypes* of the *Server* exposed in the *AddressSpace* are accessible following *hierarchical References* starting from the “DataTypes” *Object*. However, this is not required.

The “*DataTypes*” *Object* is formally defined in Table 86.

Table 86 – DataTypes Definition

Attribute	**Value**		
BrowseName	DataTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	DataType	BaseDataType	Defined in 12.2

8.2.10 EventTypes

This standard *Object Node* is the browse entry point for *EventType Nodes*. Figure 7 illustrates the structure beneath this *Node* showing some of the standard *EventTypes* defined in Clause 6. Only *Organizes References* are used to relate *Objects* and *ObjectTypes* to the "*EventTypes*" standard *Object*. The "*EventTypes*" *Object* shall not reference any other *NodeClasses*.

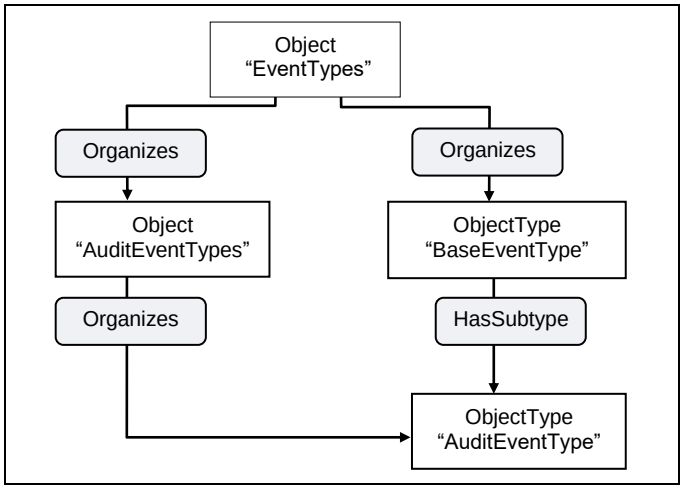

Figure 7 – EventTypes Organization

The intention of the "*EventTypes*" *Object* is that all *EventTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. It is required that the *Server* expose all its *EventTypes*, so a client can usefully subscribe to *Events*.

The "*EventTypes*" *Object* is formally defined in Table 87.

Table 87 – EventTypes Definition

Attribute	**Value**		
BrowseName	ObjectTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ObjectType	BaseEventType	Defined in 6.4.2

8.3 Server Object and its containing Objects

8.3.1 General

The *Server Object* and its containing *Objects* and *Variables* are built in a way that the information can be gained in several ways, suitable for different kinds of clients having different requirements. Annex A gives an overview of the design decisions made in providing the information in that way, and discusses the pros and cons of the different approaches. Figure 8 gives an overview of the containing *Objects* and *Variables* of the diagnostic information of the Server *Object* and where the information can be found.

The SessionsDiagnosticsSummary *Object* contains one *Object* per session and a *Variable* with an array with one entry per session. This array is of a complex *DataType* holding the diagnostic information about the session. Each *Object* representing a session references a complex *Variable* containing the information about the session using the same DataType as the array containing information about all sessions. Such a *Variable* also exposes all its information as *Variables* with simple *DataTypes* containing the same information as in the complex *DataType*. Not shown in Figure 8 is the security-related information per session, which follows the same rules.

The *Server* provides an array with an entry per subscription containing diagnostic information about this subscription. Each entry of this array is also exposed as a complex *Variable* with *Variables* for each individual value. Each *Object* representing a session also provides such an array, but providing the subscriptions of the session.

The arrays containing information about the sessions or the subscriptions may be of different length for different connections with different user credentials since not all users may see all entries of the array. That also implies that the length of the array may change if the user is impersonated. Therefore clients that subscribe to a specific index range may get unexpected results.

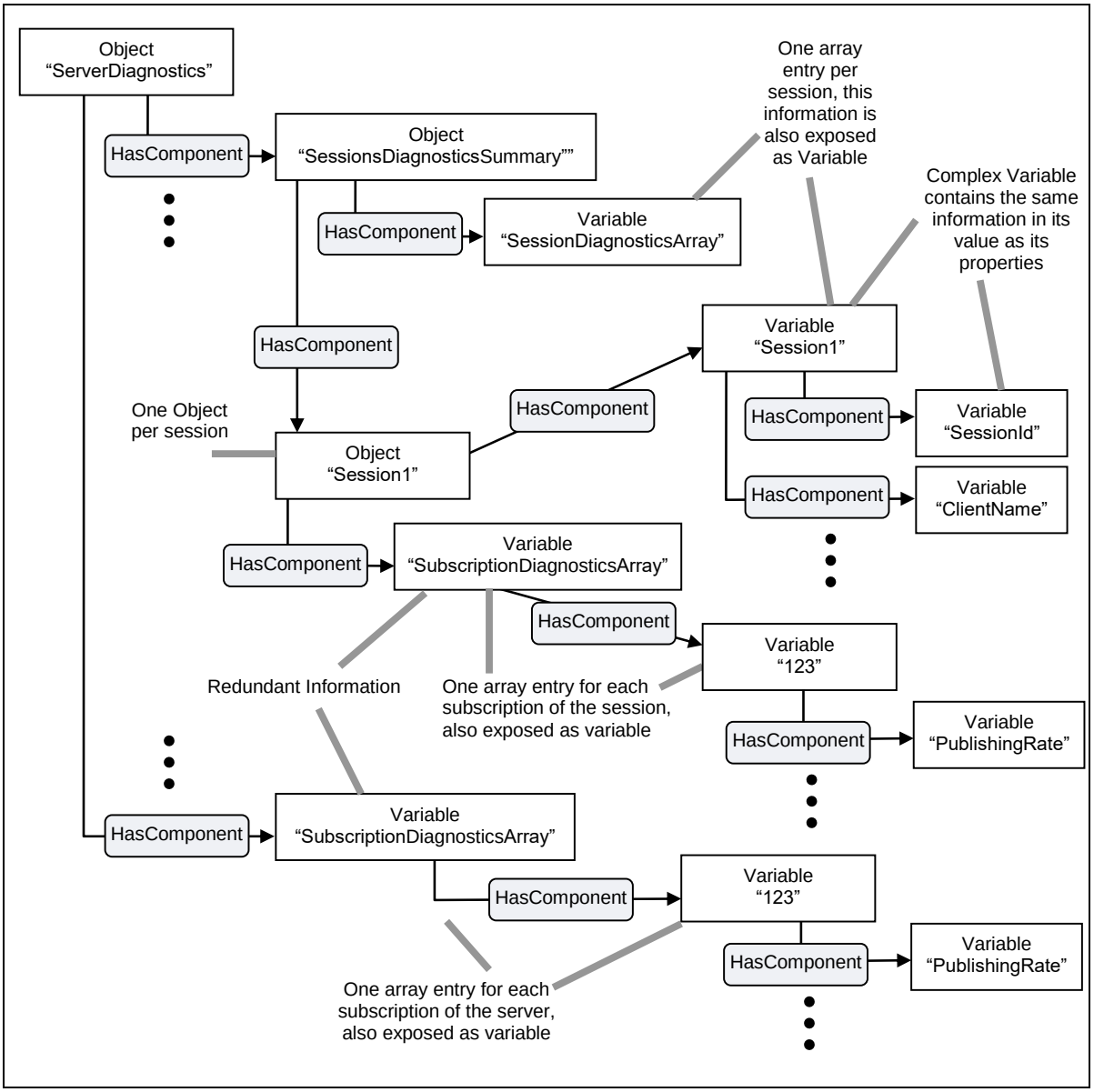

Figure 8 – Excerpt of Diagnostic Information of the Server

8.3.2 Server Object

This *Object* is used as the browse entry point for information about the *Server*. The content of this *Object* is already defined by its type definition in 6.3.1. It is formally defined in Table 88. The *Server Object* serves as root notifier, that is, its *EventNotifier Attribute* shall be set providing *Events*. All *Events* of the *Server* shall be accessible subscribing to the *Events* of the *Server Object*.

Table 88 – Server Definition

Attribute	Value				
BrowseName	Server				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasTypeDefinition	Object Type	ServerType	Defined in 6.3.1		

8.4 ModellingRule Objects

8.4.1 ExposesItsArray

The *ModellingRule ExposesItsArray* is defined in Part 3. Its representation in the *AddressSpace,* the "*ExposesItsArray*" *Object,* is formally defined in Table 89.

Table 89 – ExposesItsArray Definition

Attribute	Value		
BrowseName	ExposesItsArray		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Constraint"

8.4.2 Mandatory

The *ModellingRule Mandatory* is defined in Part 3. Its representation in the *AddressSpace,* the "*Mandatory*" *Object,* is formally defined in Table 90.

Table 90 – Mandatory Definition

Attribute	Value		
BrowseName	Mandatory		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Mandatory"

8.4.3 Optional

The *ModellingRule Optional* is defined in Part 3. Its representation in the *AddressSpace*, the "*Optional*" *Object,* is formally defined in Table 91.

Table 91 – Optional Definition

Attribute	Value		
BrowseName	Optional		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Optional"

8.4.4 OptionalPlaceholder

The *ModellingRule OptionalPlaceholder* is defined in Part 3. Its representation in the *AddressSpace*, the "*OptionalPlaceholder*" *Object,* is formally defined in Table 92.

Table 92 – OptionalPlaceholder Definition

Attribute	Value		
BrowseName	OptionalPlaceholder		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Constraint"

8.4.5 MandatoryPlaceholder

The *ModellingRule MandatoryPlaceholder* is defined in Part 3. Its representation in the *AddressSpace*, the "*MandatoryPlaceholder*" *Object,* is formally defined in Table 93.

Table 93 – MandatoryPlaceholder Definition

Attribute	**Value**		
BrowseName	MandatoryPlaceholder		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Constraint"

9 Standard Methods

9.1 GetMonitoredItems

GetMonitoredItems is used to get information about monitored items of a subscription. Its intended use is defined in Part 4.

Signature

```
GetMonitoredItems(
    [in] UInt32 subscriptionId
    [out] UInt32[] serverHandles
    [out] UInt32[] clientHandles
   );
```

Argument	**Description**
subscriptionId	Identifier of the subscription.
serverHandles	Array of *monitoredItemIds* (serverHandles) for all *MonitoredItems* of the *Subscription* identified by subscriptionId
clientHandles	Array of clientHandles for all *MonitoredItems* of the *Subscription* identified by subscriptionId

Method Result Codes (defined in Call Service)

Result Code	**Description**
Bad_SubscriptionIdInvalid	Defined in Part 4
Bad_UserAccessDenied	Defined in Part 4 The *Method* was not called in the context of the *Session* that owns the *Subscription*.

Table 94 specifies the *AddressSpace* representation for the *GetMonitoredItems Method*.

Table 94 – GetMonitoredItems Method AddressSpace Definition

Attribute	**Value**				
BrowseName	GetMonitoredItems				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.2 ResendData

ResendData is used to get the current values of the data monitored items of a *Subscription where the MonitoringMode is set to Reporting*. Its intended use is defined in Part 4.

Signature

```
ResendData(
    [in] UInt32 subscriptionId
   );
```

Argument	Description
subscriptionId	Identifier of the *Subscription* to refresh.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_SubscriptionIdInvalid	Defined in Part 4
Bad_UserAccessDenied	Defined in Part 4 The *Method* was not called in the context of the *Session* that owns the *Subscription*.

Table 95 specifies the *AddressSpace* representation for the *ResendData Method*.

Table 95 – ResendData Method AddressSpace Definition

Attribute	Value				
BrowseName	ResendData				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

9.3 SetSubscriptionDurable

SetSubscriptionDurable Method is used to set a *Subscription* into a mode where *MonitoredItem* data and event queues are stored and delivered even if an OPC UA *Client* was disconnected for a longer time or the OPC UA *Server* was restarted. Its intended use is defined in Part 4.

Signature

```
SetSubscriptionDurable(
    [in] UInt32 subscriptionId
    [in] UInt32 lifetimeInHours
    [out] UInt32 revisedLifetimeInHours
  );
```

Argument	Description
subscriptionId	Identifier of the *Subscription*.
lifetimeInHours	The requested lifetime in hours for the durable *Subscription*.
revisedLifetimeInHours	The revised lifetime in hours the *Server* applied to the durable *Subscription*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_SubscriptionIdInvalid	Defined in Part 4
Bad_InvalidState	Defined in Part 4 This is returned when a *Subscription* already contains *MonitoredItems*.
Bad_UserAccessDenied	Defined in Part 4 The *Method* was not called in the context of the *Session* that owns the *Subscription*.

Table 96 specifies the *AddressSpace* representation for the *SetSubscriptionDurable Method*.

Table 96 – SetSubscriptionDurable Method AddressSpace Definition

Attribute	Value				
BrowseName	SetSubscriptionDurable				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.4 RequestServerStateChange

The *Method RequestServerStateChange* allows a *Client* to request a state change in the *Server*.

The *Client* shall provide credentials with administrative rights when invoking this *Method* on the *Server*.

Signature

```
RequestServerStateChange(
    [in] ServerState state
    [in] DateTime estimatedReturnTime
    [in] UInt32 secondsTillShutdown
    [in] LocalizedText reason
    [in] Boolean restart
  );
```

Argument	Description
state	The requested target state for the Server. If the new state is accepted by the Server, the State in the *ServerStatus* is updated with the new value.
estimatedReturnTime	Indicates the time at which the *Server* is expected to be available in the state RUNNING_0. If no estimate is known, a null *DateTime* shall be provided. This time will be available in the *EstimatedReturnTime* Property. This parameter shall be ignored by the Server and the Property *EstimatedReturnTime* shall be set to null if the new state is RUNNING_0.
secondsTillShutdown	The number of seconds until a *Server* shutdown. This parameter is ignored unless the state is set to SHUTDOWN_4 or restart is set to True.
reason	A localized text string that describes the reason for the state change request.
restart	A flag indicating if the Server should be restarted before it attempts to change into the requested change. If the restart is True the server changes it state to SHUTDOWN_4 before the restart if secondsTillShutdown is not 0.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user is not authorized to invoke the method
Bad_InvalidState	The requested state was not accepted by the server

Table 97 specifies the *AddressSpace* representation for the *RequestServerStateChange Method*.

Table 97 – RequestServerStateChange Method AddressSpace Definition

Attribute	Value				
BrowseName	RequestServerStateChange				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

10 Standard Views

There are no core OPC UA *Views* defined.

11 Standard ReferenceTypes

11.1 References

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 98.

Table 98 – References ReferenceType

Attributes	Value		
BrowseName	References		
InverseName	--		
Symmetric	True		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HierarchicalReferences	Defined in 11.2
HasSubtype	ReferenceType	NonHierarchicalReferences	Defined in 11.3

11.2 HierarchicalReferences

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 99.

Table 99 – HierarchicalReferences ReferenceType

Attributes	Value		
BrowseName	HierarchicalReferences		
InverseName	--		
Symmetric	False		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasChild	Defined in 11.4
HasSubtype	ReferenceType	Organizes	Defined in 11.6
HasSubtype	ReferenceType	HasEventSource	Defined in 11.14

11.3 NonHierarchicalReferences

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 100.

Table 100 – NonHierarchicalReferences ReferenceType

Attributes	Value		
BrowseName	NonHierarchicalReferences		
InverseName	--		
Symmetric	True		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasModellingRule	Defined in 11.11
HasSubtype	ReferenceType	HasTypeDefinition	Defined in 11.12
HasSubtype	ReferenceType	HasEncoding	Defined in 11.13
HasSubtype	ReferenceType	GeneratesEvent	Defined in 11.16

11.4 HasChild

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 101.

Table 101 – HasChild ReferenceType

Attributes	Value		
BrowseName	HasChild		
InverseName	--		
Symmetric	False		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	Aggregates	Defined in 11.5
HasSubtype	ReferenceType	HasSubtype	Defined in 11.10

11.5 Aggregates

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 102.

Table 102 – Aggregates ReferenceType

Attributes	Value		
BrowseName	Aggregates		
InverseName	--		
Symmetric	False		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasComponent	Defined in 11.7
HasSubtype	ReferenceType	HasProperty	Defined in 11.9

11.6 Organizes

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 103.

Table 103 – Organizes ReferenceType

Attributes	Value		
BrowseName	Organizes		
InverseName	OrganizedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.7 HasComponent

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 104.

Table 104 – HasComponent ReferenceType

Attributes	Value		
BrowseName	HasComponent		
InverseName	ComponentOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasOrderedComponent	Defined in 11.8

11.8 HasOrderedComponent

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 105.

Table 105 – HasOrderedComponent ReferenceType

Attributes	Value		
BrowseName	HasOrderedComponent		
InverseName	OrderedComponentOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.9 HasProperty

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 106.

Table 106 – HasProperty ReferenceType

Attributes	Value		
BrowseName	HasProperty		
InverseName	PropertyOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.10 HasSubtype

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 107.

Table 107 – HasSubtype ReferenceType

Attributes	Value		
BrowseName	HasSubtype		
InverseName	SubtypeOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.11 HasModellingRule

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 108.

Table 108 – HasModellingRule ReferenceType

Attributes	Value		
BrowseName	HasModellingRule		
InverseName	ModellingRuleOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.12 HasTypeDefinition

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 109.

Table 109 – HasTypeDefinition ReferenceType

Attributes	Value		
BrowseName	HasTypeDefinition		
InverseName	TypeDefinitionOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.13 HasEncoding

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 110.

Table 110 – HasEncoding ReferenceType

Attributes	Value		
BrowseName	HasEncoding		
InverseName	EncodingOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.14 HasEventSource

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 111.

Table 111 – HasEventSource ReferenceType

Attributes	Value		
BrowseName	HasEventSource		
InverseName	EventSourceOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasNotifier	Defined in 11.15

11.15 HasNotifier

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 112.

Table 112 – HasNotifier ReferenceType

Attributes	Value		
BrowseName	HasNotifier		
InverseName	NotifierOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.16 GeneratesEvent

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 113.

Table 113 – GeneratesEvent ReferenceType

Attributes	Value		
BrowseName	GeneratesEvent		
InverseName	GeneratedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	AlwaysGeneratesEvent	Defined in 11.17

11.17 AlwaysGeneratesEvent

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 114.

Table 114 – AlwaysGeneratesEvent ReferenceType

Attributes	Value		
BrowseName	AlwaysGeneratesEvent		
InverseName	AlwaysGeneratedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

12 Standard DataTypes

12.1 Overview

An OPC UA *Server* need not expose its *DataTypes* in its *AddressSpace*. Independent of the exposition of *DataTypes*, it shall support the *DataTypes* as described in the following subclauses.

12.2 DataTypes defined in Part 3

Part 3 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 115.

Table 115 – Part 3 DataType Definitions

BrowseName
Argument
AudioDataType
BaseDataType
Boolean
Byte
ByteString
DataTypeDefinition
DateString
DateTime
Decimal
DecimalString
Double
Duration
DurationString
EnumDefinition
Enumeration
EnumField
EnumValueType
Float
Guid
IdType
Image
ImageBMP
ImageGIF
ImageJPG
ImagePNG
Int16
Int32
Int64
Integer
LocaleId
LocalizedText
NamingRuleType
NodeClass
NodeId
NormalizedString
Number
OptionSet
QualifiedName
SByte
String
Structure
StructureDefinition
StructureField
TimeString
TimeZoneDataType
UInt16
UInt32
UInt64
UInteger
Union
UtcTime
XmlElement

Of the *DataTypes* defined in Table 115 only some are the sources of *References* as defined in the following tables.

The *References* of the *BaseDataType* are defined in Table 116.

Table 116 – BaseDataType Definition

Attributes	**Value**		
BrowseName	BaseDataType		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Boolean	FALSE
HasSubtype	DataType	ByteString	FALSE
HasSubtype	DataType	DateTime	FALSE
HasSubtype	DataType	DataValue	FALSE
HasSubtype	DataType	DiagnosticInfo	FALSE
HasSubtype	DataType	Enumeration	TRUE
HasSubtype	DataType	ExpandedNodeId	FALSE
HasSubtype	DataType	Guid	FALSE
HasSubtype	DataType	LocalizedText	FALSE
HasSubtype	DataType	NodeId	FALSE
HasSubtype	DataType	Number	TRUE
HasSubtype	DataType	QualifiedName	FALSE
HasSubtype	DataType	String	FALSE
HasSubtype	DataType	Structure	TRUE
HasSubtype	DataType	XmlElement	FALSE

The *References* of *Structure* are defined in Table 117.

Table 117 – Structure Definition

Attributes	**Value**		
BrowseName	Structure		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Argument	FALSE
HasSubtype	DataType	UserIdentityToken	TRUE
HasSubtype	DataType	AddNodesItem	FALSE
HasSubtype	DataType	AddReferencesItem	FALSE
HasSubtype	DataType	DeleteNodesItem	FALSE
HasSubtype	DataType	DeleteReferencesItem	FALSE
HasSubtype	DataType	ApplicationDescription	FALSE
HasSubtype	DataType	BuildInfo	FALSE
HasSubtype	DataType	RedundantServerDataType	FALSE
HasSubtype	DataType	SamplingIntervalDiagnosticsDataType	FALSE
HasSubtype	DataType	ServerDiagnosticsSummaryDataType	FALSE
HasSubtype	DataType	ServerStatusDataType	FALSE
HasSubtype	DataType	SessionDiagnosticsDataType	FALSE
HasSubtype	DataType	SessionSecurityDiagnosticsDataType	FALSE
HasSubtype	DataType	ServiceCounterDataType	FALSE
HasSubtype	DataType	StatusResult	FALSE
HasSubtype	DataType	SubscriptionDiagnosticsDataType	FALSE
HasSubtype	DataTypes	ModelChangeStructureDataType	FALSE
HasSubtype	DataTypes	SemanticChangeStructureDataType	FALSE
HasSubtype	DataType	SignedSoftwareCertificate	FALSE
HasSubtype	DataType	TimeZoneDataType	FALSE
HasSubtype	DataType	EnumValueType	FALSE
HasSubtype	DataType	OptionSet	TRUE
HasSubtype	DataType	Union	TRUE
HasSubtype	DataType	StructureField	FALSE
HasSubtype	DataType	DataTypeDefinition	TRUE

The *References* of *Enumeration* are defined in Table 118.

Table 118 – Enumeration Definition

Attributes	Value		
BrowseName	Enumeration		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	IdType	FALSE
HasSubtype	DataType	NamingRuleType	FALSE
HasSubtype	DataType	NodeClass	FALSE
HasSubtype	DataType	SecurityTokenRequestType	FALSE
HasSubtype	DataType	MessageSecurityMode	FALSE
HasSubtype	DataType	RedundancySupport	FALSE
HasSubtype	DataType	ServerState	FALSE

The *References* of *ByteString* are defined in Table 119.

Table 119 – ByteString Definition

Attributes	Value		
BrowseName	ByteString		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Image	TRUE
HasSubtype	DataType	AudioDataType	FALSE

The *References* of *Number* are defined in Table 120.

Table 120 – Number Definition

Attributes	Value		
BrowseName	Number		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Integer	TRUE
HasSubtype	DataType	UInteger	TRUE
HasSubtype	DataType	Double	FALSE
HasSubtype	DataType	Float	FALSE
HasSubtype	DataType	Decimal	FALSE

The *References* of *Double* are defined in Table 121.

Table 121 – Double Definition

Attributes	Value		
BrowseName	Double		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Duration	FALSE

The *References* of *Integer* are defined in Table 122.

Table 122 – Integer Definition

Attributes	Value		
BrowseName	Integer		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	SByte	FALSE
HasSubtype	DataType	Int16	FALSE
HasSubtype	DataType	Int32	FALSE
HasSubtype	DataType	Int64	FALSE

The *References* of *DateTime* are defined in Table 123.

Table 123 – DateTime Definition

Attributes	Value		
BrowseName	DateTime		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	UtcTime	FALSE

The *References* of *String* are defined in Table 124.

Table 124 – String Definition

Attributes	Value		
BrowseName	String		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	LocaleId	FALSE
HasSubtype	DataType	NumericRange	FALSE
HasSubtype	DataType	NormalizedString	FALSE
HasSubtype	DataType	DecimalString	FALSE
HasSubtype	DataType	DurationString	FALSE
HasSubtype	DataType	TimeString	FALSE
HasSubtype	DataType	DateString	FALSE

The *References* of UInteger are defined in Table 125.

Table 125 – UInteger Definition

Attributes	Value		
BrowseName	UInteger		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Byte	FALSE
HasSubtype	DataType	UInt16	FALSE
HasSubtype	DataType	UInt32	FALSE
HasSubtype	DataType	UInt64	FALSE

The *References* of Image are defined in Table 126.

Table 126 – Image Definition

Attributes	Value		
BrowseName	Image		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	ImageBMP	FALSE
HasSubtype	DataType	ImageGIF	FALSE
HasSubtype	DataType	ImageJPG	FALSE
HasSubtype	DataType	ImagePNG	FALSE

The *References* of UInt64 are defined in Table 127.

Table 127 – UInt64 Definition

Attributes	Value		
BrowseName	UInt64		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	BitFieldMaskDataType	FALSE

The *References* of DataTypeDefinition are defined in Table 128.

Table 128 – DataTypeDefinition Definition

Attributes	Value		
BrowseName	DataTypeDefinition		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	StructureDefinition	FALSE
HasSubtype	DataType	EnumDefinition	FALSE

The *References* of EnumValueType are defined in Table 129.

Table 129 – EnumValueType Definition

Attributes	Value		
BrowseName	EnumValueType		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	EnumField	FALSE

12.3 DataTypes defined in Part 4

Part 4 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 130.

Table 130 – Part 4 DataType Definitions

BrowseName
AnonymousIdentityToken
DataValue
DiagnosticInfo
ExpandedNodeId
SignedSoftwareCertificate
UserIdentityToken
UserNameIdentityToken
X509IdentityToken
WssIdentityToken
SecurityTokenRequestType
AddNodesItem
AddReferencesItem
DeleteNodesItem
DeleteReferencesItem
NumericRange
MessageSecurityMode
ApplicationDescription

The *SecurityTokenRequestType* is an enumeration that is defined as the type of the requestType parameter of the OpenSecureChannel *Service* in Part 4.

The *AddNodesItem* is a structure that is defined as the type of the nodesToAdd parameter of the AddNodes *Service* in Part 4.

The *AddReferencesItem* is a structure that is defined as the type of the referencesToAdd parameter of the AddReferences *Service* in Part 4.

The *DeleteNodesItem* is a structure that is defined as the type of the nodesToDelete parameter of the DeleteNodes *Service* in Part 4.

The *DeleteReferencesItem* is a structure that is defined as the type of the referencesToDelete parameter of the DeleteReferences *Service* in Part 4.

The *References* of *UserIdentityToken* are defined in Table 131.

Table 131 – UserIdentityToken Definition

Attributes	Value		
BrowseName	UserIdentityToken		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	UserNameIdentityToken	FALSE
HasSubtype	DataType	X509IdentityToken	FALSE
HasSubtype	DataType	WssIdentityToken	FALSE
HasSubtype	DataType	AnonymousIdentityToken	FALSE

12.4 BuildInfo

This structure contains elements that describe the build information of the *Server*. Its elements are defined in Table 132.

Table 132 – BuildInfo Structure

Name	Type	Description
BuildInfo	structure	Information that describes the build of the software.
productUri	String	URI that identifies the software
manufacturerName	String	Name of the software manufacturer.
productName	String	Name of the software.
softwareVersion	String	Software version
buildNumber	String	Build number
buildDate	UtcTime	Date and time of the build.

Its representation in the *AddressSpace* is defined in Table 133.

Table 133 – BuildInfo Definition

Attributes	Value
BrowseName	BuildInfo

12.5 RedundancySupport

This *DataType* is an enumeration that defines the redundancy support of the *Server*. Its values are defined in Table 134.

Table 134 – RedundancySupport Values

Value	Description
NONE_0	None means that there is no redundancy support.
COLD_1	Cold means that the server supports cold redundancy as defined in Part 4.
WARM_2	Warm means that the server supports warm redundancy as defined in Part 4.
HOT_3	Hot means that the server supports hot redundancy as defined in Part 4.
TRANSPARENT_4	Transparent means that the server supports transparent redundancy as defined in Part 4.
HOT_AND_MIRRORED_5	HotAndMirrored means that the server supports HotAndMirrored redundancy as defined in Part 4.

See Part 4 for a more detailed description of the different values.

Its representation in the *AddressSpace* is defined in Table 135.

Table 135 – RedundancySupport Definition

Attributes	Value
BrowseName	RedundancySupport

12.6 ServerState

This *DataType* is an enumeration that defines the execution state of the *Server*. Its values are defined in Table 136.

Table 136 – ServerState Values

Value	Description
RUNNING_0	The *Server* is running normally. This is the usual state for a *Server*.
FAILED_1	A vendor-specific fatal error has occurred within the *Server*. The *Server* is no longer functioning. The recovery procedure from this situation is vendor-specific. Most *Service* requests should be expected to fail.
NO_CONFIGURATION_2	The *Server* is running but has no configuration information loaded and therefore does not transfer data.
SUSPENDED_3	The *Server* has been temporarily suspended by some vendor-specific method and is not receiving or sending data.
SHUTDOWN_4	The *Server* initiated a shut down or is in the process of shutting down. This *ServerState* is intended as an indication to *Clients* connected to the *Server* to orderly disconnect from the *Server* before the *Server* completes the shut down.
TEST_5	The *Server* is in Test Mode. The outputs are disconnected from the real hardware, but the *Server* will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. StatusCode will generally be returned normally.
COMMUNICATION_FAULT_6	The *Server* is running properly, but is having difficulty accessing data from its data sources. This may be due to communication problems or some other problem preventing the underlying device, control system, etc. from returning valid data. It may be a complete failure, meaning that no data is available, or a partial failure, meaning that some data is still available. It is expected that items affected by the fault will individually return with a BAD FAILURE status code indication for the items.
UNKNOWN_7	This state is used only to indicate that the OPC UA *Server* does not know the state of underlying system.

Its representation in the *AddressSpace* is defined in Table 137.

Table 137 – ServerState Definition

Attributes	Value
BrowseName	ServerState

12.7 RedundantServerDataType

This structure contains elements that describe the status of the *Server*. Its composition is defined in Table 138.

Table 138 – RedundantServerDataType Structure

Name	Type	Description
RedundantServerDataType	structure	
serverId	String	The Id of the server (not the URI).
serviceLevel	Byte	The service level of the server.
serverState	ServerState	The current state of the server.

Its representation in the *AddressSpace* is defined in Table 139.

Table 139 – RedundantServerDataType Definition

Attributes	Value
BrowseName	RedundantServerDataType

12.8 SamplingIntervalDiagnosticsDataType

This structure contains diagnostic information about the sampling rates currently used by the *Server*. Its elements are defined in Table 140.

Table 140 – SamplingIntervalDiagnosticsDataType Structure

Name	Type	Description
SamplingIntervalDiagnosticsDataType	structure	
samplingInterval	Duration	The sampling interval in milliseconds.
sampledMonitoredItemsCount	UInt32	The number of *MonitoredItems* being sampled at this sample rate.
maxSampledMonitoredItemsCount	UInt32	The maximum number of *MonitoredItems* being sampled at this sample rate at the same time since the server was started (restarted).
disabledMonitoredItemsSamplingCount	UInt32	The number of *MonitoredItems* at this sample rate whose sampling currently disabled.

Its representation in the *AddressSpace* is defined in Table 141.

Table 141 – SamplingIntervalDiagnosticsDataType Definition

Attributes	Value
BrowseName	SamplingIntervalDiagnosticsDataType

12.9 ServerDiagnosticsSummaryDataType

This structure contains diagnostic summary information for the *Server*. Its elements are defined in Table 142.

Table 142 – ServerDiagnosticsSummaryDataType Structure

Name	Type	Description
ServerDiagnosticsSummaryDataType	structure	
serverViewCount	UInt32	The number of server-created views in the server.
currentSessionCount	UInt32	The number of client sessions currently established in the server.
cumulatedSessionCount	UInt32	The cumulative number of client sessions that have been established in the server since the server was started (or restarted). This includes the *currentSessionCount*.
securityRejectedSessionCount	UInt32	The number of client session establishment requests (ActivateSession and CreateSession) that were rejected due to security constraints since the server was started (or restarted).
rejectedSessionCount	UInt32	The number of client session establishment requests (ActivateSession and CreateSession) that were rejected since the server was started (or restarted). This number includes the *securityRejectedSessionCount*.
sessionTimeoutCount	UInt32	The number of client sessions that were closed due to timeout since the server was started (or restarted).
sessionAbortCount	UInt32	The number of client sessions that were closed due to errors since the server was started (or restarted).
publishingIntervalCount	UInt32	The number of publishing intervals currently supported in the server.
currentSubscriptionCount	UInt32	The number of subscriptions currently established in the server.
cumulatedSubscriptionCount	UInt32	The cumulative number of subscriptions that have been established in the server since the server was started (or restarted). This includes the *currentSubscriptionCount*.
securityRejectedRequestsCount	UInt32	The number of requests that were rejected due to security constraints since the server was started (or restarted). The requests include all *Services* defined in Part 4, also requests to create sessions.
rejectedRequestsCount	UInt32	The number of requests that were rejected since the server was started (or restarted). The requests include all *Services* defined in Part 4, also requests to create sessions. This number includes the *securityRejectedRequestsCount*.

Its representation in the *AddressSpace* is defined in Table 143.

Table 143 – ServerDiagnosticsSummaryDataType Definition

Attributes	Value
BrowseName	ServerDiagnosticsSummaryDataType

12.10 ServerStatusDataType

This structure contains elements that describe the status of the *Server*. Its composition is defined in Table 144.

Table 144 – ServerStatusDataType Structure

Name	Type	Description
ServerStatusDataType	structure	
startTime	UtcTime	Time (UTC) the *Server* was started. This is constant for the *Server* instance and is not reset when the *Server* changes state. Each instance of a *Server* should keep the time when the process started.
currentTime	UtcTime	The current time (UTC) as known by the *Server*.
state	ServerState	The current state of the *Server*. Its values are defined in 12.6.
buildInfo	BuildInfo	
secondsTillShutdown	UInt32	Approximate number of seconds until the *Server* will be shut down. The value is only relevant once the state changes into SHUTDOWN_*4*. After the *Server* shut down is initated, the state changes to SHUTDOWN_4 and the actual shut down should be delayed for a configurable time if *Clients* are connected to the *Server* to allow these Clients an orderly disconnect.
shutdownReason	LocalizedText	An optional localized text indicating the reason for the shutdown. The value is only relevant once the state changes into SHUTDOWN_*4*.

Its representation in the *AddressSpace* is defined in Table 145.

Table 145 – ServerStatusDataType Definition

Attributes	Value
BrowseName	ServerStatusDataType

12.11 SessionDiagnosticsDataType

This structure contains diagnostic information about client sessions. Its elements are defined in Table 146. Most of the values represented in this structure provide information about the number of calls of a *Service*, the number of currently used *MonitoredItems*, etc. Those numbers need not provide the exact value; they need only provide the approximate number, so that the *Server* is not burdened with providing the exact numbers.

Table 146 – SessionDiagnosticsDataType Structure

Name	Type	Description
SessionDiagnosticsDataType	structure	
sessionId	NodeId	Server-assigned identifier of the session.
sessionName	String	The name of the session provided in the CreateSession request.
clientDescription	Application Description	The description provided by the client in the CreateSession request.
serverUri	String	The serverUri request in the CreateSession request.
endpointUrl	String	The endpointUrl passed by the client to the CreateSession request.
localeIds	LocaleId[]	Array of LocaleIds specified by the client in the open session call.
actualSessionTimeout	Duration	The requested session timeout specified by the client in the open session call.
maxResponseMessageSize	UInt32	The maximum size for the response message sent to the client.
clientConnectionTime	UtcTime	The server timestamp when the client opens the session.
clientLastContactTime	UtcTime	The server timestamp of the last request of the client in the context of the session.
currentSubscriptionsCount	UInt32	The number of subscriptions currently used by the session.
currentMonitoredItemsCount	UInt32	The number of *MonitoredItems* currently used by the session
currentPublishRequestsInQueue	UInt32	The number of publish requests currently in the queue for the session.
totalRequestCount	ServiceCounter DataType	Counter of all *Services*, identifying the number of received requests of any *Services* on the session.
unauthorizedRequestCount	UInt32	Counter of all *Services*, identifying the number of *Service* requests that were rejected due to authorization failure

Name	Type	Description
readCount	ServiceCounter DataType	Counter of the Read *Service*, identifying the number of received requests of this *Service* on the session.
historyReadCount	ServiceCounter DataType	Counter of the HistoryRead *Service*, identifying the number of received requests of this *Service* on the session.
writeCount	ServiceCounter DataType	Counter of the Write *Service*, identifying the number of received requests of this *Service* on the session.
historyUpdateCount	ServiceCounter DataType	Counter of the HistoryUpdate *Service*, identifying the number of received requests of this *Service* on the session.
callCount	ServiceCounter DataType	Counter of the Call *Service*, identifying the number of received requests of this *Service* on the session.
createMonitoredItemsCount	ServiceCounter DataType	Counter of the CreateMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session.
modifyMonitoredItemsCount	ServiceCounter DataType	Counter of the ModifyMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session.
setMonitoringModeCount	ServiceCounter DataType	Counter of the SetMonitoringMode *Service*, identifying the number of received requests of this *Service* on the session.
setTriggeringCount	ServiceCounter DataType	Counter of the SetTriggering *Service*, identifying the number of received requests of this *Service* on the session.
deleteMonitoredItemsCount	ServiceCounter DataType	Counter of the DeleteMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session.
createSubscriptionCount	ServiceCounter DataType	Counter of the CreateSubscription *Service*, identifying the number of received requests of this *Service* on the session.
modifySubscriptionCount	ServiceCounter DataType	Counter of the ModifySubscription *Service*, identifying the number of received requests of this *Service* on the session.
setPublishingModeCount	ServiceCounter DataType	Counter of the SetPublishingMode *Service*, identifying the number of received requests of this *Service* on the session.
publishCount	ServiceCounter DataType	Counter of the Publish *Service*, identifying the number of received requests of this *Service* on the session.
republishCount	ServiceCounter DataType	Counter of the Republish *Service*, identifying the number of received requests of this *Service* on the session.
transferSubscriptionsCount	ServiceCounter DataType	Counter of the TransferSubscriptions *Service*, identifying the number of received requests of this *Service* on the session.
deleteSubscriptionsCount	ServiceCounter DataType	Counter of the DeleteSubscriptions *Service*, identifying the number of received requests of this *Service* on the session.
addNodesCount	ServiceCounter DataType	Counter of the AddNodes *Service*, identifying the number of received requests of this *Service* on the session.
addReferencesCount	ServiceCounter DataType	Counter of the AddReferences *Service*, identifying the number of received requests of this *Service* on the session.
deleteNodesCount	ServiceCounter DataType	Counter of the DeleteNodes *Service*, identifying the number of received requests of this *Service* on the session.
deleteReferencesCount	ServiceCounter DataType	Counter of the DeleteReferences *Service*, identifying the number of received requests of this *Service* on the session.
browseCount	ServiceCounter DataType	Counter of the Browse *Service*, identifying the number of received requests of this *Service* on the session.
browseNextCount	ServiceCounter DataType	Counter of the BrowseNext *Service*, identifying the number of received requests of this *Service* on the session.
translateBrowsePathsToNodeIdsCount	ServiceCounter DataType	Counter of the TranslateBrowsePathsToNodeIds *Service*, identifying the number of received requests of this *Service* on the session.
queryFirstCount	ServiceCounter DataType	Counter of the QueryFirst *Service*, identifying the number of received requests of this *Service* on the session.
queryNextCount	ServiceCounter DataType	Counter of the QueryNext *Service*, identifying the number of received requests of this *Service* on the session.
registerNodesCount	ServiceCounter DataType	Counter of the RegisterNodes *Service*, identifying the number of received requests of this *Service* on the session.
unregisterNodesCount	ServiceCounter DataType	Counter of the UnregisterNodes*Service*, identifying the number of received requests of this *Service* on the session.

Its representation in the *AddressSpace* is defined in Table 147.

Table 147 – SessionDiagnosticsDataType Definition

Attributes	Value
BrowseName	SessionDiagnosticsDataType

12.12 SessionSecurityDiagnosticsDataType

This structure contains security-related diagnostic information about client sessions. Its elements are defined in Table 148. Because this information is security-related, it shall only be accessible by authorised users.

Table 148 – SessionSecurityDiagnosticsDataType Structure

Name	Type	Description
SessionSecurityDiagnosticsDataType	structure	
sessionId	NodeId	Server-assigned identifier of the session.
clientUserIdOfSession	String	Name of authenticated user when creating the session.
clientUserIdHistory	String[]	Array containing the name of the authenticated user currently active (either from creating the session or from calling the *ActivateSession Service*) and the history of those names. Each time the active user changes, an entry shall be made at the end of the array. The active user is always at the end of the array. Servers may restrict the size of this array, but shall support at least a size of 2. How the name of the authenticated user can be obtained from the system via the information received as part of the session establishment is defined in 6.4.3.
authenticationMechanism	String	Type of authentication currently used by the session. The String shall be one of the lexical names of the *UserIdentityTokenType* Enum.
encoding	String	Which encoding is used on the wire. The String shall be 'XML', 'JSON' or 'UA Binary'.
transportProtocol	String	Which transport protocol is used. The String shall be the scheme from the URL used to establish the session. For example, 'opc.tcp', 'opc.wss' or 'https'. The formal protocol URL scheme strings are defined in Part 6.
securityMode	MessageSecurityMode	The message security mode used for the session.
securityPolicyUri	String	The name of the security policy used for the session.
clientCertificate	ByteString	The application instance certificate provided by the client in the CreateSession request.

Its representation in the *AddressSpace* is defined in Table 149.

Table 149 – SessionSecurityDiagnosticsDataType Definition

Attributes	Value
BrowseName	SessionSecurityDiagnosticsDataType

12.13 ServiceCounterDataType

This structure contains diagnostic information about subscriptions. Its elements are defined in Table 150.

Table 150 – ServiceCounterDataType Structure

Name	Type	Description
ServiceCounterDataType	structure	
totalCount	UInt32	The number of *Service* requests that have been received.
errorCount	UInt32	The total number of *Service* requests that were rejected.

Its representation in the *AddressSpace* is defined in Table 151.

Table 151 – ServiceCounterDataType Definition

Attributes	Value
BrowseName	ServiceCounterDataType

12.14 StatusResult

This structure combines a *StatusCode* and diagnostic information and can, for example, be used by Methods to return several *StatusCodes* and the corresponding diagnostic information that are not handled in the *Call Service* parameters. The elements of this *DataType* are defined in Table 152. Whether the diagnosticInfo is returned depends on the setting of the *Service* calls.

Table 152 – StatusResult Structure

Name	Type	Description
StatusResult	structure	
statusCode	StatusCode	The StatusCode.
diagnosticInfo	DiagnosticInfo	The diagnostic information for the statusCode.

Its representation in the *AddressSpace* is defined in Table 153.

Table 153 – StatusResult Definition

Attributes	Value
BrowseName	StatusResult

12.15 SubscriptionDiagnosticsDataType

This structure contains diagnostic information about subscriptions. Its elements are defined in Table 154.

Table 154 – SubscriptionDiagnosticsDataType Structure

Name	Type	Description
SubscriptionDiagnosticsDataType	structure	
sessionId	NodeId	Server-assigned identifier of the session the subscription belongs to.
subscriptionId	UInt32	Server-assigned identifier of the subscription.
priority	Byte	The priority the client assigned to the subscription.
publishingInterval	Duration	The publishing interval of the subscription in milliseconds
maxKeepAliveCount	UInt32	The maximum keep-alive count of the subscription.
maxLifetimeCount	UInt32	The maximum lifetime count of the subscription.
maxNotificationsPerPublish	UInt32	The maximum number of notifications per publish response.
publishingEnabled	Boolean	Whether publishing is enabled for the subscription.
modifyCount	UInt32	The number of ModifySubscription requests received for the subscription.
enableCount	UInt32	The number of times the subscription has been enabled.
disableCount	UInt32	The number of times the subscription has been disabled.
republishRequestCount	UInt32	The number of Republish *Service* requests that have been received and processed for the subscription.
republishMessageRequestCount	UInt32	The total number of messages that have been requested to be republished for the subscription. Note that due to the design of the Republish *Service* this number is always equal to the republishRequestCount.
republishMessageCount	UInt32	The number of messages that have been successfully republished for the subscription.
transferRequestCount	UInt32	The total number of TransferSubscriptions *Service* requests that have been received for the subscription.
transferredToAltClientCount	UInt32	The number of times the subscription has been transferred to an alternate client.
transferredToSameClientCount	UInt32	The number of times the subscription has been transferred to an alternate session for the same client.
publishRequestCount	UInt32	The number of Publish *Service* requests that have been received and processed for the subscription.
dataChangeNotificationsCount	UInt32	The number of data change Notifications sent by the subscription.
eventNotificationsCount	UInt32	The number of Event Notifications sent by the subscription.
notificationsCount	UInt32	The total number of Notifications sent by the subscription.
latePublishRequestCount	UInt32	The number of times the subscription has entered the LATE State, i.e. the number of times the publish timer expires and there are unsent notifications.
currentKeepAliveCount	UInt32	The number of times the subscription has entered the KEEPALIVE State.
currentLifetimeCount	UInt32	The current lifetime count of the subscription.
unacknowledgedMessageCount	UInt32	The number of unacknowledged messages saved in the republish queue.
discardedMessageCount	UInt32	The number of messages that were discarded before they were acknowledged.
monitoredItemCount	UInt32	The total number of monitored items of the subscription, including the disabled monitored items.
disabledMonitoredItemCount	UInt32	The number of disabled monitored items of the subscription.
monitoringQueueOverflowCount	UInt32	The number of times a monitored item dropped notifications because of a queue overflow.
nextSequenceNumber	UInt32	Sequence number for the next notification message.
eventQueueOverFlowCount	UInt32	The number of times a monitored item in the subscription has generated an Event of type EventQueueOverflowEventType.

Its representation in the *AddressSpace* is defined in Table 155.

Table 155 – SubscriptionDiagnosticsDataType Definition

Attributes	Value
BrowseName	SubscriptionDiagnosticsDataType

12.16 ModelChangeStructureDataType

This structure contains elements that describe changes of the model. Its composition is defined in Table 156.

Table 156 – ModelChangeStructureDataType Structure

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>ModelChangeStructure DataType</td><td>structure</td><td></td></tr>
<tr><td>affected</td><td>NodeId</td><td>NodeId of the Node that was changed. The client should assume that the affected Node has been created or deleted, had a Reference added or deleted, or the DataType has changed as described by the verb.</td></tr>
<tr><td>affectedType</td><td>NodeId</td><td>If the affected Node was an Object or Variable, affectedType contains the NodeId of the TypeDefinitionNode of the affected Node. Otherwise it is set to null.</td></tr>
<tr><td>verb</td><td>Byte</td><td>Describes the changes happening to the affected Node.
The verb is an 8-bit unsigned integer used as bit mask with the structure defined in the following table:
<table>
<tr><th>Field</th><th>Bit</th><th>Description</th></tr>
<tr><td>NodeAdded</td><td>0</td><td>Indicates the affected Node has been added.</td></tr>
<tr><td>NodeDeleted</td><td>1</td><td>Indicates the affected Node has been deleted.</td></tr>
<tr><td>ReferenceAdded</td><td>2</td><td>Indicates a Reference has been added. The affected Node may be either a SourceNode or TargetNode. Note that an added bidirectional Reference is reflected by two changes.</td></tr>
<tr><td>ReferenceDeleted</td><td>3</td><td>Indicates a Reference has been deleted. The affected Node may be either a SourceNode or TargetNode. Note that a deleted bidirectional Reference is reflected by two changes.</td></tr>
<tr><td>DataTypeChanged</td><td>4</td><td>This verb may be used only for affected Nodes that are Variables or VariableTypes. It indicates that the DataType Attribute has changed.</td></tr>
<tr><td>Reserved</td><td>5:7</td><td>Reserved for future use. Shall always be zero.</td></tr>
</table>
A verb may identify several changes on the affected Node at once. This feature should be used if event compression is used (see Part 3 for details).
Note that all verbs shall always be considered in the context where the ModelChangeStructureDataType is used. A NodeDeleted may indicate that a Node was removed from a view but still exists in other Views.</td></tr>
</table>

Its representation in the *AddressSpace* is defined in Table 157.

Table 157 – ModelChangeStructureDataType Definition

Attributes	Value
BrowseName	ModelChangeStructureDataType

12.17 SemanticChangeStructureDataType

This structure contains elements that describe a change of the model. Its composition is defined in Table 158.

Table 158 – SemanticChangeStructureDataType Structure

Name	Type	Description
SemanticChangeStructureDataType	structure	
affected	NodeId	*NodeId* of the *Node* that owns the *Property* that has changed.
affectedType	NodeId	If the *affected Node* was an *Object* or *Variable*, *affectedType* contains the *NodeId* of the *TypeDefinitionNode* of the *affected Node*. Otherwise it is set to null.

Its representation in the *AddressSpace* is defined in Table 159.

Table 159 – SemanticChangeStructureDataType Definition

Attributes	Value
BrowseName	SemanticChangeStructureDataType

12.18 BitFieldMaskDataType

This simple *DataType* is a subtype of UInt64 and represents a bit mask up to 32 bits where individual bits can be written without modifying the other bits.

The first 32 bits (least significant bits) of the *BitFieldMaskDataType* represent the bit mask and the second 32 bits represent the validity of the bits in the bit mask. When the *Server* returns the value to the client, the validity provides information of which bits in the bit mask have a meaning. When the client passes the value to the *Server*, the validity defines which bits should be written. Only those bits defined in validity are changed in the bit mask, all others stay the same. The *BitFieldMaskDataType* can be used as *DataType* in the *OptionSetType VariableType*

Its representation in the *AddressSpace* is defined in Table 160.

Table 160 – BitFieldMaskDataType Definition

Attributes	Value
BrowseName	BitFieldMaskDataType

12.19 NetworkGroupDataType

This structure contains information on different network paths for one *Server*. Its composition is defined in Table 161.

Table 161 – NetworkGroupDataType Structure

Name	Type	Description
NetworkGroupDataType	structure	
serverUri	String	URI of the Server represented by the network group.
networkPaths	EndpointUrlListDataType[]	Array of different network paths to the server, for example provided by different network cards in a Server node. Each network path can have several Endpoints representing different protocol options for the same path.

Its representation in the *AddressSpace* is defined in Table 162.

Table 162 – NetworkGroupDataType Definition

Attributes	Value
BrowseName	NetworkGroupDataType

12.20 EndpointUrlListDataType

This structure represents a list of URLs of an *Endpoint*. Its composition is defined in Table 163.

Table 163 – EndpointUrlListDataType Structure

Name	Type	Description
EndpointUrlListDataType	structure	
endpointUrlList	String[]	List of URLs of an Endpoint.

Its representation in the *AddressSpace* is defined in Table 164.

Table 164 – EndpointUrlListDataType Definition

Attributes	Value
BrowseName	EndpointUrlListDataType

12.21 KeyValuePair

This *DataType* is used to provide a key value pair. The *KeyValuePair* is formally defined in Table 165.

Table 165 – KeyValuePair Structure

Name	Type	Description
KeyValuePair	structure	
key	QualifiedName	The key of the value.
value	BaseDataType	The value associated with the key.

12.22 EndpointType

This *structure describes an Endpoint.* The *EndpointType* is formally defined in Table 166.

Table 166 – EndpointType Structure

Name	Type	Description
EndpointType	structure	
endpointUrl	String	The URL for the *Endpoint*.
securityMode	MessageSecurityMode	The type of message security. The type *MessageSecurityMode* type is defined in Part 4.
securityPolicyUri	String	The URI of the *SecurityPolicy*.
transportProfileUri	String	The URI of the *Transport Profile*.

Annex A
(informative)

Design decisions when modelling the server information

A.1 Overview

This annex describes the design decisions of modelling the information provided by each OPC UA *Server*, exposing its capabilities, diagnostic information, and other data needed to work with the *Server*, such as the *NamespaceArray*.

This annex gives an example of what should be considered when modelling data using the Address Space Model. General considerations for using the Address Space Model can be found in Part 3.

This annex is for information only, that is, each *Server* vendor can model its data in the appropriate way that fits its needs.

The following subclauses describe the design decisions made while modelling the *Server Object*. General *DataTypes*, *VariableTypes* and *ObjectTypes* such as the *EventTypes* described in this standard are not taken into account.

A.2 ServerType and Server Object

The first decision is to decide at what level types are needed. Typically, each *Server* will provide one *Server Object* with a well-known *NodeId*. The *NodeIds* of the containing *Nodes* are also well-known because their symbolic name is specified in this standard and the *NodeId* is based on the symbolic name in Part 6. Nevertheless, aggregating *Servers* may want to expose the *Server Objects* of the OPC UA *Servers* they are aggregating in their *AddressSpace*. Therefore, it is very helpful to have a type definition for the *Server Object*. The *Server Object* is an *Object*, because it groups a set of *Variables* and *Objects* containing information about the *Server*. The *ServerType* is a complex *ObjectType*, because the basic structure of the *Server Object* should be well-defined. However, the *Server Object* can be extended by adding *Variables* and *Objects* in an appropriate structure of the *Server Object* or its containing *Objects*.

A.3 Typed complex Objects beneath the Server Object

Objects beneath the *Server Object* used to group information, such as *Server* capabilities or diagnostics, are also typed because an aggregating *Server* may want to provide only part of the *Server* information, such as diagnostics information, in its *AddressSpace*. Clients are able to program against these structures if they are typed, because they have its type definition.

A.4 Properties versus DataVariables

Since the general description in Part 3 about the semantic difference between *Properties* and *DataVariables* are not applicable for the information provided about the *Server* the rules described in Part 3 are used.

If simple data structures should be provided, *Properties* are used. Examples of *Properties* are the *NamespaceArray* of the *Server Object* and the *MinSupportedSampleRate* of the *ServerCapabilities Object*.

If complex data structures are used, *DataVariables* are used. Examples of *DataVariables* are the *ServerStatus* of the *Server Object* and the *ServerDiagnosticsSummary* of the *ServerDiagnostics Object*.

A.5 Complex Variables using complex DataTypes

DataVariables providing complex data structures expose their information as complex *DataTypes*, as well as components in the *AddressSpace*. This allows access to simple values as well as access to the whole information at once in a transactional context.

For example, the *ServerStatus Variable* of the *Server Object* is modelled as a complex *DataVariable* having the *ServerStatusDataType* providing all information about the *Server* status. But it also exposes the *CurrentTime* as a simple *DataVariable*, because a client may want to read only the current time of the *Server*, and is not interested in the build information, etc.

A.6 Complex Variables having an array

A special case of providing complex data structures is an array of complex data structures. The *SubscriptionDiagnosticsArrayType* is an example of how this is modelled. It is an array of a complex data structure, providing information of a subscription. Because a *Server* typically has several subscriptions, it is an array. Some clients may want to read the diagnostic information about all subscriptions at once; therefore it is modelled as an array in a *Variable*. On the other hand, a client may be interested in only a single entry of the complex structure, such as the *PublishRequestCount*. Therefore, each entry of the array is also exposed individually as a complex *DataVariable*, having each entry exposed as simple data.

Note that it is never necessary to expose the individual entries of an array to access them separately. The *Services* already allow accessing individual entries of an array of a *Variable*. However, if the entries should also be used for other purposes in the *AddressSpace,* such as having *References* or additional *Properties* or exposing their complex structure using *DataVariables,* it is useful to expose them individually.

A.7 Redundant information

Providing redundant information should generally be avoided. But to fulfil the needs of different clients, it may be helpful.

Using complex *DataVariables* automatically leads to providing redundant information, because the information is directly provided in the complex *DataType* of the *Value Attribute* of the complex *Variable*, and also exposed individually in the components of the complex *Variable*.

The diagnostics information about subscriptions is provided in two different locations. One location is the *SubscriptionDiagnosticsArray* of the *ServerDiagnostics Object*, providing the information for all subscriptions of the *Server*. The second location is the *SubscriptionDiagnosticsArray* of each individual *SessionDiagnosticsObject Object*, providing only the subscriptions of the session. This is useful because some clients may be interested in only the subscriptions grouped by sessions, whereas other clients may want to access the diagnostics information of all sessions at once.

The *SessionDiagnosticsArray* and the *SessionSecurityDiagnosticsArray* of the *SessionsDiagnosticsSummary Object* do not expose their individual entries, although they represent an array of complex data structures. But the information of the entries can also be accessed individually as components of the *SessionDiagnostics Objects* provided for each session by the *SessionsDiagnosticsSummary Object*. A client can either access the arrays (or parts of the arrays) directly or browse to the *SessionDiagnostics Objects* to get the information of the individual entries. Thus, the information provided is redundant, but the *Variables* containing the arrays do not expose their individual entries.

A.8 Usage of the BaseDataVariableType

All *DataVariables* used to expose complex data structures of complex *DataVariables* have the *BaseDataVariableType* as type definition if they are not complex by themselves. The reason for this approach is that the complex *DataVariables* already define the semantic of the containing *DataVariables* and this semantic is not used in another context. It is not expected that they are subtyped, because they should reflect the data structure of the *DataType* of the complex *DataVariable*.

A.9 Subtyping

Subtyping is used for modelling information about the redundancy support of the *Server*. Because the provided information shall differ depending on the supported redundancy of the *Server*, subtypes of the *ServerRedundancyType* will be used for this purpose.

Subtyping is also used as an extensibility mechanism (see A.10).

A.10 Extensibility mechanism

The information of the *Server* will be extended by other parts of this series of standards, by companion specifications or by *Server* vendors. There are preferred ways to provide the additional information.

Do not subtype *DataTypes* to provide additional information about the *Server*. Clients might not be able to read those new defined *DataTypes* and are not able to get the information, including the basic information. If information is added by several sources, the *DataType* hierarchy may be difficult to maintain. Note that this rule applies to the information about the *Server*; in other scenarios this may be a useful way to add information.

Add *Objects* containing *Variables* or add *Variables* to the *Objects* defined in this part. If, for example, additional diagnostic information per subscription is needed, add a new *Variable* containing in array with an entry per subscription in the same places that the *SubscriptionDiagnosticsArray* is used.

Use subtypes of the *ServerVendorCapabilityType* to add information about the server-specific capabilities on the *ServerCapabilities Objects*. Because this extensibility point is already defined in this part, clients will look there for additional information.

Use a subtype of the *VendorServerInfoType* to add server-specific information. Because an *Object* of this type is already defined in this part, clients will look there for server-specific information.

Annex B
(normative)

StateMachines

B.1 General

This annex describes the basic infrastructure to model state machines. It defines *ObjectTypes*, *VariableTypes* and *ReferenceTypes* and explains how they should be used.

This annex is an integral part of this standard, that is, the types defined in this annex have to be used as defined. However, it is not required but strongly recommended that a *Server* uses these types to expose its state machines. The defined types may be subtyped to refine their behaviour.

When a *Server* exposes its state machine using the types defined in this annex, it might only provide a simplified view on its internal state machine, hiding for example substates or putting several internal states into one exposed state.

The scope of the state machines described in this annex is to provide an appropriate foundation for state machines needed for Part 9 and Part 10. It does not provide more complex functionality of a state machine like parallel states, forks and joins, history states, choices and junctions, etc. However, the base state machine defined in this annex can be extended to support such concepts.

The following clauses describe examples of state machines, define state machines in the context of this annex and define the representation of state machines in OPC UA. Finally, some examples of state machines, represented in OPC UA, are given.

B.2 Examples of finite state machines

B.2.1 Simple state machine

The following example provides an overview of the base features that the state machines defined in this annex will support. In the following, a more complex example is given, that also supports sub-state machines.

Figure B.1 gives an overview over a simple state machine. It contains the three states “State1”, “State2” and “State3”. There are transitions from “State1” to “State2”, “State2” to “State2”, etc. Some of the transitions provide additional information with regard to what causes (or triggers) the transition, for example the call of “Method1” for the transition from “State1” to “State2”. The effect (or action) of the transition can also be specified, for example the generation of an *Event* of the “EventType1” in the same transition. The notation used to identify the cause is simply listing it on the transition, the effect is prefixed with a “/”. More than one cause or effect are separated by a “,”. Not every transition has to have a cause or effect, for example the transition between “State2” and “State3”.

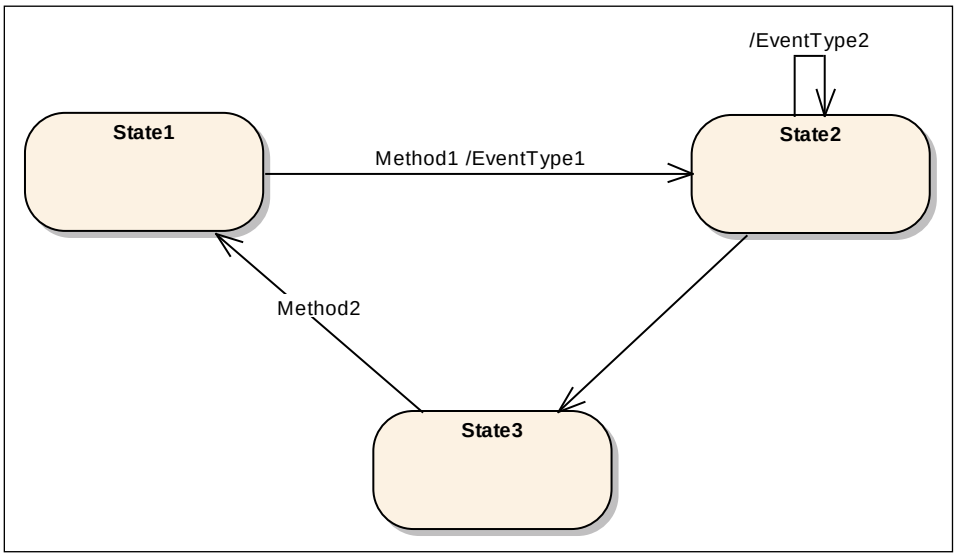

Figure B.1 – Example of a simple state machine

For simplicity, the state machines described in this annex will only support causes in form of specifying *Methods* that have to be called and effects in form of *EventTypes* of *Events* that are generated. However, the defined infrastructure allows extending this to support additional different causes and effects.

B.2.2 State machine containing substates

Figure B.2 shows an example of a state machine where “State6” is a sub-state-machine. This means, that when the overall state machine is in State6, this state can be distinguished to be in the sub-states “State7” or “State8”. Sub-state-machines can be nested, that is, “State7” could be another sub-state-machine.

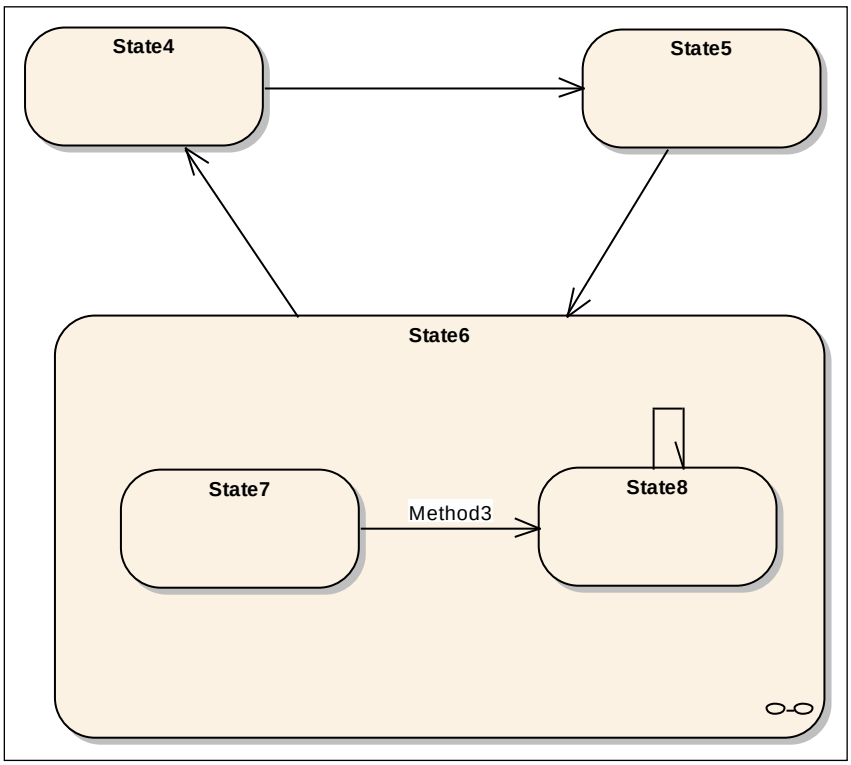

Figure B.2 – Example of a state machine having a sub-machine

B.3 Definition of state machine

The infrastructure of state machines defined in this annex only deals with the basics of state machines needed to support Part 9 and Part 10. The intention is to keep the basic simple but extensible.

For the state machines defined in this annex we assume that state machines are typed and instances of a type have their states and semantics specified by the type. For some types, this means that the states and transitions are fixed. For other types the states and transitions may be dynamic or unknown. A state machine where all the states are specified explicitly by the type is called a finite state machine.

Therefore we distinguish between *StateMachineType* and *StateMachine* and their subtypes like *FiniteStateMachineType*. The *StateMachineType* specifies a description of the state machine, that is, its states, transitions, etc., whereas the *StateMachine* is an instance of the *StateMachineType* and only contains the current state.

Each *StateMachine* contains information about the current state. If the *StateMachineType* has *SubStateMachines*, the *StateMachine* also contains information about the current state of the *SubStateMachines*. *StateMachines* which have their states completely defined by the type are instances of a *FiniteStateMachineType*.

Each *FiniteStateMachineType* has one or more *States*. For simplicity, we do not distinguish between different *States* like the start or the end states.

Each *State* can have one or more *SubStateMachines*.

Each *FiniteStateMachineType* may have one or more *Transitions*. A *Transition* is directed and points from one *State* to another *State*.

Each *Transition* can have one or more *Causes*. A *Cause* leads a *FiniteStateMachine* to change its current *State* from the source of the *Transition* to its target. In this annex we only specify *Method* calls to be *Causes* of *Transitions*. *Transitions* do not have to have a *Cause*. A *Transition* can always be caused by some server-internal logic that is not exposed in the *AddressSpace*.

Each *Transition* can have one or more *Effects*. An *Effect* occurs if the *Transition* is used to change the *State* of a *StateMachine*. In this annex we only specify the generation of *Events* to be *Effects* of a *Transition*. A *Transition* is not required to expose any *Effects* in the *AddressSpace.*

Although this annex only specifies simple concepts for state machines, the provided infrastructure is extensible. If needed, special *States* can be defined as well as additional *Causes* or *Effects*.

B.4 Representation of state machines in the AddressSpace

B.4.1 Overview

The types defined in this annex are illustrated in Figure B.3. The *MyFiniteStateMachineType* is a minimal example which illustrates how these *Types* can be used to describe a *StateMachine*. See Part 9 and Part 10 for additional examples of *StateMachines*.

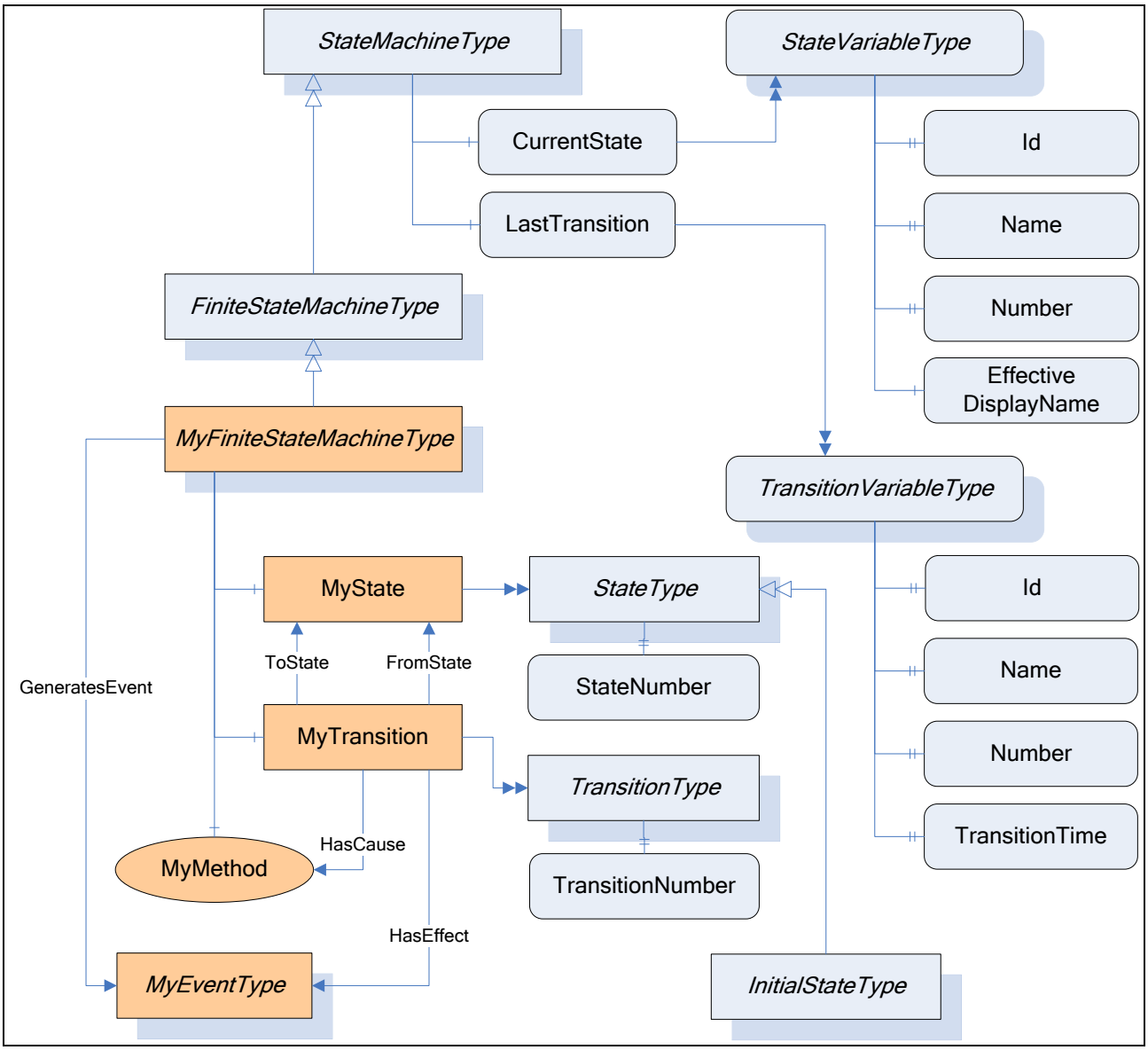

Figure B.3 – The StateMachine Information Model

B.4.2 StateMachineType

The *StateMachineType* is the base *ObjectType* for all *StateMachineTypes*. It defines a single *Variable* which represents the current state of the machine. An instance of this *ObjectType* shall generate an *Event* whenever a significant state change occurs. The *Server* decides which state changes are significant. *Servers* shall use the *GeneratesEvent ReferenceType* to indicate which *Event(s)* could be produced by the *StateMachine*.

Subtypes may add *Methods* which affect the state of the machine. The *Executable Attribute* is used to indicate whether the *Method* is valid given the current state of the machine. The generation of *AuditEvents* for *Methods* is defined in Part 4. A *StateMachine* may not be active. In this case, the *CurrentState* and *LastTransition Variables* shall have a status equal to *Bad_StateNotActive* (see Table B.17).

Subtypes may add components which are instances of *StateMachineTypes*. These components are considered to be sub-states of the *StateMachine*. *SubStateMachines* are only active when the parent machine is in an appropriate state.

Events produced by *SubStateMachines* may be suppressed by the parent machine. In some cases, the parent machine will produce a single *Event* that reflects changes in multiple *SubStateMachines*.

FiniteStateMachineType is subtype of *StateMachineType* that provides a mechanism to explicitly define the states and transitions. A *Server* should use this mechanism if it knows what the possible states are and the state machine is not trivial. The *FiniteStateMachineType* is defined in B.4.5.

The *StateMachineType* is formally defined in Table B.1.

Table B.1 – StateMachineType Definition

Attribute	**Value**				
BrowseName	StateMachineType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType.					
HasSubtype	ObjectType	FiniteStateMachineType	Defined in B.4.5		
HasComponent	Variable	CurrentState	LocalizedText	StateVariableType	Mandatory
HasComponent	Variable	LastTransition	LocalizedText	TransitionVariableType	Optional

CurrentState stores the current state of an instance of the *StateMachineType*. *CurrentState* provides a human readable name for the current state which may not be suitable for use in application control logic. Applications should use the *Id Property* of *CurrentState* if they need a unique identifier for the state.

LastTransition stores the last transition which occurred in an instance of the *StateMachineType*. *LastTransition* provides a human readable name for the last transition which may not be suitable for use in application control logic. Applications should use the *Id Property* of *LastTransition* if they need a unique identifier for the transition.

B.4.3 StateVariableType

The *StateVariableType* is the base *VariableType* for *Variables* that store the current state of a *StateMachine* as a human readable name.

The *StateVariableType* is formally defined in Table B.2.

Table B.2 – StateVariableType Definition

Attribute	**Value**				
BrowseName	StateVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseDataVariableType* defined in 7.4. Note that a *Reference* to this subtype is not shown in the definition of the *BaseDataVariableType*.					
HasSubtype	VariableType	FiniteStateVariableType	Defined in B.4.6		
HasProperty	Variable	Id	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Name	QualifiedName	PropertyType	Optional
HasProperty	Variable	Number	UInt32	PropertyType	Optional
HasProperty	Variable	EffectiveDisplayName	LocalizedText	PropertyType	Optional

Id is a name which uniquely identifies the current state within the *StateMachineType*. A subtype may restrict the *DataType*.

Name is a *QualifiedName* which uniquely identifies the current state within the *StateMachineType*.

Number is an integer which uniquely identifies the current state within the *StateMachineType*.

EffectiveDisplayName contains a human readable name for the current state of the state machine after taking the state of any *SubStateMachines* in account. There is no rule specified

for which state or sub-state should be used. It is up to the *Server* and will depend on the semantics of the *StateMachineType*.

StateMachines produce *Events* which may include the current state of a *StateMachine*. In that case *Servers* shall provide all the optional *Properties* of the *StateVariableType* in the *Event*, even if they are not provided on the instances in the *AddressSpace*.

B.4.4 TransitionVariableType

The *TransitionVariableType* is the base *VariableType* for *Variables* that store a *Transition* that occurred within a *StateMachine* as a human readable name.

The *SourceTimestamp* for the value specifies when the *Transition* occurred. This value may also be exposed with the *TransitionTime Property*.

The *TransitionVariableType* is formally defined in Table B.3.

Table B.3 – TransitionVariableType Definition

Attribute	**Value**				
BrowseName	TransitionVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseDataVariableType* defined in 7.4. Note that a *Reference* to this subtype is not shown in the definition of the *BaseDataVariableType*.					
HasSubtype	VariableType	FiniteTransitionVariableType	Defined in B.4.7		
HasProperty	Variable	Id	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Name	QualifiedName	PropertyType	Optional
HasProperty	Variable	Number	UInt32	PropertyType	Optional
HasProperty	Variable	TransitionTime	UtcTime	PropertyType	Optional
HasProperty	Variable	EffectiveTransitionTime	UtcTime	PropertyType	Optional

Id is a name which uniquely identifies a *Transition* within the *StateMachineType*. A subtype may restrict the *DataType*.

Name is a *QualifiedName* which uniquely identifies a transition within the *StateMachineType*.

Number is an integer which uniquely identifies a transition within the *StateMachineType*.

TransitionTime specifies when the transition occurred.

EffectiveTransitionTime specifies the time when the current state or one of its substates was entered. If, for example, a StateA is active and – while active – switches several times between its substates SubA and SubB, then the *TransitionTime* stays at the point in time where StateA became active whereas the *EffectiveTransitionTime* changes with each change of a substate.

B.4.5 FiniteStateMachineType

The *FiniteStateMachineType* is the base *ObjectType* for *StateMachines* that explicitly define the possible *States* and *Transitions*. Once the *States* and *Transitions* are defined subtypes shall not add new *States* and *Transitions* (see B.4.18). *Subtypes* may add causes or effects.

The *States* of the machine are represented with instances of the *StateType ObjectType.* Each *State* shall have a *BrowseName* which is unique within the *StateMachine* and shall have a *StateNumber* which shall also be unique across all *States* defined in the *StateMachine*. Be aware that *States* in a *SubStateMachine* may have the same *StateNumber* or *BrowseName* as *States* in the parent machine. A concrete subtype of *FiniteStateMachineType* shall define at least one *State*.

A *StateMachine* may define one *State* which is an instance of the *InitialStateType*. This *State* is the *State* that the machine goes into when it is activated.

The *Transitions* that may occur are represented with instances of the *TransitionType*. Each *Transition* shall have a *BrowseName* which is unique within the *StateMachine* and may have a *TransitionNumber* which shall also be unique across all *Transitions* defined in the *StateMachine*.

The initial *State* for a *Transition* is a *StateType Object* which is the target of a *FromState Reference*. The final *State* for a *Transition* is a *StateType Object* which is the target of a *ToState Reference*. The *FromState* and *ToState References* shall always be specified.

A *Transition* may produce an *Event*. The *Event* is indicated by a *HasEffect Reference* to a subtype of *BaseEventType*. The *StateMachineType* shall have *GeneratesEvent References* to the targets of a *HasEffect Reference* for each of its *Transitions*.

A *FiniteStateMachineType* may define *Methods* that cause a transition to occur. These *Methods* are targets of *HasCause References* for each of the *Transitions* that may be triggered by the *Method*. The *Executable Attribute* for a *Method* is used to indicate whether the current *State* of the machine allows the *Method* to be called.

A *FiniteStateMachineType* may have sub-state-machines which are represented as instances of *StateMachineType ObjectTypes*. Each *State* shall have a *HasSubStateMachine Reference* to the *StateMachineType Object* which represents the child *States*. The *SubStateMachine* is not active if the parent *State* is not active. In this case the *CurrentState* and *LastTransition Variables* of the *SubStateMachine* shall have a status equal to *Bad_StateNotActive* (see Table B.17).

The *FiniteStateMachineType* is formally defined in Table B.4.

Table B.4 – FiniteStateMachineType Definition

Attribute	**Value**				
BrowseName	FiniteStateMachineType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the StateMachineType defined in 6.2.					
HasComponent	Variable	CurrentState	LocalizedText	FiniteStateVariableType	Mandatory
HasComponent	Variable	LastTransition	LocalizedText	FiniteTransitionVariableType	Optional
HasComponent	Variable	AvailableStates	NodeId[]	BaseDataVariableType	Optional
HasComponent	Variable	AvailableTransitions	NodeId[]	BaseDataVariableType	Optional

In some *Servers* an instance of a StateMachine may restrict the *States* and / or *Transitions* that are available. These restrictions may result from the internal design of the instance. For example the *StateMachine* for an instrument's limit alarm which only supports Hi and HiHi and can not produce a Low or LowLow. An instance of a *StateMachine* may also dynamically change the available *States* and/or *Transitions* based on its operating mode. For example when a piece of equipment is in a maintenance mode the available *States* may be limited to some subset of the *States* available during normal operation.

The *AvailableStates Variable* provides a *NodeId* list of the *States* that are present in the *StateMachine* instance. The list may change during operation of the *Server*.

The *AvailableTransitions Variable* provides a *NodeId* list of the *Transitions* that are present in the *StateMachine* instance. The list may change during operation of the *Server*.

An example of a FiniteStateMachine type is shown in Figure B.4 below.

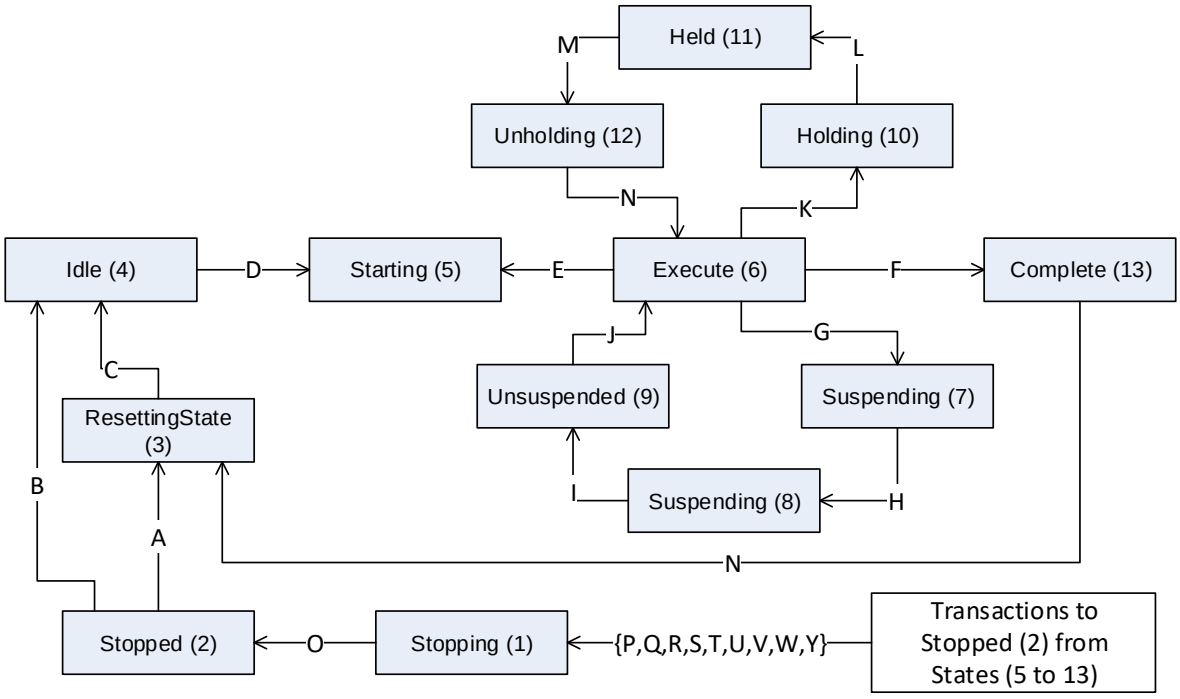

Figure B.4 – Example of a FiniteStateMachine type

An example instance of the type is shown in Figure B.5. In this example the *States* {7,8,9} and the *Transitions* {G,H,I,J} are not available in this instance.

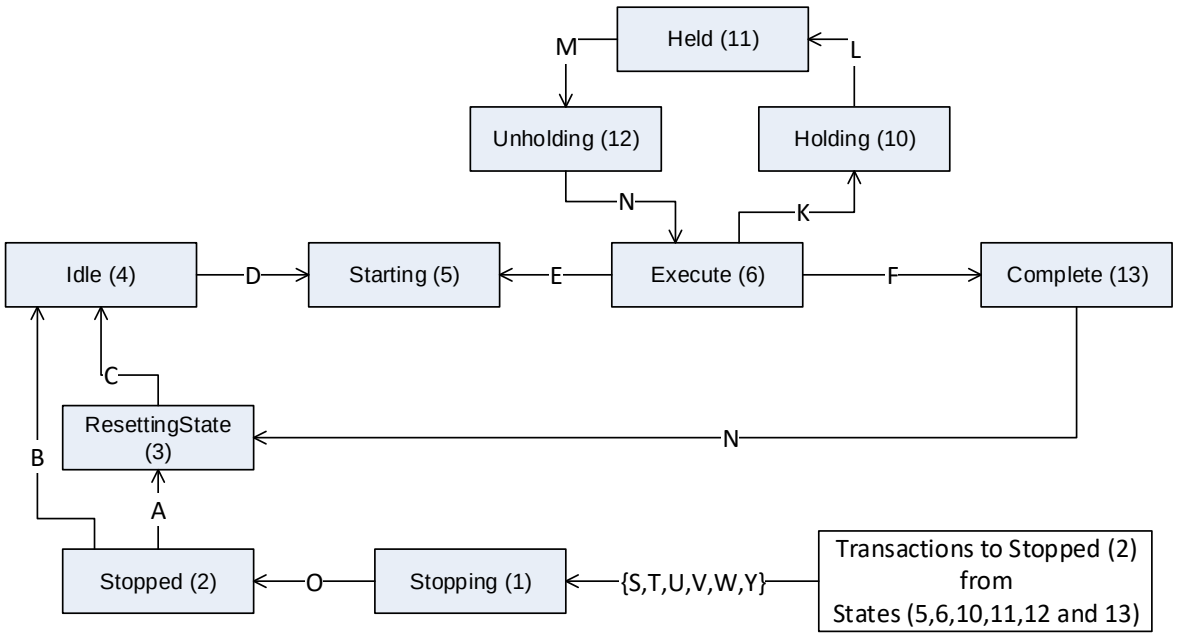

Figure B.5 – Example of a FiniteStateMachine instance

B.4.6 FiniteStateVariableType

The *FiniteStateVariableType* is a subtype of *StateVariableType* and is used to store the current state of a Finite*StateMachine* as a human readable name.

The *FiniteStateVariableType* is formally defined in Table B.5.

Table B.5 – FiniteStateVariableType Definition

Attribute	Value				
BrowseName	FiniteStateVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *StateVariableType* defined in B.4.3					
HasProperty	Variable	Id	NodeId	PropertyType	Mandatory

Id is inherited from the *StateVariableType* and overridden to reflect the required *DataType*. This value shall be the *NodeId* of one of the *State Objects* of the *FiniteStateMachineType.*

The *Name Property* is inherited from *StateVariableType.* Its *Value* shall be the *BrowseName* of one of the *State Objects* of the *FiniteStateMachineType*.

The *Number Property* is inherited from *StateVariableType.* Its *Value* shall be the *StateNumber* for one of the *State Objects* of the *FiniteStateMachineType*.

B.4.7 FiniteTransitionVariableType

The *FiniteTransitionVariableType* is a subtype of *TransitionVariableType* and is used to store a *Transition* that occurred within a *FiniteStateMachine* as a human readable name.

The *FiniteTransitionVariableType* is formally defined in Table B.6.

Table B.6 – FiniteTransitionVariableType Definition

Attribute	Value				
BrowseName	FiniteTransitionVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TransitionVariableType defined inB.4.4. Note that a *Reference* to this subtype is not shown in the definition of the *BaseDataVariableType*.					
HasProperty	Variable	Id	NodeId	PropertyType	Mandatory

Id is inherited from the *TransitionVariableType* and overridden to reflect the required *DataType*. This value shall be the *NodeId* of one of the *Transition Objects* of the *FiniteStateMachineType*.

The *Name Property* is inherited from the *TransitionVariableType.* Its *Value* shall be the *BrowseName* of one of the *Transition Objects* of the *FiniteStateMachineType*.

The *Number Property* is inherited from the *TransitionVariableType.* Its *Value* shall be the *TransitionNumber* for one of the *Transition Objects* of the *FiniteStateMachineType*.

B.4.8 StateType

States of a *FiniteStateMachine* are represented as *Objects* of the *StateType.*

The *StateType* is formally defined in Table B.7.

Table B.7 – StateType Definition

Attribute	Value				
BrowseName	StateType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType.					
HasProperty	Variable	StateNumber	UInt32	PropertyType	Mandatory
HasSubtype	ObjectType	InitialStateType	Defined in B.4.9		

B.4.9 InitialStateType

The *InitialStateType* is a subtype of the *StateType* and is formally defined in Table B.8. An *Object* of the *InitialStateType* represents the *State* that a *FiniteStateMachine* enters when it is activated. Each *FiniteStateMachine* can have at most one *State* of type *InitialStateType*, but a *FiniteStateMachine* does not have to have a *State* of this type.

A *SubStateMachine* goes into its initial state whenever the parent state is entered. However, a state machine may define a transition that goes directly to a state of the *SubStateMachine*. In this case the *SubStateMachine* goes into that *State* instead of the initial *State*. The two scenarios are illustrated in Figure B.6. The transition from State5 to State6 causes the *SubStateMachine* to go into the initial *State* (State7), however, the transition from State4 to State8 causes the parent machine to go to State6 and the *SubStateMachine* will go to State8.

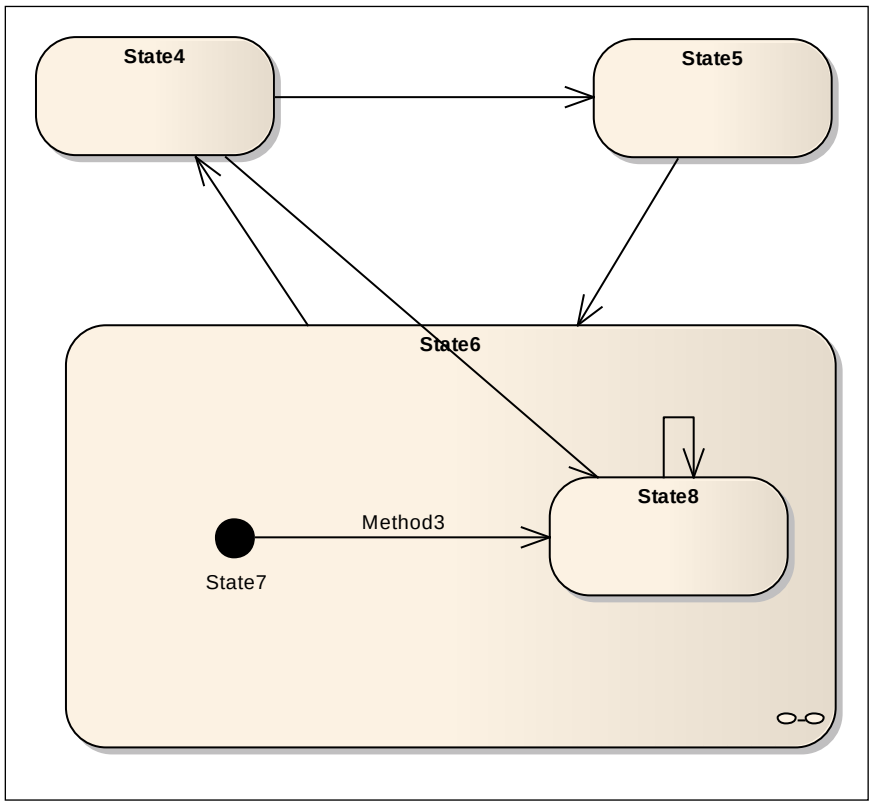

Figure B.6 – Example of an initial State in a sub-machine

If no initial state for a *SubStateMachine* exists and the *State* having the *SubStateMachine* is entered directly, then the *State* of the *SubStateMachine* is server-specific.

Table B.8 – InitialStateType Definition

Attribute	Value				
BrowseName	InitialStateType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *StateType* defined in B.4.8					

B.4.10 TransitionType

Transitions of a *FiniteStateMachine* are represented as *Objects* of the *ObjectType TransitionType* formally defined in Table B.9.

Each valid *Transition* shall have exactly one *FromState Reference* and exactly one *ToState Reference*, each pointing to an *Object* of the *ObjectType StateType*.

Each *Transition* can have one or more *HasCause References* pointing to the cause that triggers the *Transition*.

Each *Transition* can have one or more *HasEffect References* pointing to the effects that occur when the *Transition* was triggered.

Table B.9 – TransitionType Definition

Attribute	Value				
BrowseName	TransitionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Mandatory

B.4.11 FromState

The *FromState ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to the starting *State* the *Transition* connects.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* of this *ReferenceType* shall be an *Object* of the *ObjectType StateType* or one of its subtypes.

The representation of the *FromState ReferenceType* in the *AddressSpace* is specified in Table B.10.

Table B.10 – FromState ReferenceType

Attributes	Value		
BrowseName	FromState		
InverseName	ToTransition		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.12 ToState

The *ToState ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to the ending *State* the *Transition* connects.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* of this *ReferenceType* shall be an *Object* of the *ObjectType StateType* or one of its subtypes.

References of this *ReferenceType* may be only exposed uni-directional. Sometimes this is required, for example, if a *Transition* points to a *State* of a sub-machine.

The representation of the *ToState ReferenceType* in the *AddressSpace* is specified in Table B.11.

Table B.11 – ToState ReferenceType

Attributes	**Value**		
BrowseName	ToState		
InverseName	FromTransition		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.13 HasCause

The *HasCause ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point from a *Transition* to something that causes the *Transition*. In this annex we only define *Methods* as *Causes*. However, the *ReferenceType* is not restricted to point to *Methods*. The referenced Methods can, but do not have to point to a Method of the StateMachineType. For example, it is allowed to point to a server-wide restart Method leading the state machine to go into its initial state.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* can be of any *NodeClass*.

The representation of the *HasCause ReferenceType* in the *AddressSpace* is specified in Table B.12.

Table B.12 – HasCause ReferenceType

Attributes	**Value**		
BrowseName	HasCause		
InverseName	MayBeCausedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.14 HasEffect

The *HasEffect ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to something that will be effected when the *Transition* is triggered. In this annex we only define *EventTypes* as *Effects*. However, the *ReferenceType* is not restricted to point to *EventTypes*.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* can be of any *NodeClass*.

The representation of the *HasEffect ReferenceType* in the *AddressSpace* is specified in Table B.13.

Table B.13 – HasEffect ReferenceType

Attributes	Value		
BrowseName	HasEffect		
InverseName	MayBeEffectedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.15 HasSubStateMachine

The *HasSubStateMachine ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point from a *State* to an instance of a *StateMachineType* which represents the sub-states for the *State*.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType StateType*. The *TargetNode* shall be an *Object* of the *ObjectType StateMachineType* or one of its subtypes. Each *Object* can be the *TargetNode* of at most one *HasSubStateMachine Reference*.

The *SourceNode* (the state) and the *TargetNode* (the *SubStateMachine*) shall belong to the same *StateMachine*, that is, both shall be referenced from the same *Object* of type *StateMachineType* using a *HasComponent Reference* or a subtype of *HasComponent*.

The representation of the *HasSubStateMachine ReferenceType* in the *AddressSpace* is specified in Table B.14.

Table B.14 – HasSubStateMachine ReferenceType

Attributes	Value		
BrowseName	HasSubStateMachine		
InverseName	SubStateMachineOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.16 TransitionEventType

The *TransitionEventType* is a subtype of the *BaseEventType*. It can be used to generate an *Event* identifying that a *Transition* of a *StateMachine* was triggered. It is formally defined in Table B.15.

Table B.15 – TransitionEventType

Attribute	Value				
BrowseName	TransitionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the base *BaseEventType* defined in 6.4.2					
HasComponent	Variable	Transition	LocalizedText	TransitionVariableType	Mandatory
HasComponent	Variable	FromState	LocalizedText	StateVariableType	Mandatory
HasComponent	Variable	ToState	LocalizedText	StateVariableType	Mandatory

The *TransitionEventType* inherits the *Properties* of the *BaseEventType*.

The inherited *Property SourceNode* shall be filled with the *NodeId* of the *StateMachine* instance where the *Transition* occurs. If the *Transition* occurs in a *SubStateMachine*, then the *NodeId* of the *SubStateMachine* has to be used. If the Transition occurs between a *StateMachine* and a *SubStateMachine*, then the *NodeId* of the *StateMachine* has to be used, independent of the direction of the *Transition*.

Transition identifies the *Transition* that triggered the *Event*.

FromState identifies the *State* before the *Transition*.

ToState identifies the *State* after the *Transition*.

B.4.17 AuditUpdateStateEventType

The *AuditUpdateStateEventType* is a subtype of the *AuditUpdateMethodEventType*. It can be used to generate an *Event* identifying that a *Transition* of a *StateMachine* was triggered. It is formally defined in Table B.16.

Table B.16 – AuditUpdateStateEventType

Attribute	**Value**				
BrowseName	AuditUpdateStateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateMethodEventType* defined in 6.4.27					
HasProperty	Variable	OldStateId	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	NewStateId	BaseDataType	PropertyType	Mandatory

The *AuditUpdateStateEventType* inherits the *Properties* of the *AuditUpdateMethodEventType*.

The inherited *Property SourceNode* shall be filled with the *NodeId* of the *StateMachine* instance where the *State* changed. If the *State* changed in a *SubStateMachine*, then the *NodeId* of the *SubStateMachine* has to be used.

The *SourceName* for *Events* of this type should be the effect that generated the event (e.g. the name of a Method). If the effect was generated by a *Method* call, the *SourceName* should be the name of the *Method* prefixed with “Method/”.

OldStateId reflects the *Id* of the state prior the change.

NewStateId reflects the new *Id* of the state after the change.

B.4.18 Special Restrictions on subtyping StateMachines

In general, all rules on subtyping apply for *StateMachine* types as well. Some additional rules apply for *StateMachine* types. If a StateMachine type is not abstract, subtypes of it shall not change the behaviour of it. That means, that in this case a subtype shall not add *States* and it shall not add *Transitions* between its *States*. However, a subtype may add *SubStateMachines*, it may add *Transitions* from the *States* to the *States* of the *SubStateMachine*, and it may add *Causes* and *Effects* to a *Transition*. In addition, a subtype of a *StateMachine* type shall not remove *States* or *Transitions*.

B.4.19 Specific StatusCodes for StateMachines

In Table B.17 specific *StatusCodes* used for *StateMachines* are defined.

Table B.17 – Specific StatusCodes for StateMachines

Symbolic Id	**Description**
Bad_StateNotActive	The accessed state is not active.

B.5 Examples of StateMachines in the AddressSpace

B.5.1 StateMachineType using inheritance

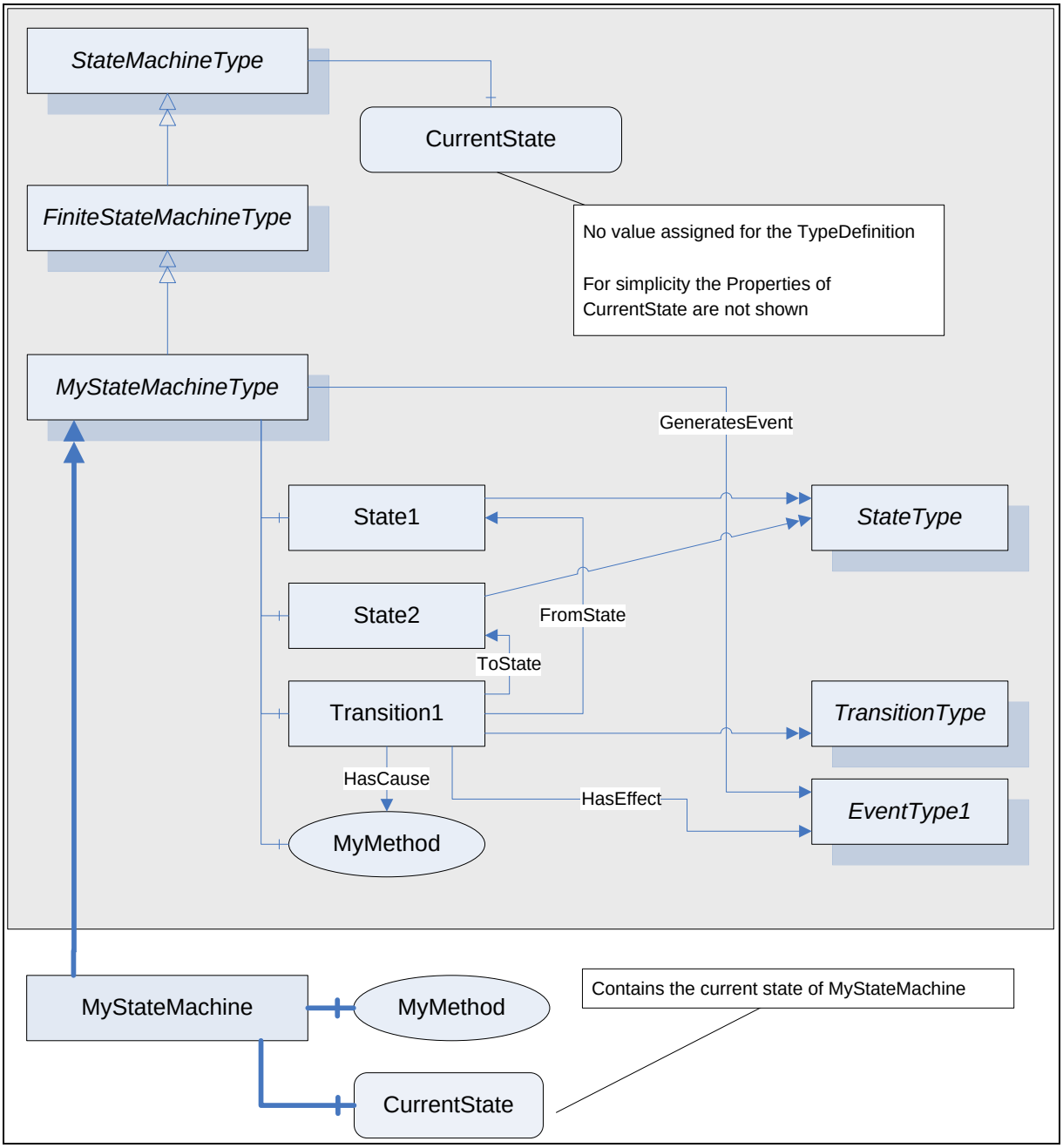

Figure B.7 – Example of a StateMachineType using inheritance

In Figure B.7 an example of a *StateMachine* is given using the Notation defined in Part 3. First, a new *StateMachineType* is defined, called "MyStateMachineType", inheriting from the base *FiniteStateMachineType*. It contains two *States*, "State1" and "State2" and a *Transition* "Transition1" between them. The *Transition* points to a *Method* "MyMethod" as the *Cause* of the *Transition* and an *EventType* "EventType1" as the *Effect* of the *Transition*.

Instances of "MyStateMachineType" can be created, for example "MyStateMachine". It has a *Variable* "CurrentState" representing the current *State*. The "MyStateMachine" *Object* only includes the *Nodes* which expose information specific to the instance.

B.5.2 StateMachineType with a sub-machine using inheritance

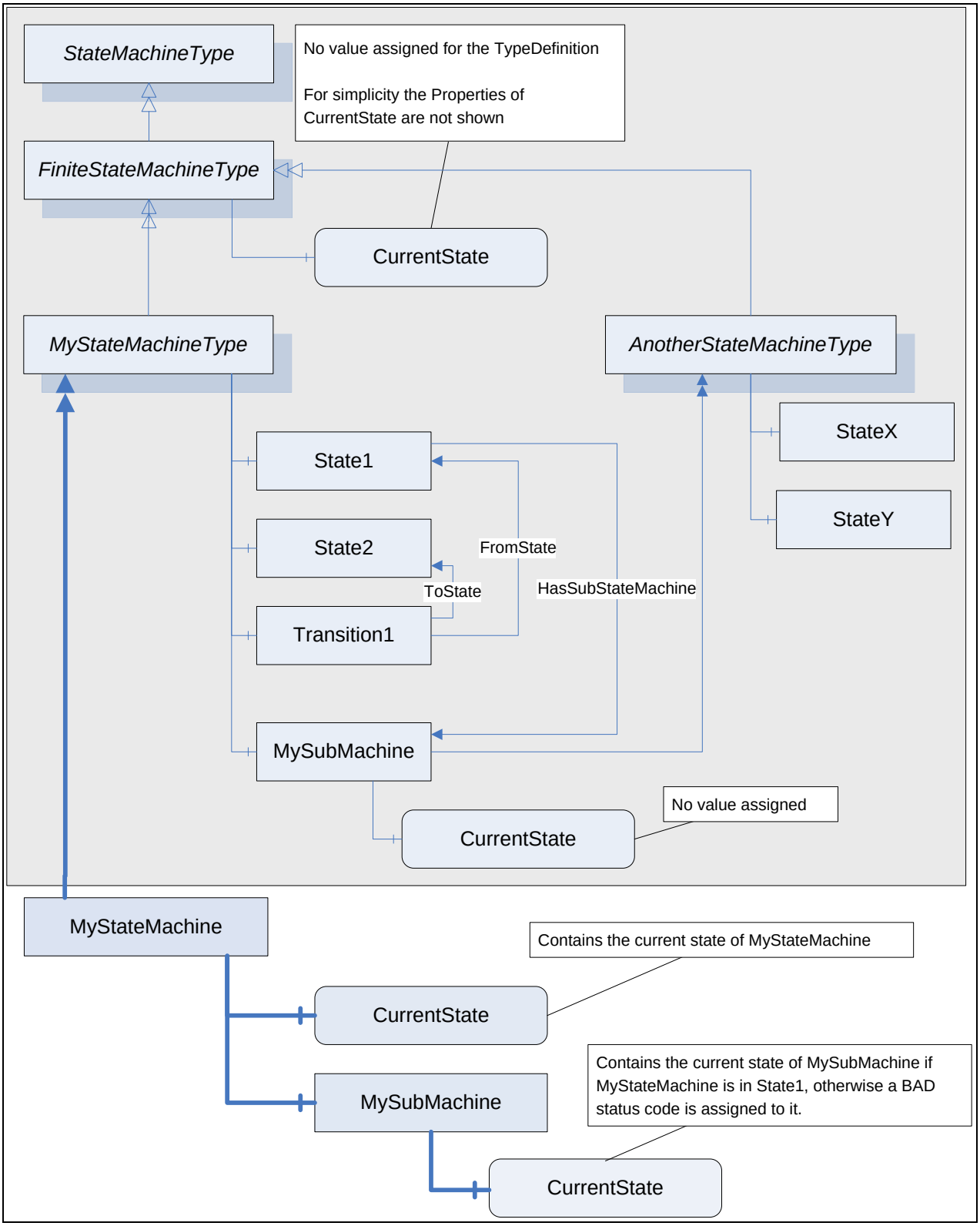

Figure B.8 – Example of a StateMachineType with a SubStateMachine using inheritance

Figure B.8 gives an example of a *StateMachineType* having a *SubStateMachine* for its "State1". For simplicity no effects and causes are shown, as well as type information for the *States* or *ModellingRules*.

The "MyStateMachineType" contains an *Object* "MySubMachine" of type "AnotherStateMachineType" representing a *SubStateMachine*. The "State1" references this

Object with a *HasSubStateMachine Reference*, thus it is a *SubStateMachine* of “State1”. Since “MySubMachine” is an *Object* of type “AnotherStateMachineType” it has a *Variable* representing the current *State*. Since it is used as an *InstanceDeclaration*, no value is assigned to this *Variable*.

An *Object* of “MyStateMachineType”, called “MyStateMachine” has *Variables* for the current *State*, but also has an *Object* “MySubMachine” and a *Variable* representing the current state of the *SubStateMachine*. Since the *SubStateMachine* is only used when “MyStateMachine” is in “State1”, a client would receive a *Bad_StateNotActive StatusCode* when reading the *SubStateMachine CurrentState Variable* if “MyStateMachine” is in a different *State*.

B.5.3 StateMachineType using containment

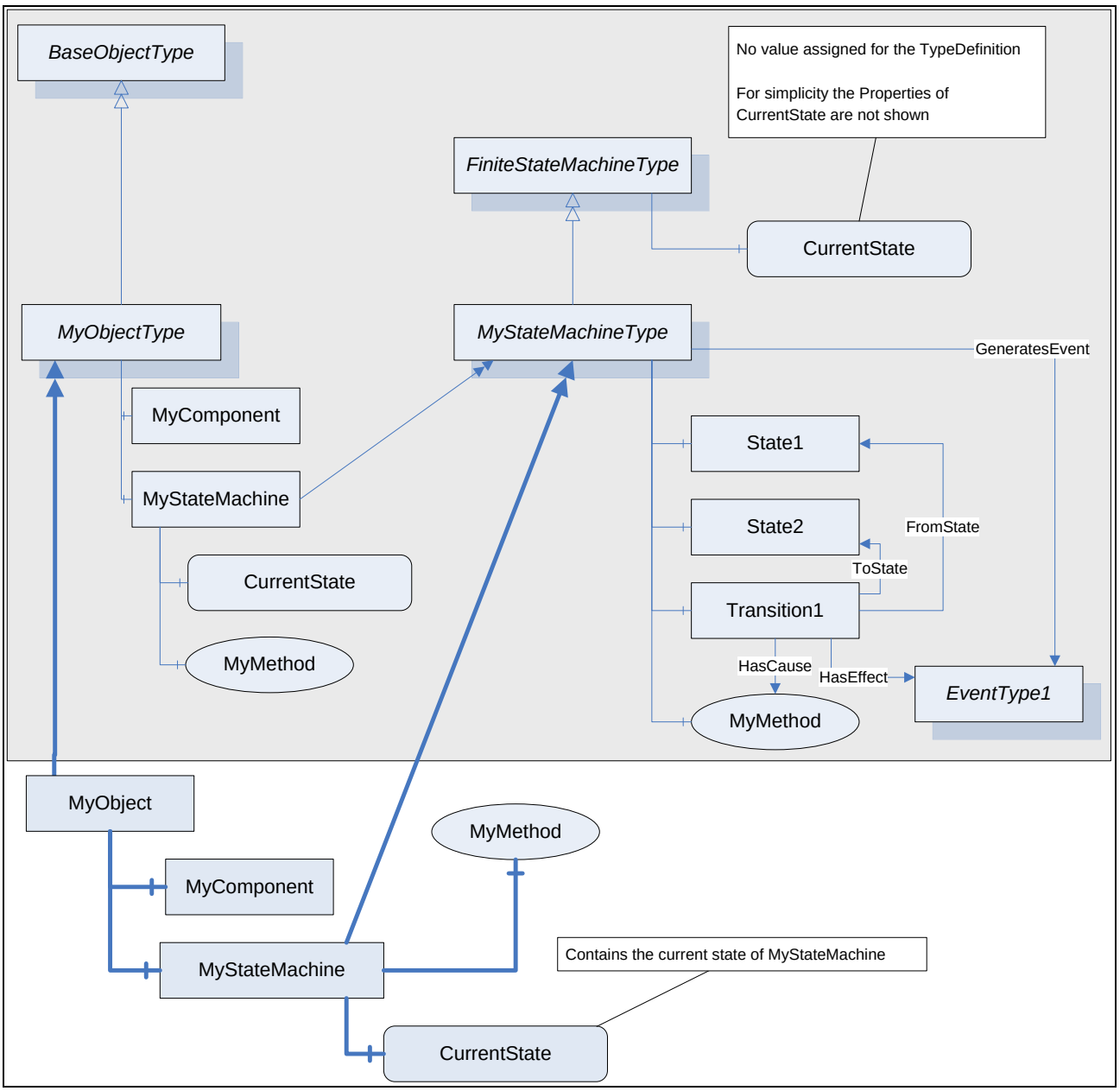

Figure B.9 – Example of a StateMachineType using containment

Figure B.9 gives an example of an *ObjectType* not only representing a *StateMachine* but also having some other functionality. The *ObjectType* “MyObjectType” has an *Object* “MyComponent” representing this other functionality. But it also contains a *StateMachine* “MyStateMachine” of the type “MyStateMachineType”. *Objects* of “MyObjectType” also contain such an *Object* representing the StateMachine and a *Variable* containing the current state of the StateMachine, as shown in the Figure.

B.5.4 Example of a StateMachine having Transition to SubStateMachine

The *StateMachines* shown so far only had *Transitions* between *States* on the same level, that is, on the same *StateMachine*. Of cause, it is possible and often required to have *Transitions* between *States* of the *StateMachine* and *States* of its *SubStateMachine*.

Because a *SubStateMachine* can be defined by another *StateMachineType* and this type can be used in several places, it is not possible to add a bi-directional *Reference* from one of the shared *States* of the *SubStateMachine* to another *StateMachine*. In this case it is suitable to expose the *FromState* or *ToState References* uni-directional, that is, only pointing from the *Transition* to the *State* and not being able to browse to the other direction. If a *Transition* points from a *State* of a *SubStateMachine* to a *State* of another sub-machine, both, the *FromState* and the *ToState Reference*, are handled uni-directional.

A Client shall be able to handle the information of a *StateMachine* if the *ToState* and *FromState References* are only exposed as forward *References* and the inverse *References* are omitted.

Figure B.10 gives an example of a state machine having a transition from a sub-state to a state.

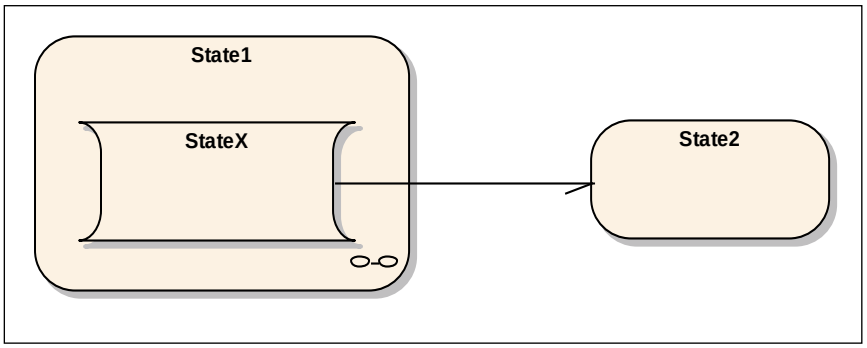

Figure B.10 – Example of a state machine with transitions from sub-states

In Figure B.11, the representation of this example as *StateMachineType* in the *AddressSpace* is given. The “Transition1”, part of the definition of “MyStateMachineType”, points to the “StateX” of the *StateMachineType* “AnotherStateMachineType”. The *Reference* is only exposed as forward *Reference* and the inverse *Reference* is omitted. Thus, there is no *Reference* from the “StateX” of “AnotherStateMachineType” to any part of “MyStateMachineType” and “AnotherStateMachineType” can be used in other places as well.

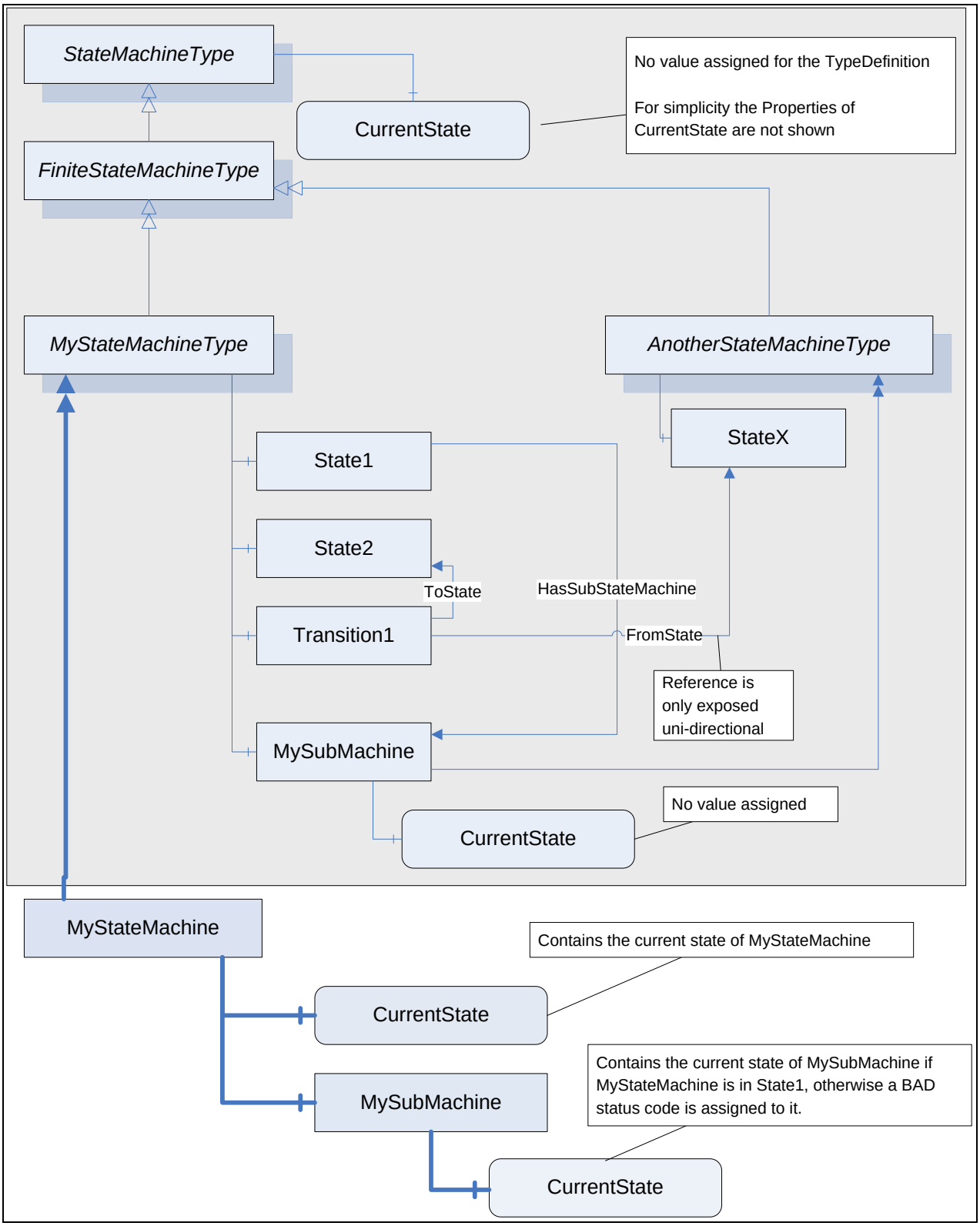

Figure B.11 – Example of a StateMachineType having Transition to SubStateMachine

Annex C
(normative)

File Transfer

C.1 Overview

This annex describes an information model for file transfer. Files could be modelled in OPC UA as simple Variables using ByteStrings. However, the overall message size in OPC UA is limited due to resources and security issues (denial of service attacks). Only accessing parts of the array can lead to concurrency issues if one client is reading the array while others are manipulating it. Therefore the *ObjectType FileType* is defined representing a file with *Methods* to access the file. The life-cycle of a file stored on a hard disk and an instance of the *FileType* representing the file in an OPC UA *AddressSpace* can be independent.

In addition to representing individual files this annex also defines a way to represent a whole file system or a part of a file system. This can be done using the *FileDirectoryType* in combination with the *FileType*. The *FileDirectoryType* provides *Methods* to create delete and move files and directories. The root of a file system or part of a file system is represented by an instance of the *FileDirectoryType* with the *BrowseName FileSystem*. All directories below the root directory are represented by instances of the *FileDirectoryType* or a subtype. All files below the root directory are represented by instances of the *FileType* or a subtype.

In different situations like transfer of configuration files or firmware update, the files are temporary and an additional handshake is necessary to create the file for reading or to apply the file after writing it to the server. This use case is covered by the *TemporaryFileTransferType* defined in this annex.

This annex is an integral part of this standard, that is, the types defined in this annex have to be used as defined. However, it is not required but strongly recommended that a *Server* uses these types to expose its files. The defined types may be subtyped to refine their behaviour.

C.2 FileType

This *ObjectType* defines a type for files. It is formally defined in Table C.1.

Table C.1 – FileType

Attribute	**Value**				
BrowseName	FileType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	Size	UInt64	PropertyType	Mandatory
HasProperty	Variable	Writable	Boolean	PropertyType	Mandatory
HasProperty	Variable	UserWritable	Boolean	PropertyType	Mandatory
HasProperty	Variable	OpenCount	UInt16	PropertyType	Mandatory
HasProperty	Variable	MimeType	String	PropertyType	Optional
HasComponent	Method	Open	Defined in C.2.1		Mandatory
HasComponent	Method	Close	Defined in C.2.2		Mandatory
HasComponent	Method	Read	Defined in C.2.3		Mandatory
HasComponent	Method	Write	Defined in C.2.4		Mandatory
HasComponent	Method	GetPosition	Defined in C.2.5		Mandatory
HasComponent	Method	SetPosition	Defined in C.2.6		Mandatory

Size defines the size of the file in Bytes. When a file is opened for write the size might not be accurate.

Writable indicates whether the file is writable. It does not take any user access rights into account, i.e. although the file is writable this may be restricted to a certain user / user group.

The *Property* does not take into account whether the file is currently opened for writing by another client and thus currently locked and not writable by others.

UserWritable indicates whether the file is writable taking user access rights into account. The Property does not take into account whether the file is currently opened for writing by another client and thus currently locked and not writable by others.

OpenCount indicates the number of currently valid file handles on the file.

The optional *Property MimeType* contains the media type of the file based on RFC 2046.

Note that all *Methods* on a file require a fileHandle, which is returned in the *Open Method*.

C.2.1 Open

Open is used to open a file represented by an *Object* of FileType. When a client opens a file it gets a file handle that is valid while the session is open. Clients shall use the Close *Method* to release the handle when they do not need access to the file anymore. Clients can open the same file several times for read. A request to open for writing shall return Bad_NotWritable when the file is already opened. A request to open for reading shall return Bad_NotReadable when the file is already opened for writing.

Signature

```
Open(
    [in] Byte mode
    [out] UInt32 fileHandle
  );
```

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>mode</td><td>Indicates whether the file should be opened only for read operations or for read and write operations and where the initial position is set.
The mode is an 8-bit unsigned integer used as bit mask with the structure defined in the following table:
<table>
<tr><th>Field</th><th>Bit</th><th>Description</th></tr>
<tr><td>Read</td><td>0</td><td>The file is opened for reading. If this bit is not set the Read Method cannot be executed.</td></tr>
<tr><td>Write</td><td>1</td><td>The file is opened for writing. If this bit is not set the Write Method cannot be executed.</td></tr>
<tr><td>EraseExisting</td><td>2</td><td>This bit can only be set if the file is opened for writing (Write bit is set). The existing content of the file is erased and an empty file is provided.</td></tr>
<tr><td>Append</td><td>3</td><td>When the Append bit is set the file is opened at end of the file, otherwise at begin of the file. The SetPosition Method can be used to change the position.</td></tr>
<tr><td>Reserved</td><td>4:7</td><td>Reserved for future use. Shall always be zero.</td></tr>
</table>
</td></tr>
<tr><td>fileHandle</td><td>A handle for the file used in other method calls indicating not the file (this is done by the Object of the Method call) but the access request and thus the position in the file. The fileHandle is generated by the server and is unique for the Session. Clients cannot transfer the fileHandle to another Session but need to get a new fileHandle by calling the Open Method.</td></tr>
</table>

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotReadable	See Part 4 for a general description. File might be locked and thus not readable.
Bad_NotWritable	See Part 4 for a general description.
Bad_InvalidState	See Part 4 for a general description. The file is locked and thus not writable.
Bad_InvalidArgument	See Part 4 for a general description. Mode setting is invalid.
Bad_NotFound	See Part 4 for a general description.
Bad_UnexpectedError	See Part 4 for a general description.

Table C.2 specifies the *AddressSpace* representation for the *Open Method*.

Table C.2 – Open Method AddressSpace Definition

Attribute	Value				
BrowseName	Open				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.2.2 Close

Close is used to close a file represented by a FileType. When a client closes a file the handle becomes invalid.

Signature

```
Close(
    [in] UInt32 fileHandle
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See Part 4 for a general description. Invalid file handle in call.

Table C.3 specifies the *AddressSpace* representation for the *Close Method*.

Table C.3 – Close Method AddressSpace Definition

Attribute	Value				
BrowseName	Close				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

C.2.3 Read

Read is used to read a part of the file starting from the current file position. The file position is advanced by the number of bytes read.

Signature

```
Read(
    [in] UInt32 fileHandle
    [in] Int32 length
    [out] ByteString data
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
Length	Defines the length in bytes that should be returned in data, starting from the current position of the file handle. If the end of file is reached all data until the end of the file is returned. The *Server* is allowed to return less data than specified length. Only positive values are allowed.
Data	Contains the returned data of the file. If the ByteString is empty it indicates that the end of the file is reached.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See Part 4 Invalid file handle in call or non-positive length.
Bad_UnexpectedError	See Part 4 for a general description.
Bad_InvalidState	See Part 4 for a general description. File was not opened for read access.

Table C.4 specifies the *AddressSpace* representation for the *Read Method*.

Table C.4 – Read Method AddressSpace Definition

Attribute	Value				
BrowseName	Read				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.2.4 Write

Write is used to write a part of the file starting from the current file position. The file position is advanced by the number of bytes written.

Signature

```
Write(
    [in] UInt32 fileHandle
    [in] ByteString data
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
data	Contains the data to be written at the position of the file. It is server-dependent whether the written data are persistently stored if the session is ended without calling the Close Method with the fileHandle. Writing an empty or null *ByteString* returns a Good result code without any affect on the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See Part 4 for a general description. Invalid file handle in call.
Bad_NotWritable	See Part 4 for a general description. File might be locked and thus not writable.
Bad_InvalidState	See Part 4 for a general description. File was not opened for write access.

Table C.5 specifies the *AddressSpace* representation for the *Write Method*.

Table C.5 – Write Method AddressSpace Definition

Attribute	Value				
BrowseName	Write				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

C.2.5 GetPosition

GetPosition is used to provide the current position of the file handle.

Signature

```
GetPosition(
    [in] UInt32 fileHandle
    [out] UInt64 position
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
Position	The position of the fileHandle in the file. If a Read or Write is called it starts at that position.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See Part 4 for a general description. Invalid file handle in call.

Table C.6 specifies the *AddressSpace* representation for the *GetPosition Method*.

Table C.6 – GetPosition Method AddressSpace Definition

Attribute	Value				
BrowseName	GetPosition				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.2.6 SetPosition

SetPosition is used to set the current position of the file handle.

Signature

```
SetPosition(
    [in] UInt32 fileHandle
    [in] UInt64 position
   );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
Position	The position to be set for the fileHandle in the file. If a Read or Write is called it starts at that position. If the position is higher than the file size the position is set to the end of the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See Part 4 for a general description. Invalid file handle in call.

Table C.7 specifies the *AddressSpace* representation for the *SetPosition Method*.

Table C.7 – SetPosition Method AddressSpace Definition

Attribute	Value				
BrowseName	SetPosition				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

C.3 File System

C.3.1 FileDirectoryType

This *ObjectType* defines a type for the representation of file directories. It is formally defined in Table C.8.

It is expected that OPC UA *Servers* will create vendor-specific subtypes of the *FileDirectoryType* with additional functionalities like *Methods* for creating symbolic links or

setting access permissions. OPC UA *Clients* providing specialized file transfer user interfaces should be prepared to expose such additional *Methods* to the user.

Table C.8 – FileDirectoryType

Attribute	Value				
BrowseName	FileDirectoryType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the FolderType defined in 6.6.					
Organizes	Object	<FileDirectoryName>		FileDirectoryType	OptionalPlaceholder
Organizes	Object	<FileName>		FileType	OptionalPlaceholder
HasComponent	Method	CreateDirectory	Defined in C.3.3		Mandatory
HasComponent	Method	CreateFile	Defined in C.3.4		Mandatory
HasComponent	Method	Delete	Defined in C.3.5		Mandatory
HasComponent	Method	MoveOrCopy	Defined in C.3.6		Mandatory

Instances of the *ObjectType* contain a list of *FileDirectoryType Objects* representing the subdirectories of the file directory represented by the instance of this *ObjectType*.

Instances of the *ObjectType* contain a list of *FileType Objects* representing the files in the file directory represented by the instance of this *ObjectType*.

C.3.2 FileSystem Object

The support of file directory structures is declared by aggregating an instance of the *FileDirectoryType* with the *BrowseName FileSystem* as illustrated in Figure C.1.

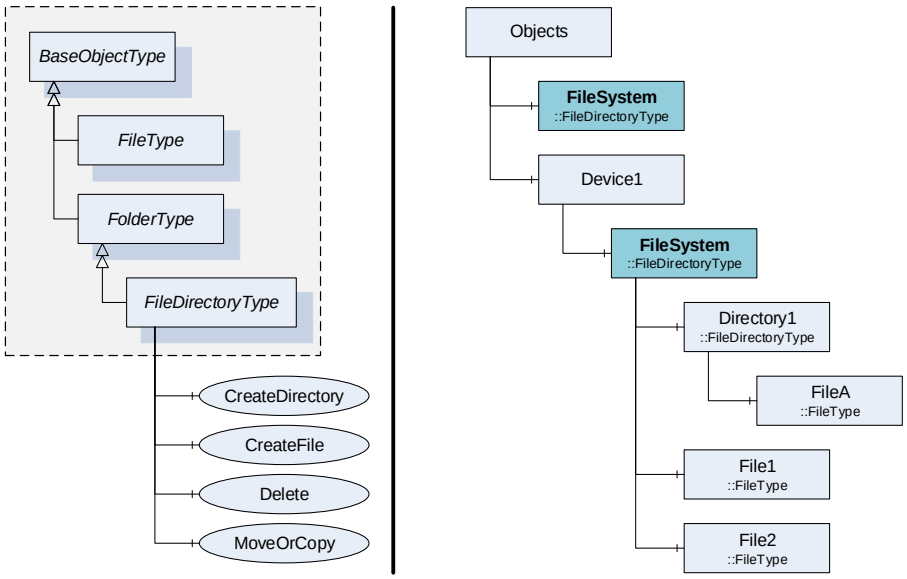

Figure C.1 – FileSystem Example

The *Object* representing the root of a file directory structure shall have the *BrowseName* ***FileSystem***. An OPC UA *Server* may have different *FileSystem Objects* in the *AddressSpace*. *HasComponent* is used to reference a *FileSystem* from aggregating *Objects* like the *Objects Folder* or the *Object* representing a device.

C.3.3 CreateDirectory

CreateDirectory is used to create a new *FileDirectoryType Object* organized by this *Object*.

Signature

```
CreateDirectory(
    [in] String     directoryName
    [out] NodeId    directoryNodeId
```

```
);
```

Argument	Description
directoryName	The name of the directory to create. The name is used for the BrowseName and DisplayName of the directory object and also for the directory in the file system. For the BrowseName, the directoryName is used for the name part of the QualifiedName. The namespace index is Server specific. For the DisplayName, the directoryName is used for the text part of the LocalizedText. The locale part is Server specific.
directoryNodeId	The NodeId of the created directory Object.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_BrowseNameDuplicated	See Part 4 for a general description. A directory with the name already exists.
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.9 specifies the *AddressSpace* representation for the *CreateDirectory Method*.

Table C.9 – CreateDirectory Method AddressSpace Definition

Attribute	Value				
BrowseName	CreateDirectory				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.3.4 CreateFile

CreateFile is used to create a new *FileType Object* organized by this *Object*. The created file can be written using the *Write Method* of the *FileType*.

Signature

```
CreateFile(
    [in] String     fileName
    [in] Boolean    requestFileOpen
    [out] NodeId    fileNodeId
    [out] UInt32    fileHandle
);
```

Argument	Description
fileName	The name of the file to create. The name is used for the BrowseName and DisplayName of the file object and also for the file in the file system. For the BrowseName, the fileName is used for the name part of the QualifiedName. The namespace index is Server specific. For the DisplayName, the fileName is used for the text part of the LocalizedText. The locale part is Server specific.
requestFileOpen	Flag indicating if the new file should be opened with the Write and Read bits set in the open mode after the creation of the file. If the flag is set to True, the file is created and opened for writing. If the flag is set to False, the file is just created.
fileNodeId	The NodeId of the created file Object.
fileHandle	The fileHandle is returned if the requestFileOpen is set to True. The fileNodeId and the fileHandle can be used to access the new file through the FileType Object representing the new file. If requestFileOpen is set to False, the returned value shall be 0 and shall be ignored by the caller.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_BrowseNameDuplicated	See Part 4 for a general description. A file with the name already exists.
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.10 specifies the *AddressSpace* representation for the *CreateFile Method*.

Table C.10 – CreateFile Method AddressSpace Definition

Attribute	Value				
BrowseName	CreateFile				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.3.5 Delete

Delete is used to delete a file or directory organized by this *Object*.

Signature

```
Delete(
    [in] NodeId objectToDelete
  );
```

Argument	Description
objectToDelete	The NodeId of the file or directory to delete. In the case of a directory, all file and directory Objects below the directory to delete are deleted recursively.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	See Part 4 for a general description. A file or directory with the provided NodeId is not organized by this object.
Bad_InvalidState	See Part 4 for a general description. The file or directory is locked and thus cannot be deleted.
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.11 specifies the *AddressSpace* representation for the *Delete Method*.

Table C.11 – Delete Method AddressSpace Definition

Attribute	Value				
BrowseName	Delete				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

C.3.6 MoveOrCopy

MoveOrCopy is used to move or copy a file or directory organized by this *Object* to another directory or to rename a file or directory.

Signature

```
MoveOrCopy(
    [in] NodeId     objectToMoveOrCopy
    [in] NodeId     targetDirectory
    [in] Boolean    createCopy
    [in] String     newName
    [out] NodeId    newNodeId
  );
```

Argument	Description
objectToMoveOrCopy	The NodeId of the file or directory to move or copy.
targetDirectory	The NodeId of the target directory of the move or copy command. If the file or directory is just renamed, the targetDirectory matches the ObjectId passed to the method call.
createCopy	A flag indicating if a copy of the file or directory should be created at the target directory.
newName	The new name of the file or directory in the new location. If the string is empty, the name is unchanged.
newNodeId	The NodeId of the moved or copied object. Even if the Object is moved, the Server may return a new NodeId.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_BrowseNameDuplicated	See Part 4 for a general description. A file or directory with the name already exists.
Bad_NotFound	See Part 4 for a general description. A file or directory with the provided NodeId is not organized by this object.
Bad_InvalidState	See Part 4 for a general description. The file or directory is locked and thus cannot be moved or copied.
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.12 specifies the *AddressSpace* representation for the *MoveOrCopy Method*.

Table C.12 – MoveOrCopy Method AddressSpace Definition

Attribute	Value				
BrowseName	MoveOrCopy				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.4 Temporary File Transfer

C.4.1 TemporaryFileTransferType

This *ObjectType* defines a type for the representation of temporary file transfers. It is formally defined in Table C.13. The *Methods GenerateFileForRead* or *GenerateFileForWrite* generate a temporary *FileType Object* that is not browsable in the *AddressSpace* and can only be accessed with the *NodeId* and *FileHandle* returned by the *Methods* in the same *Session*. This *Object* is used to transfer the temporary file between OPC UA *Client* and *Server*.

Table C.13 – TemporaryFileTransferType

Attribute	Value				
BrowseName	TemporaryFileTransferType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2.					
HasProperty	Variable	ClientProcessingTimeout	Duration	PropertyType	Mandatory
HasComponent	Method	GenerateFileForRead	Defined in C.4.3		Mandatory
HasComponent	Method	GenerateFileForWrite	Defined in C.4.4		Mandatory
HasComponent	Method	CloseAndCommit	Defined in C.4.5		Mandatory
HasComponent	Object	<TransferState>		FileTransferStateMachine Type	OptionalPlaceholder

The *Property ClientProcessingTimeout* defines the maximum time in milliseconds the *Server* accepts between *Method* calls necessary to complete a file read transfer or a file write transfer transaction. This includes the *Method* calls to read or write the file content from the virtual temporary *FileType Object*. If the *Client* exceeds the timeout between *Method* calls, the *Server* may close the file and cancel the corresponding transfer transaction. Any open temporary transfer file shall be deleted if the *Session* used to create the file is no longer valid.

The *TransferState Objects* are used to expose the state of a transfer transaction in the case that the preparation of a file for reading or the processing of the file after writing completes asynchronous after the corresponding *Method* execution. If the transactions are completed

when the *Method* is returned, the optional *TransferState Objects* are not available. A *Server* may allow more than one parallel read transfer. A *Server* may not allow more than one write transfer or a parallel read and writer transfer.

C.4.2 File Transfer Sequences

The sequence of *Method* calls necessary to execute a read file transfer transaction is illustrated in Figure C.2.

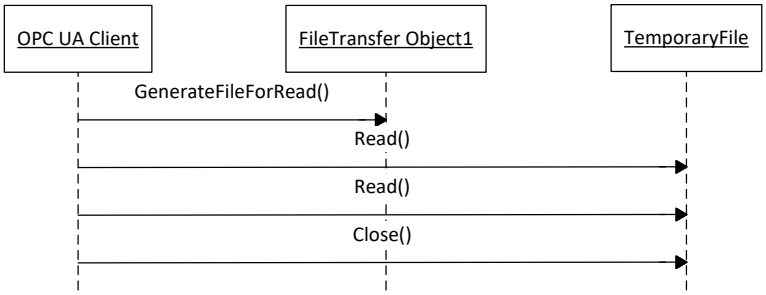

Figure C.2 – Read File Transfer Example Sequence

The read file transfer transaction is started with the Method *GenerateFileForRead* defined by the *TemporaryFileTransferType*. After a successful call of this *Method*, the *Client* reads the file content by calling the *Method Read* defined by the *FileType* until the whole file is transferred from the *Server* to the *Client*. The transaction is completed by calling the *Method Close* defined by the *FileType*.

The sequence of *Method* calls necessary to execute a write file transfer transaction is illustrated in Figure C.3.

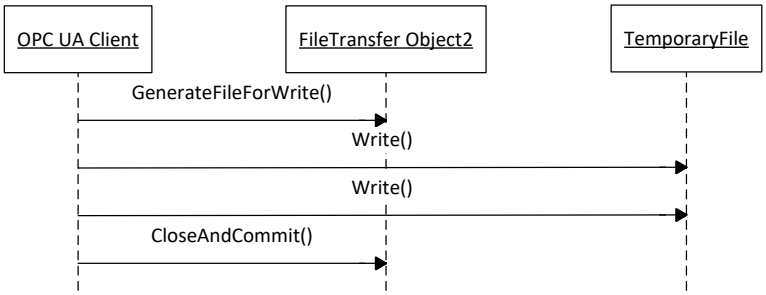

Figure C.3 – Write File Transfer Example Sequence

The write file transfer transaction is started with the *Method StartWriteTransfer* defined by the *TemporaryFileTransferType*. After a successful call of this *Method*, the *Client* writes the file content by calling the *Method Write* defined by the *FileType* until the whole file is transferred from the *Client* to the *Server*. The transaction is completed by calling the *Method*

CloseAndCommit defined by the *TemporaryFileTransferType*. If the *Client* wants to abort the operation it uses the *Close Method* of the temporary *FileType Object*.

C.4.3 GenerateFileForRead

GenerateFileForRead is used to start the read file transaction. A successful call of this *Method* creates a temporary *FileType Object* with the file content and returns the *NodeId* of this *Object* and the file handle to access the *Object*.

Signature

```
GenerateFileForRead(
    [in]  BaseDataType    generateOptions
    [out] NodeId          fileNodeId
    [out] UInt32          fileHandle
    [out] NodeId          completionStateMachine
  );
```

Argument	Description
generateOptions	The optional parameter can be used to specify server specific file generation options. To allow such options, the *Server* shall specify a concrete *DataType* in the *Argument Structure* for this argument in the instance of the *Method*. If the *DataType* is *BaseDataType*, the Client shall pass Null for this argument. Examples for concrete DataTypes are OptionsSet – Used to provide a bit mask for file content selection String – Can be used to provide a string filter or a regular expression Structure – Can be used to provide a structure with create settings e.g. to create a report Enumeration – Can be used to provide a list of options
fileNodeId	NodeId of the temporary file.
fileHandle	The fileHandle of the opened *TransferFile*. The fileHandle can be used to access the *TransferFile Methods Read* and *Close*.
completionStateMachine	If the creation of the file is completed asynchronous, the parameter returns the NodeId of the corresponding *FileTransferStateMachineType Object*. If the creation of the file is already completed, the parameter is null. If a *FileTransferStateMachineType* Object NodeId is returned, the *Read* Method of the file fails until the *TransferState* changed to *ReadTransfer*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.14 specifies the *AddressSpace* representation for the *GenerateFileForRead Method*.

Table C.14 – GenerateFileForRead Method AddressSpace Definition

Attribute	Value				
BrowseName	StartReadTransfer				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.4.4 GenerateFileForWrite

GenerateFileForWrite is used to start the write file transaction. A successful call of this *Method* creates a temporary *FileType Object* and returns the *NodeId* of this *Object* and the file handle to access the *Object*.

Signature

```
GenerateFileForWrite(
    [in] BaseDataType     generateOptions
    [out] NodeId          fileNodeId
    [out] UInt32          fileHandle
```

```
    );
```

Argument	Description
generateOptions	The optional parameter can be used to specify server specific file generation options. To allow such options, the *Server* shall specify a concrete *DataType* in the *Argument Structure* for this argument in the instance of the *Method*. If the *DataType* is *BaseDataType*, the Client shall pass Null for this argument. Examples for concrete DataTypes are OptionsSet Used to provide a bit mask for file use selection Structure Can be used to provide a structure with create settings e.g. firmware update settings Enumeration Can be used to provide a list of options like file handling options
fileNodeId	NodeId of the temporary file.
fileHandle	The fileHandle of the opened *TransferFile*. The fileHandle can be used to access the *TransferFile Methods Write* and *Close*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.15 specifies the *AddressSpace* representation for the *GenerateFileForWrite Method*.

Table C.15 – GenerateFileForWrite Method AddressSpace Definition

Attribute	Value				
BrowseName	StartWriteTransfer				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.4.5 CloseAndCommit

CloseAndCommit is used to apply the content of the written file and to delete the temporary file after the completion of the transaction.

Signature

```
CloseAndCommit(
    [in]  UInt32          fileHandle
    [out] NodeId          completionStateMachine
  );
```

Argument	Description
fileHandle	The fileHandle used to write the file.
completionStateMachine	If the processing of the file is completed asynchronous, the parameter returns the NodeId of the corresponding *FileTransferStateMachineType Object*. If the processing of the file is already completed, the parameter is null. If a *FileTransferStateMachineType* Object NodeId is returned, the processing is in progress until the *TransferState* changed to *Idle*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See Part 4 for a general description.

Table C.16 specifies the *AddressSpace* representation for the *CloseAndCommit Method*.

Table C.16 – CloseAndCommit Method AddressSpace Definition

Attribute	Value				
BrowseName	CloseAndCommit				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

C.4.6 FileTransferStateMachineType

The states of the file transfer state machine are shown in Figure C.4.

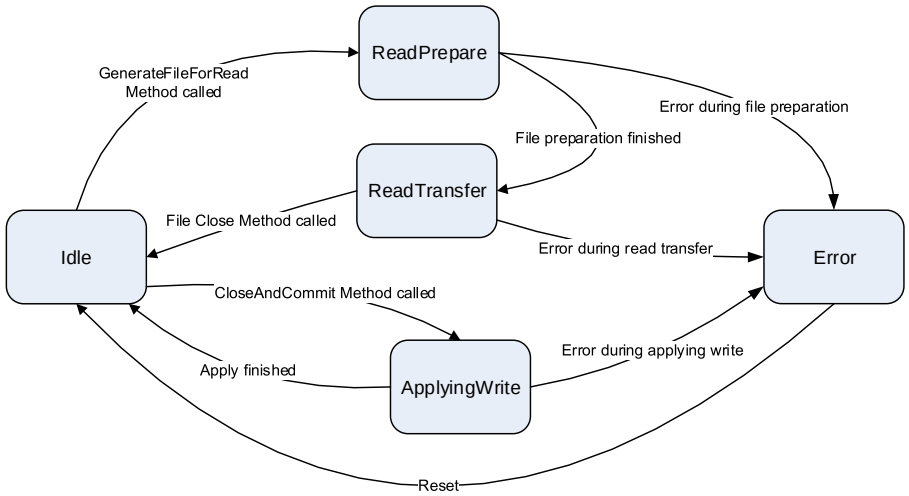

Figure C.4 – File Transfer States

The *FileTransferStateMachineType* and the related type are illustrated in Figure C.5.

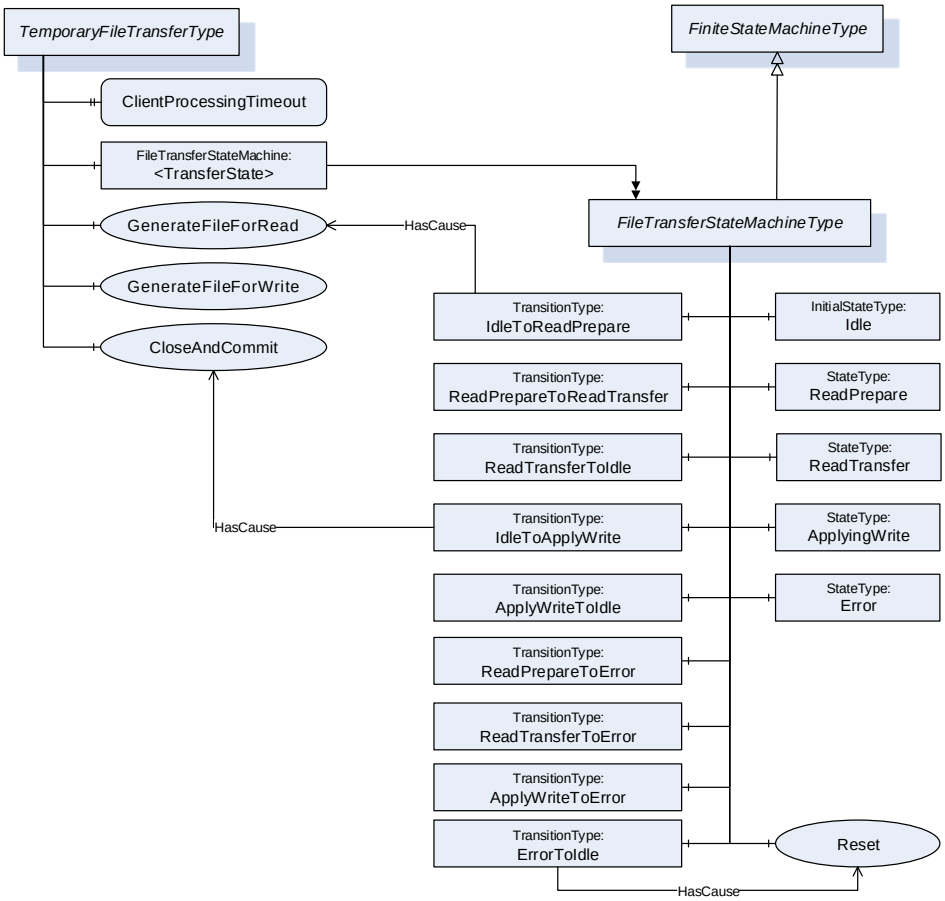

Figure C.5 – FileTransferStateMachineType

This *ObjectType* defines the StateMachine for asynchronous processing of temporary file transfers. It is formally defined in Table C.17.

Table C.17 – FileTransferStateMachineType

Attribute	**Value**				
BrowseName	FileTransferStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the FiniteStateMachineType defined in B.4.5.					
HasComponent	Object	Idle		InitialStateType	
HasComponent	Object	ReadPrepare		StateType	
HasComponent	Object	ReadTransfer		StateType	
HasComponent	Object	ApplyWrite		StateType	
HasComponent	Object	Error		StateType	
HasComponent	Object	IdleToReadPrepare		TransitionType	
HasComponent	Object	ReadPrepareToReadTransfer		TransitionType	
HasComponent	Object	ReadTransferToIdle		TransitionType	
HasComponent	Object	IdleToApplyWrite		TransitionType	
HasComponent	Object	ApplyWriteToIdle		TransitionType	
HasComponent	Object	ReadPrepareToError		TransitionType	
HasComponent	Object	ReadTransferToError		TransitionType	
HasComponent	Object	ApplyWriteToError		TransitionType	
HasComponent	Object	ErrorToIdle		TransitionType	
HasComponent	Method	Reset	Defined in C 4.7		

Table C.18 – FileTransferStateMachineType transitions

BrowseName	References	BrowseName	TypeDefinition
Transitions			
IdleToReadPrepare	FromState	Idle	StateType
	ToState	ReadPrepare	StateType
	HasEffect	TransitionEventType	
ReadPrepareToReadTransfer	FromState	ReadPrepare	StateType
	ToState	ReadTransfer	StateType
	HasEffect	TransitionEventType	
ReadTransferToIdle	FromState	ReadTransfer	StateType
	ToState	Idle	StateType
	HasEffect	TransitionEventType	
IdleToApplyWrite	FromState	Idle	StateType
	ToState	ApplyWrite	StateType
	HasEffect	TransitionEventType	
ApplyWriteToIdle	FromState	ApplyWrite	StateType
	ToState	Idle	StateType
	HasEffect	TransitionEventType	
ReadPrepareToError	FromState	ReadPrepare	StateType
	ToState	Error	StateType
	HasEffect	TransitionEventType	
ReadTransferToError	FromState	ReadTransfer	StateType
	ToState	Error	StateType
	HasEffect	TransitionEventType	
ApplyWriteToError	FromState	ApplyWrite	StateType
	ToState	Error	StateType
	HasEffect	TransitionEventType	
ErrorToIdle	FromState	Error	StateType
	ToState	Idle	StateType
	HasEffect	TransitionEventType	

C.4.7 Reset

Reset is used to reset the Error state of a *FileTransferStateMachineType Object*.

Signature

```
Reset();
```

Annex D
(normative)

DataTypeDictionary

D.1 Overview

This annex defines a way to provide encoding information for custom *DataTypes*. In previous releases of the specification this approach was defined in Part 3. In Part 3 a simplified approach is now defined having a *DataTypeDefinition Attribute* on the *DataType Node*. The approach using *DataTypeDictionaries* is provided for backwards compatibility and in case some specific requirements cannot be fulfilled with the simplified approach. It is recommended to only use the approach using the *DataTypeDefinition Attribute*.

D.2 Data Type Model

Part 3 defines the data type model. A *DataType* points to one or several *DataTypeEncoding Objects*. The approach of *DataTypeDictionaries* extends this model (see Figure D.1). The *DataTypeEncoding Object* points to exactly one *Variable* of type *DataTypeDescriptionType*. The *DataTypeDescription Variable* belongs to a *DataTypeDictionary Variable*.

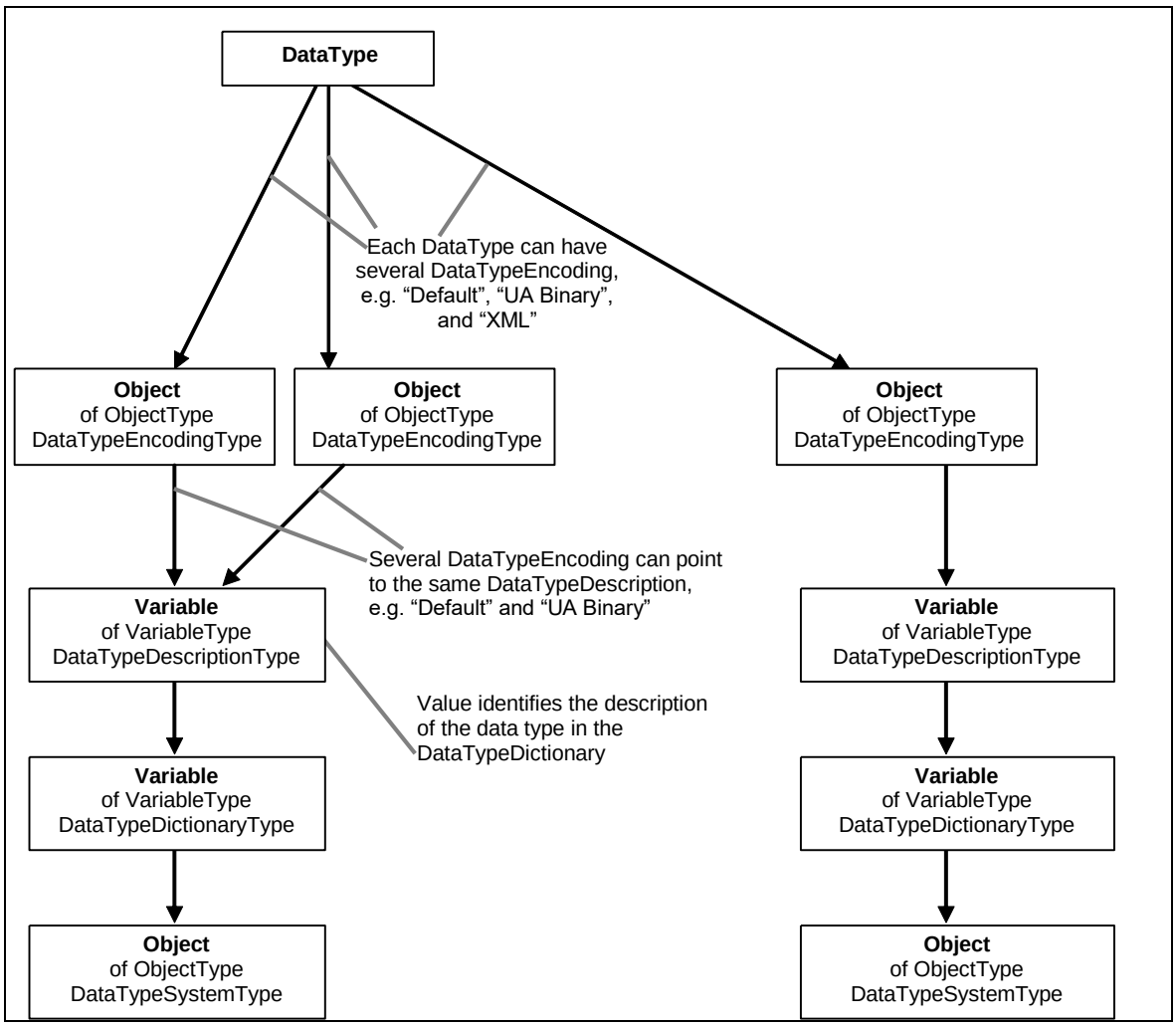

Figure D.1 – DataType Model

The *DataTypeDictionary* describes a set of *DataTypes* in sufficient detail to allow *Clients* to parse/interpret *Variable Values* that they receive and to construct *Values* that they send. The *DataTypeDictionary* is represented as a *Variable* of type *DataTypeDictionaryType* in the

AddressSpace, the description about the *DataTypes* is contained in its *Value Attribute*. All containing *DataTypes* exposed in the *AddressSpace* are represented as *Variables* of type *DataTypeDescriptionType*. The *Value* of one of these Variables identifies the description of a *DataType* in the *Value Attribute* of the *DataTypeDictionary*.

The *DataType* of a *DataTypeDictionary Variable* is always a ByteString. The format and conventions for defining *DataTypes* in this ByteString are defined by *DataTypeSystem*s. *DataTypeSystems* are identified by *NodeIds*. They are represented in the *AddressSpace* as *Objects* of the *ObjectType DataTypeSystemType*. Each *Variable* representing a *DataTypeDictionary* references a *DataTypeSystem Object* to identify their *DataTypeSystem*.

A client shall recognise the *DataTypeSystem* to parse any of the type description information. OPC UA *Clients* that do not recognise a *DataTypeSystem* will not be able to interpret its type descriptions, and consequently, the values described by them. In these cases, *Clients* interpret these values as opaque ByteStrings.

OPC Binary and W3C XML Schema are examples of *DataTypeSystems*. The OPC Binary *DataTypeSystem* is defined in Annex E. OPC Binary uses XML to describe binary data values. W3C XML Schema is specified in XML Schema Part 1 and XML Schema Part 2.

D.3 DataTypeDictionary, DataTypeDescription, DataTypeEncoding and DataTypeSystem

A *DataTypeDictionary* is an entity that contains a set of type descriptions, such as an XML schema. *DataTypeDictionaries* are defined as *Variables* of the *VariableType DataTypeDictionaryType*.

A *DataTypeSystem* specifies the format and conventions for defining *DataTypes* in *DataTypeDictionaries. DataTypeSystems* are defined as *Objects* of the *ObjectType DataTypeSystemType*.

The *ReferenceType* used to relate *Objects* of the *ObjectType DataTypeSystemType* to *Variables* of the *VariableType DataTypeDictionaryType* is the *HasComponent ReferenceType*. Thus, the *Variable* is always the *TargetNode* of a *HasComponent Reference*; this is a requirement for *Variables*. However, for *DataTypeDictionaries* the *Server* shall always provide the inverse *Reference*, since it is necessary to know the *DataTypeSystem* when processing the *DataTypeDictionary*.

Changes may be a result of a change to a type description, but it is more likely that dictionary changes are a result of the addition or deletion of type descriptions. This includes changes made while the *Server* is offline so that the new version is available when the *Server* restarts. *Clients* may subscribe to the *DataTypeVersion Property* to determine if the *DataTypeDictionary* has changed since it was last read.

The *Server* may, but is not required to, make the *DataTypeDictionary* contents available to *Clients* through the *Value Attribute*. *Clients* should assume that *DataTypeDictionary* contents are relatively large and that they will encounter performance problems if they automatically read the *DataTypeDictionary* contents each time they encounter an instance of a specific *DataType*. The client should use the *DataTypeVersion Property* to determine whether the locally cached copy is still valid. If the client detects a change to the *DataTypeVersion*, then it shall re-read the *DataTypeDictionary*. This implies that the *DataTypeVersion* shall be updated by a *Server* even after restart since *Clients* may persistently store the locally cached copy.

The *Value Attribute* of the *DataTypeDictionary* containing the type descriptions is a ByteString whose formatting is defined by the *DataTypeSystem*. For the “XML Schema” *DataTypeSystem*, the ByteString contains a valid XML Schema document. For the “OPC Binary” *DataTypeSystem*, the ByteString contains a string that is a valid XML document. The *Server* shall ensure that any change to the contents of the ByteString is matched with a corresponding change to the *DataTypeVersion Property*. In other words, the client may safely use a cached copy of the *DataTypeDictionary*, as long as the *DataTypeVersion* remains the same.

DataTypeDictionaries are complex *Variables* which expose their *DataTypeDescriptions* as *Variables* using *HasComponent References.* A *DataTypeDescription* provides the information

necessary to find the formal description of a *DataType* within the *DataTypeDictionary*. The *Value* of a *DataTypeDescription* depends on the *DataTypeSystem* of the *DataTypeDictionary*. When using "OPC Binary" dictionaries the *Value* shall be the name of the *TypeDescription*. When using "XML Schema" dictionaries the Value shall be an Xpath expression (see XPATH) which points to an XML element in the schema document.

Like *DataTypeDictionaries* each *DataTypeDescription* provides the *Property DataTypeVersion* indicating whether the type description of the *DataType* has changed. Changes to the *DataTypeVersion* may impact the operation of *Subscriptions*. If the *DataTypeVersion* changes for a *Variable* that is being monitored for a *Subscription* and that uses this *DataTypeDescription*, then the next data change *Notification* sent for the *Variable* will contain a status that indicates the change in the *DataTypeDescription*.

DataTypeEncoding Objects of the *DataTypes* reference their *DataTypeDescriptions* of the *DataTypeDictionaries* using *HasDescription* References. *Servers* shall provide the inverse *References* that relate the *DataTypeDescriptions* back to the *DataTypeEncoding Objects*. If a *DataType Node* is exposed in the *AddressSpace*, it shall provide its *DataTypeEncodings* and if a *DataTypeDictionary* is exposed then it should expose all of its *DataTypeDescriptions*. Both of these *References* shall be bi-directional.

Figure D.2 provides an example how *DataTypes* are modelled in the *AddressSpace*.

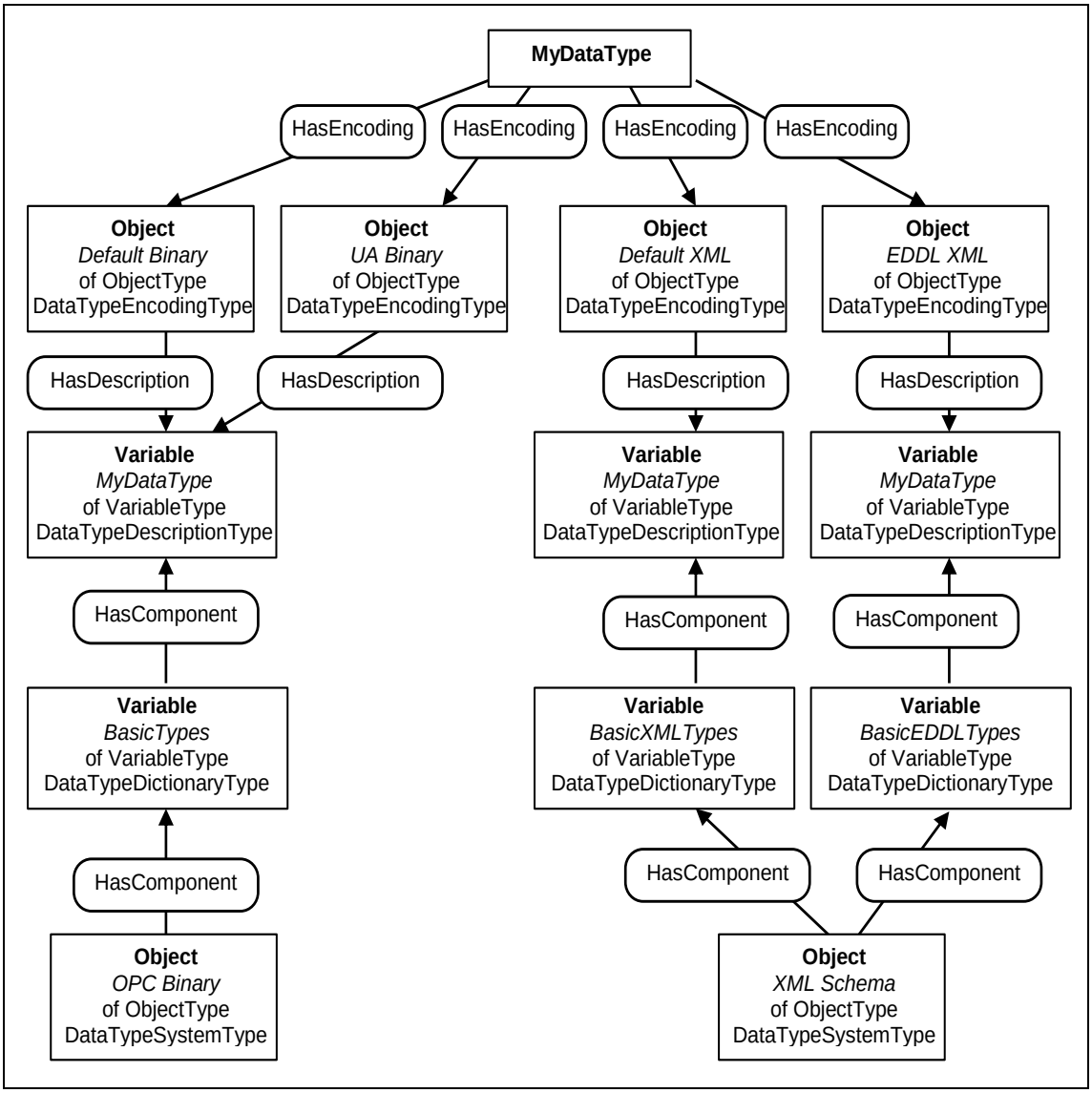

Figure D.2 – Example of DataType Modelling

In some scenarios an OPC UA *Server* may have resource limitations which make it impractical to expose large *DataTypeDictionaries*. In these scenarios the *Server* may be able to provide access to descriptions for individual DataTypes even if the entire dictionary cannot be read. For this reason, this standard defines a *Property* for the *DataTypeDescription* called *DictionaryFragment*. This *Property* is a ByteString that contains a subset of the *DataTypeDictionary* which describes the format of the *DataType* associated with the *DataTypeDescription*. Thus, the *Server* splits the large *DataTypeDictionary* into several small parts and *Clients* can access without affecting the overall system performance.

However, *Servers* should provide the whole *DataTypeDictionary* at once if this is possible. It is typically more efficient to read the whole *DataTypeDictionary* at once instead of reading individual parts.

D.4 AddressSpace Organization

In 8.2.9 the standard *Object* is introduced as entry point for *DataTypes* that the *Server* wishes to expose in the *AddressSpace*. When using *DataTypeSystems* and *DataTypeDictionaries* those Nodes can be referenced by this *Object* as well. The standard *Object* uses *Organizes References* to reference *Objects* of the *DataTypeSystemType* representing *DataTypeSystems*. Referenced by those *Objects* are *DataTypeDictionaries* that refer to their *DataTypeDescriptions*. However, it is not required to provide the *DataTypeSystem Objects*, and the *DataTypeDictionary* need not to be provided.

Because *DataTypes* are not related to *DataTypeDescriptions* using *hierarchical References*, *DataType Nodes* should be made available using *Organizes References* pointing either directly from the “DataTypes” *Object* to the *DataType Nodes* or using additional *Folder Objects* for grouping purposes. The intent is that all *DataTypes* of the *Server* exposed in the *AddressSpace* are accessible following *hierarchical References* starting from the “DataTypes” *Object*. However, this is not required.

Figure D.3 illustrates this hierarchy using the “OPC Binary” and “XML Schema” standard *DataTypeSystems* as examples. Other *DataTypeSystems* may be defined under this *Object*.

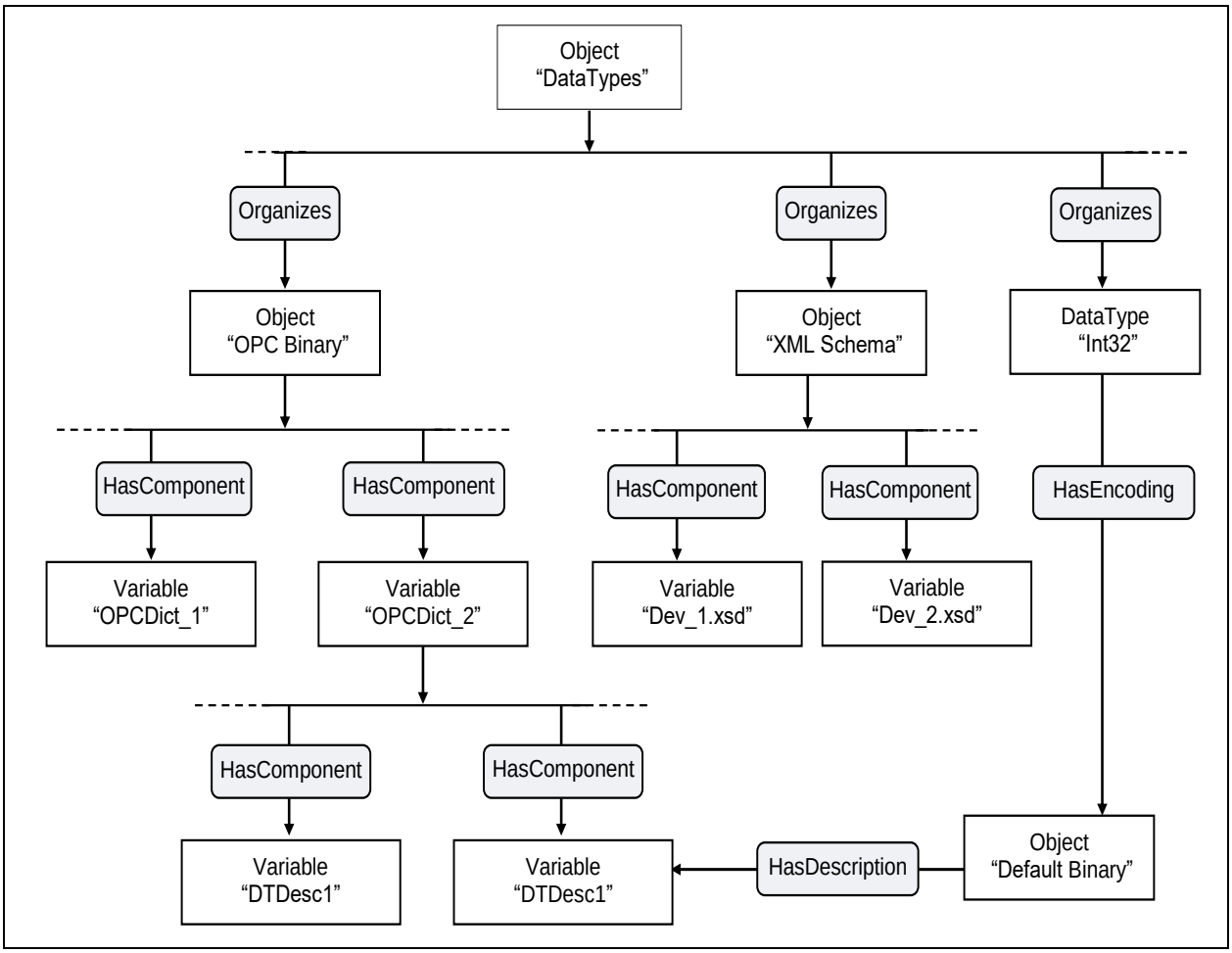

Figure D.3 – DataTypes Organization

Each *DataTypeSystem Object* is related to its *DataTypeDictionary Nodes* using *HasComponent References*. Each *DataTypeDictionary Node* is related to its *DataTypeDescription Nodes* using *HasComponent References*. These *References* indicate that the *DataTypeDescriptions* are defined in the dictionary.

In the example, the "DataTypes" *Object* references the *DataType* "Int32" using an *Organizes Reference*. The *DataType* uses the non-hierarchical *HasEncoding Reference* to point to its default encoding, which references a *DataTypeDescription* using the non-hierarchical *HasDescription Reference*.

In case *DataTypeSystems* are used, the standard *Objects* "OPC Binary" and "XML Schema" defined in D.5.5 and D.5.6 are connected via a *Organizes Reference* from the "DataTypes" *Object*.

D.5 Node Definitions

D.5.1 HasDescription

The *HasDescription ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to reference the *DataTypeDescription* of a *DataTypeEncoding*.

The *SourceNode* of *References* of this type shall be an *Object* of the *ObjectType DataTypeEncodingType* or one of its subtypes.

The *TargetNode* of this *ReferenceType* shall be a *Variable* of the *VariableType DataTypeDescriptionType* or one of its subtypes.

Its representation in the *AddressSpace* is specified in Table D.1.

Table D.1 – HasDescription ReferenceType

Attributes	**Value**		
BrowseName	HasDescription		
InverseName	DescriptionOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

D.5.2 DataTypeDictionaryType

The *DataTypeDictionaryType VariableType* is used as the type for the *DataTypeDictionaries*. It is formally defined in Table D.2.

Table D.2 – DataTypeDictionaryType Definition

Attribute	**Value**				
BrowseName	DataTypeDictionaryType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ByteString				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasProperty	Variable	DataTypeVersion	String	PropertyType	Optional
HasProperty	Variable	NamespaceUri	String	PropertyType	Optional
HasProperty	Variable	Deprecated	Boolean	Property Type	Optional

The *Property DataTypeVersion* is explained in D.3.

The *NamespaceUri* is the URI for the namespace described by the *Value Attribute* of the *DataTypeDictionary*. This is not always the same as the *NamespaceUri* of the *DataType NodeId*.

The *Deprecated Property* is used to indicate that all of the *DataType* definitions represented by the *DataTypeDictionaryType* are available through a *DataTypeDefinition Attribute*. Servers that provide *DataType* definitions as a *DataTypeDefinition Attribute* and through a *DataTypeDictionaryType* shall expose this *Property*.

D.5.3 DataTypeDescriptionType

The *DataTypeDescriptionType VariableType* is used as the type for the *DataTypeDescriptions*. It is formally defined in Table D.3.

Table D.3 – DataTypeDescriptionType Definition

Attribute	**Value**				
BrowseName	DataTypeDescriptionType				
IsAbstract	False				
ValueRank	-1 (-1 = Scalar)				
DataType	ByteString				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasProperty	Variable	DataTypeVersion	String	PropertyType	Optional
HasProperty	Variable	DictionaryFragment	ByteString	PropertyType	Optional

The *Properties* DataTypeVersion and DictionaryFragment are explained in D.3.

D.5.4 DataTypeSystemType

The *DataTypeSystems ObjectType* is used as type for the *DataTypeSystems*. There are no *References* specified for this *ObjectType*. It is formally defined in Table D.4.

Table D.4 – DataTypeSystemType Definition

Attribute		Value			
BrowseName		DataTypeSystemType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

D.5.5 OPC Binary

OPC Binary is a standard *DataTypeSystem* defined by OPC. It is represented in the *AddressSpace* by an *Object Node*. The OPC Binary *DataTypeSystem* is defined in Part 3. OPC Binary uses XML to describe complex binary data values. The "*OPC Binary*" *Object* is formally defined in Table D.5.

Table D.5 – OPC Binary Definition

Attribute	Value		
BrowseName	OPC Binary		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	DataTypeSystemType	Defined in D.5.4

D.5.6 XML Schema

XML Schema is a standard *DataTypeSystem* defined by the W3C. It is represented in the *AddressSpace* by an *Object Node*. XML Schema documents are XML documents whose xmlns attribute in the first line is:

schema xmlns =http://www.w3.org/1999/XMLSchema

The "*XML Schema*" *Object* is formally defined in Table D.6.

Table D.6 – XML Schema Definition

Attribute	Value		
BrowseName	XML Schema		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	DataTypeSystemType	Defined in D.5.4

Annex E
(normative)

OPC Binary Type Description System

E.1 Concepts

The OPC Binary XML Schema defines the format of OPC Binary *TypeDictionaries*. Each OPC Binary *TypeDictionary* is an XML document that contains one or more *TypeDescriptions* that describe the format of a binary-encoded value. Applications that have no advanced knowledge of a particular binary encoding can use the OPC Binary *TypeDescription* to interpret or construct a value.

The OPC Binary Type Description System does not define a standard mechanism to *encode* data in binary. It only provides a standard way to describe an existing binary encoding. Many binary encodings will have a mechanism to describe types that could be encoded; however, these descriptions are useful only to applications that have knowledge of the type description system used with each binary encoding. The OPC Binary Type Description System is a generic syntax that can be used by any application to interpret any binary encoding.

The OPC Binary Type Description System was originally defined in the OPC Complex Data Specification. The OPC Binary Type Description System described in Annex C is quite different and is correctly described as the OPC Binary Type Description System Version 2.0.

Each *TypeDescription* is identified by a *TypeName* which shall be unique within the *TypeDictionary* that defines it. Each *TypeDictionary* also has a *TargetNamespace* which should be unique among all OPC Binary *TypeDictionaries*. This means that the *TypeName* qualified with the *TargetNamespace* for the dictionary should be a globally-unique identifier for a *TypeDescription.*

Figure E.1 below illustrates the structure of an OPC Binary *TypeDictionary*.

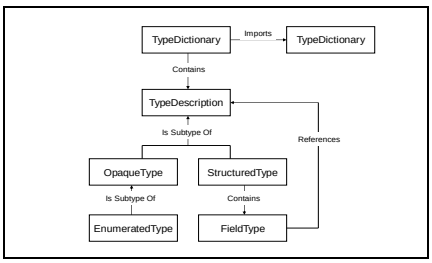

Figure E.1 – OPC Binary Dictionary Structure

Each binary encoding is built from a set of opaque building blocks that are either primitive types with a fixed length or variable-length types with a structure that is too complex to describe properly in an XML document. These building blocks are described with an *OpaqueType*. An instance of one of these building blocks is a binary-encoded value.

The OPC Binary Type Description System defines a set of standard *OpaqueTypes* that all OPC Binary *TypeDictionaries* should use to build their *TypeDescriptions*. These standard type descriptions are described in Clause E.3.

In some cases, the binary encoding described by an *OpaqueType* may have a fixed size which would allow an application to skip an encoded value that it does not understand. If that is the case, then the *LengthInBits* attribute should be specified for the *OpaqueType.* If authors of *TypeDictionaries* need to define new *OpaqueTypes* that do not have a fixed size then they should use the documentation elements to describe how to encode binary values for the type. This description should provide enough detail to allow a human to write a program that can interpret instances of the type.

A *StructuredType* breaks a complex value into a sequence of values that are described by a *FieldType*. Each *FieldType* has a name, type and a number of qualifiers that specify when the field is used and how many instances of the type exist. A *FieldType* is described completely in E.2.6.

An *EnumeratedType* describes a numeric value that has a limited set of possible values, each of which has a descriptive name. *EnumeratedTypes* provide a convenient way to capture semantic information associated with what would otherwise be an opaque numeric value.

E.2 Schema Description

E.2.1 TypeDictionary

The *TypeDictionary* element is the root element of an OPC Binary Dictionary. The components of this element are described in Table E.1.

Table E.1 – TypeDictionary Components

Name	Type	Description
Documentation	Documentation	An element that contains human-readable text and XML that provides an overview of what is contained in the dictionary.
Import	ImportDirective[]	Zero or more elements that specify other *TypeDictionaries* that are referenced by *StructuredTypes* defined in the dictionary. Each import element specifies the *NamespaceUri* of the *TypeDictionary* being imported. The *TypeDictionary* element shall declare an XML namespace prefix for each imported namespace.
TargetNamespace	xs:string	Specifies the URI that qualifies all *TypeDescriptions* defined in the dictionary.
DefaultByteOrder	ByteOrder	Specifies the default *ByteOrder* for all *TypeDescriptions* that have the *ByteOrderSignificant* attribute set to “true”. This value overrides the setting in any imported *TypeDictionary*. This value is overridden by the *DefaultByteOrder* specified on a *TypeDescription*.
TypeDescription	TypeDescription[]	One or more elements that describe the structure of a binary encoded value. A TypeDescription is an abstract type. A dictionary may only contain the *OpaqueType*, *EnumeratedType* and *StructuredType* elements.

E.2.2 TypeDescription

A *TypeDescription* describes the structure of a binary encoded value. A *TypeDescription* is an abstract base type and only instances of subtypes may appear in a *TypeDictionary*. The components of a *TypeDescription* are described in Table E.2.

Table E.2 – TypeDescription Components

Name	Type	Description
Documentation	Documentation	An element that contains human readable text and XML that describes the type. This element should capture any semantic information that would help a human to understand what is contained in the value.
Name	xs: NCName	An attribute that specifies a name for the *TypeDescription* that is unique within the dictionary. The fields of structured types reference *TypeDescriptions* by using this name qualified with the dictionary namespace URI.
DefaultByteOrder	ByteOrder	An attribute that specifies the default *ByteOrder* for the type description. This value overrides the setting in any *TypeDictionary* or in any *StructuredType* that references the type description.
anyAttribute	*	Authors of a *TypeDictionary* may add their own attributes to any *TypeDescription* that shall be qualified with a namespace defined by the author. Applications should not be required to understand these attributes in order to interpret a binary encoded instance of the type.

E.2.3 OpaqueType

An *OpaqueType* describes a binary encoded value that is either a primitive fixed length type or that has a structure too complex to capture in an OPC Binary type dictionary. Authors of type dictionaries should avoid defining *OpaqueTypes* that do not have a fixed length because it would prevent applications from interpreting values that use these types without having built-in knowledge of the *OpaqueType.* The OPC Binary Type Description System defines many

standard *OpaqueTypes* that should allow authors to describe most binary encoded values as *StructuredTypes*.

The components of an *OpaqueType* are described in Table E.3.

Table E.3 – OpaqueType Components

Name	Type	Description
TypeDescription	TypeDescription	An *OpaqueType* inherits all elements and attributes defined for a *TypeDescription* in Table E.2.
LengthInBits	xs:string	An attribute which specifies the length of the *OpaqueType* in bits. This value should always be specified. If this value is not specified the *Documentation* element should describe the encoding in a way that a human understands.
ByteOrderSignificant	xs:boolean	An attribute that indicates whether byte order is significant for the type. If byte order is significant then the application shall determine the byte order to use for the current context before interpreting the encoded value. The application determines the byte order by looking for the *DefaultByteOrder* attribute specified for containing *StructuredTypes* or the *TypeDictionary*. If *StructuredTypes* are nested the inner *StructuredTypes* override the byte order of the outer descriptions. If the *DefaultByteOrder* attribute is specified for the *OpaqueType*, then the *ByteOrder* is fixed and does not change according to context. If this attribute is “true”, then the *LengthInBits* attribute shall be specified and it shall be an integer multiple of 8 bits.

E.2.4 EnumeratedType

An *EnumeratedType* describes a binary-encoded numeric value that has a fixed set of valid values. The encoded binary value described by an *EnumeratedType* is always an unsigned integer with a length specified by the *LengthInBits* attribute.

The names for each of the enumerated values are not required to interpret the binary encoding, however, they form part of the documentation for the type.

The components of an *EnumeratedType* are described in Table E.4.

Table E.4 – EnumeratedType Components

Name	Type	Description
OpaqueType	OpaqueTypeDescription	An *EnumeratedType* inherits all elements and attributes defined for a *TypeDescription* in Table E.2 and for an *OpaqueType* defined in Table E.3. The *LengthInBits* attribute shall always be specified.
EnumeratedValue	EnumeratedValue	One or more elements that describe the possible values for the instances of the type.

E.2.5 StructuredType

A *StructuredType* describes a type as a sequence of binary-encoded values. Each value in the sequence is called a *Field*. Each *Field* references a *TypeDescription* that describes the binary-encoded value that appears in the field. A *Field* may specify that zero, one or multiple instances of the type appear within the sequence described by the *StructuredType*.

Authors of type dictionaries should use *StructuredTypes* to describe a variety of common data constructs including arrays, unions and structures.

Some fields have lengths that are not multiples of 8 bits. Several of these fields may appear in a sequence in a structure, however, the total number of bits used in the sequence shall be fixed and it shall be a multiple of 8 bits. Any field which does not have a fixed length shall be aligned on a byte boundary.

A sequence of fields which do not line up on byte boundaries are specified from the least significant bit to the most significant bit. Sequences which are longer than one byte overflow from the most significant bit of the first byte into the least significant bit of the next byte.

The components of a *StructuredType* are described in Table E.5.

Table E.5 – StructuredType Components

Name	Type	Description
TypeDescription	TypeDescription	A *StructuredType* inherits all elements and attributes defined for a *TypeDescription* in Table E.2.
Field	FieldType	One or more elements that describe the fields of the structure. Each field shall have a name that is unique within the *StructuredType*. Some fields may reference other fields in the *StructuredType* by using this name.

E.2.6 FieldType

A *FieldType* describes a binary encoded value that appears in sequence within a *StructuredType*. Every *FieldType* shall reference a *TypeDescription* that describes the encoded value for the field.

A *FieldType* may specify an array of encoded values.

Fields may be optional and they reference other *FieldTypes*, which indicate if they are present in any specific instance of the type.

The components of a *FieldType* are described in Table E.6.

Table E.6 – FieldType Components

Name	Type	Description
Documentation	Documentation	An element that contains human readable text and XML that describes the field. This element should capture any semantic information that would help a human to understand what is contained in the field.
Name	xs:string	An attribute that specifies a name for the *Field* that is unique within the *StructuredType*. Other fields in the structured type reference a *Field* by using this name.
TypeName	xs:QName	An attribute that specifies the *TypeDescription* that describes the contents of the field. A field may contain zero or more instances of this type depending on the settings for the other attributes and the values in other fields.
Length	xs:unsignedInt	An attribute that indicates the length of the field. This value may be the total number of encoded bytes or it may be the number of instances of the type referenced by the field. The *IsLengthInBytes* attributes specifies which of these definitions applies.
LengthField	xs:string	An attribute that indicates which other field in the *StructuredType* specifies the length of the field. The length of the field may be in bytes or it may be the number of instances of the type referenced by the field. The *IsLengthInBytes* attributes specify which of these definitions applies. If this attribute refers to a field that is not present in an encoded value, then the default value for the length is 1. This situation could occur if the field referenced is an optional field (see the *SwitchField* attribute). The length field shall be a fixed length Base-2 representation of an integer. If the length field is one of the standard signed integer types and the value is a negative integer, then the field is not present in the encoded stream. The *FieldType* referenced by this attribute shall precede the field with the *StructuredType*.
IsLengthInBytes	xs:boolean	An attribute that indicates whether the *Length* or *LengthField* attributes specify the length of the field in bytes or in the number of instances of the type referenced by the field.
SwitchField	xs:string	If this attribute is specified, then the field is optional and may not appear in every instance of the encoded value. This attribute specifies the name of another *Field* that controls whether this field is present in the encoded value. The field referenced by this attribute shall be an integer value (see the *LengthField* attribute). The current value of the switch field is compared to the *SwitchValue* attribute using the *SwitchOperand.* If the condition evaluates to true then the field appears in the stream. If the *SwitchValue* attribute is not specified, then this field is present if the value of the switch field is non-zero. The *SwitchOperand* field is ignored if it is present. If the *SwitchOperand* attribute is missing, then the field is present if the value of the switch field is equal to the value of the *SwitchValue* attribute. The *Field* referenced by this attribute shall precede the field with the *StructuredType*.

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>SwitchValue</td><td>xs:unsignedInt</td><td>This attribute specifies when the field appears in the encoded value. The value of the field referenced by the SwitchField attribute is compared using the SwitchOperand attribute to this value. The field is present if the expression evaluates to true. The field is not present otherwise.</td></tr>
<tr><td>SwitchOperand</td><td>xs:string</td><td>This attribute specifies how the value of the switch field should be compared to the switch value attribute. This field is an enumeration with the following values:

Equal — SwitchField is equal to the SwitchValue.
GreaterThan — SwitchField is greater than the SwitchValue.
LessThan — SwitchField is less than the SwitchValue.
GreaterThanOrEqual — SwitchField is greater than or equal to the SwitchValue.
LessThanOrEqual — SwitchField is less than or equal to the SwitchValue.
NotEqual — SwitchField is not equal to the SwitchValue.
In each case the field is present if the expression is true.</td></tr>
<tr><td>Terminator</td><td>xs:hexBinary</td><td>This attribute indicates that the field contains one or more instances of TypeDescription referenced by this field and that the last value has the binary encoding specified by the value of this attribute.
If this attribute is specified then the TypeDescription referenced by this field shall either have a fixed byte order (i.e. byte order is not significant or explicitly specified) or the containing StructuredType shall explicitly specify the byte order.
Examples:

<table>
<tr><th>Field Data Type</th><th>Terminator</th><th>Byte Order</th><th>Hexadecimal String</th></tr>
<tr><td>Char</td><td>tab character</td><td>not applicable</td><td>09</td></tr>
<tr><td>WideChar</td><td>tab character</td><td>BigEndian</td><td>0009</td></tr>
<tr><td>WideChar</td><td>tab character</td><td>LittleEndian</td><td>0900</td></tr>
<tr><td>Int16</td><td>1</td><td>BigEndian</td><td>0001</td></tr>
<tr><td>Int16</td><td>1</td><td>LittleEndian</td><td>0100</td></tr>
</table>
</td></tr>
<tr><td>anyAttribute</td><td>*</td><td>Authors of a TypeDictionary may add their own attributes to any FieldType which shall be qualified with a namespace defined by the authors. Applications should not be required to understand these attributes in order to interpret a binary encoded field value.</td></tr>
</table>

E.2.7 EnumeratedValue

An *EnumeratedValue* describes a possible value for an *EnumeratedType*.

The components of an *EnumeratedValue* are described in Table E.7.

Table E.7 – EnumeratedValue Components

Name	Type	Description
Name	xs:string	This attribute specifies a descriptive name for the enumerated value.
Value	xs:int	This attribute specifies the numeric value that could appear in the binary encoding.

E.2.8 ByteOrder

A *ByteOrder* is an enumeration that describes a possible value byte orders for *TypeDescriptions* that allow different byte orders to be used. There are two possible values: BigEndian and LittleEndian. BigEndian indicates the most significant byte appears first in the binary encoding. LittleEndian indicates that the least significant byte appears first.

E.2.9 ImportDirective

An *ImportDirective* specifies a *TypeDictionary* that is referenced by types defined in the current dictionary.

The components of an *ImportDirective* are described in Table E.8.

Table E.8 – ImportDirective Components

Name	Type	Description
Namespace	xs:string	This attribute specifies the *TargetNamespace* for the *TypeDictionary* being imported. This may be a well-known URI which means applications need not have access to the physical file to recognise types that are referenced.
Location	xs:string	This attribute specifies the physical location of the XML file containing the *TypeDictionary* to import. This value could be a URL for a network resource, a NodeId in an OPC UA *Server* address space or a local file path.

E.3 Standard Type Descriptions

The OPC Binary Type Description System defines a number of standard type descriptions that can be used to describe many common binary encodings using a *StructuredType*. The standard type descriptions are described in Table E.9.

Table E.9 – Standard Type Descriptions

Type name	Description
Bit	A single bit value.
Boolean	A two-state logical value represented as an 8-bit value.
SByte	An 8-bit signed integer.
Byte	An 8-bit unsigned integer.
Int16	A 16-bit signed integer.
UInt16	A 16-bit unsigned integer.
Int32	A 32-bit signed integer.
UInt32	A 32-bit unsigned integer.
Int64	A 64-bit signed integer.
UInt64	A 64-bit unsigned integer.
Float	An ISO/IEC/IEEE 60559:2011 single precision floating point value.
Double	AnISO/IEC/IEEE 60559:2011 : Information technology – Microprocessor Systems – Floating-Point arithmetic ISO/IEC/IEEE 60559:2011 double precision floating point value.
Char	An 8-bit UTF-8 character value.
String	A sequence of UTF-8 characters preceded by the number of UTF-8 Code Units (bytes).
WideString	A sequence of UTF-16 characters preceded by the number of UTF-16 Code Units.
DateTime	A 64-bit signed integer representing the number of 100 nanoseconds intervals since 1601-01-01 00:00:00. This is the same as the WIN32 FILETIME type.
ByteString	A sequence of bytes preceded by its length in bytes.
Guid	A 128-bit structured type that represents a WIN32 GUID value.

E.4 Type Description Examples

1. A 128-bit signed integer.

```
<opc:OpaqueType Name="Int128" LengthInBits="128" ByteOrderSignificant="true">
  <opc:Documentation>A 128-bit signed integer.</opc:Documentation>
</opc:OpaqueType>
```

2. A 16-bit value divided into several fields.

```
<opc:StructuredType Name="Quality">
  <opc:Documentation>An OPC COM-DA quality value.</opc:Documentation>
  <opc:Field Name="LimitBits" TypeName="opc:Bit" Length="2" />
  <opc:Field Name="QualityBits" TypeName="opc:Bit" Length="6"/>
  <opc:Field Name="VendorBits" TypeName="opc:Byte" />
</opc:StructuredType>
```

When using bit fields, the least significant bits within a byte shall appear first.

3. A structured type with optional fields.

```
<opc:StructuredType Name="DataValue">
  <opc:Documentation>A value with an associated timestamp, and
quality.</opc:Documentation>
```

```
  <opc:Field Name="ValueSpecified" TypeName="Bit" />
  <opc:Field Name="StatusCodeSpecified" TypeName="Bit" />
  <opc:Field Name="TimestampSpecified" TypeName="Bit" />
  <opc:Field Name="Reserved1" TypeName="Bit" Length="5" />
  <opc:Field Name="Value" TypeName="Variant" SwitchField="ValueSpecified" />
  <opc:Field Name="Quality" TypeName="Quality" SwitchField="StatusCodeSpecified" />
  <opc:Field Name="Timestamp"
             TypeName="opc:DateTime" SwitchField="SourceTimestampSpecified" />
</opc:StructuredType>
```

It is necessary to explicitly specify any padding bits required to ensure subsequent fields line up on byte boundaries.

4. An array of integers.

```
<opc:StructuredType Name="IntegerArray">
  <opc:Documentation>An array of integers prefixed by its length.</opc:Documentation>
  <opc:Field Name="Size" TypeName="opc:Int32" />
  <opc:Field Name="Array" TypeName="opc:Int32" LengthField="Size" />
</opc:StructuredType>
```

Nothing is encoded for the Array field if the Size field has a value ≤ 0.

5. An array of integers with a terminator instead of a length prefix.

```
<opc:StructuredType Name="IntegerArray" DefaultByteOrder="LittleEndian">
  <opc:Documentation>An array of integers terminated with a known
value.</opc:Documentation>
  <opc:Field Name="Value" TypeName="opc:Int16" Terminator="FF7F" />
</opc:StructuredType>
```

The terminator is 32,767 converted to hexadecimal with LittleEndian byte order.

6. A simple union.

```
<opc:StructuredType Name="Variant">
  <opc:Documentation>A union of several types.</opc:Documentation>
  <opc:Field Name="ArrayLengthSpecified" TypeName="opc:Bit" Length="1"/>
  <opc:Field Name="VariantType" TypeName="opc:Bit" Length="7" />
  <opc:Field Name="ArrayLength" TypeName="opc:Int32"
      SwitchField="ArrayLengthSpecified" />
  <opc:Field Name="Int32" TypeName="opc:Int32" LengthField="ArrayLength"
      SwitchField="VariantType" SwitchValue="1" />
  <opc:Field Name="String" TypeName="opc:String" LengthField="ArrayLength"
      SwitchField="VariantType" SwitchValue="2" />
  <opc:Field Name="DateTime" TypeName="opc:DateTime" LengthField="ArrayLength"
      SwitchField="VariantType" SwitchValue="3" />
</opc:StructuredType>
```

The *ArrayLength* field is optional. If it is not present in an encoded value, then the length of all fields with *LengthField* set to “ArrayLength” have a length of 1.

It is valid for the *VariantType* field to have a value that has no matching field defined. This simply means all optional fields are not present in the encoded value.

7. An enumerated type.

```
<opc:EnumeratedType Name="TrafficLight" LengthInBits="32">
  <opc:Documentation>The possible colours for a traffic signal.</opc:Documentation>
  <opc:EnumeratedValue Name="Red" Value="4">
    <opc:Documentation>Red says stop immediately.</opc:Documentation>
  </opc:EnumeratedValue>
  <opc:EnumeratedValue Name="Yellow" Value="3">
    <opc:Documentation>Yellow says prepare to stop.</opc:Documentation>
  </opc:EnumeratedValue>
  <opc:EnumeratedValue Name="Green" Value="2">
    <opc:Documentation>Green says you may proceed.</opc:Documentation>
  </opc:EnumeratedValue>
</opc:EnumeratedType>
```

The documentation element is used to provide human readable description of the type and values.

8. A nillable array.

```
<opc:StructuredTypen Name="NillableArray">
  <opc:Documentation>An array where a length of -1 means null.</opc:Documentation>
  <opc:Field Name="Length" TypeName="opc:Int32" />
  <opc:Field
      Name="Int32"
      TypeName="opc:Int32"
      LengthField="Length"
      SwitchField="Length"
      SwitchValue="0"
      SwitchOperand="GreaterThanOrEqual" />
</opc:StructuredType>
```

If the length of the array is −1 then the array does not appear in the stream.

E.5 OPC Binary XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  targetNamespace="http://opcfoundation.org/BinarySchema/"
  elementFormDefault="qualified"
  xmlns="http://opcfoundation.org/BinarySchema/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="Documentation">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:any minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
      <xs:anyAttribute/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="ImportDirective">
    <xs:attribute name="Namespace" type="xs:string" use="optional" />
    <xs:attribute name="Location" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:simpleType name="ByteOrder">
    <xs:restriction base="xs:string">
      <xs:enumeration value="BigEndian" />
      <xs:enumeration value="LittleEndian" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="TypeDescription">
    <xs:sequence>
      <xs:element ref="Documentation" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="Name" type="xs:NCName" use="required" />
    <xs:attribute name="DefaultByteOrder" type="ByteOrder" use="optional" />
    <xs:anyAttribute processContents="lax" />
  </xs:complexType>

  <xs:complexType name="OpaqueType">
    <xs:complexContent>
      <xs:extension base="TypeDescription">
        <xs:attribute name="LengthInBits" type="xs:int" use="optional" />
        <xs:attribute name="ByteOrderSignificant" type="xs:boolean" default="false" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="EnumeratedValue">
    <xs:sequence>
      <xs:element ref="Documentation"  minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="optional" />
    <xs:attribute name="Value" type="xs:unsignedInt" use="optional" />
  </xs:complexType>

  <xs:complexType name="EnumeratedType">
    <xs:complexContent>
      <xs:extension base="OpaqueTypeDescription">
        <xs:sequence>
       <xs:element name="EnumeratedValue"
                   type="EnumeratedValueDescription" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:extension>
```

```
    </xs:complexContent>
  </xs:complexType>

  <xs:simpleType name="SwitchOperand">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Equals" />
      <xs:enumeration value="GreaterThan" />
      <xs:enumeration value="LessThan" />
      <xs:enumeration value="GreaterThanOrEqual" />
      <xs:enumeration value="LessThanOrEqual" />
      <xs:enumeration value="NotEqual" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="FieldType">
    <xs:sequence>
      <xs:element ref="Documentation" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required" />
    <xs:attribute name="TypeName" type="xs:QName" use="optional" />
    <xs:attribute name="Length" type="xs:unsignedInt" use="optional" />
    <xs:attribute name="LengthField" type="xs:string" use="optional" />
    <xs:attribute name="IsLengthInBytes" type="xs:boolean" default="false" />
    <xs:attribute name="SwitchField" type="xs:string" use="optional" />
    <xs:attribute name="SwitchValue" type="xs:unsignedInt" use="optional" />
    <xs:attribute name="SwitchOperand" type="SwitchOperand" use="optional" />
    <xs:attribute name="Terminator" type="xs:hexBinary" use="optional" />
    <xs:anyAttribute processContents="lax" />
  </xs:complexType>

  <xs:complexType name="StructuredType">
    <xs:complexContent>
      <xs:extension base="TypeDescription">
        <xs:sequence>
          <xs:element name="Field" type="FieldType"
                      minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="TypeDictionary">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Documentation"  minOccurs="0" maxOccurs="1" />
        <xs:element name="Import" type="ImportDirective"
                    minOccurs="0" maxOccurs="unbounded" />
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="OpaqueType" type="OpaqueType" />
          <xs:element name="EnumeratedType" type="EnumeratedType" />
          <xs:element name="StructuredType" type="StructuredType" />
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="TargetNamespace" type="xs:string" use="required" />
      <xs:attribute name="DefaultByteOrder" type="ByteOrder" use="optional" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

E.6 OPC Binary Standard TypeDictionary

```
<?xml version="1.0" encoding="utf-8"?>
<opc:TypeDictionary
  xmlns="http://opcfoundation.org/BinarySchema/"
  xmlns:opc="http://opcfoundation.org/BinarySchema/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  TargetNamespace="http://opcfoundation.org/BinarySchema/"
>
  <opc:Documentation>This dictionary defines the standard types used by the OPC Binary
type description system.</opc:Documentation>

  <opc:OpaqueType Name="Bit" LengthInBits="1">
    <opc:Documentation>A single bit.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Boolean" LengthInBits="8">
```

```
    <opc:Documentation>A two state logical value represented as a 8-bit
value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="SByte" LengthInBits="8">
    <opc:Documentation>An 8-bit signed integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Byte" LengthInBits="8">
    <opc:Documentation>A 8-bit unsigned integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Int16" LengthInBits="16" ByteOrderSignificant="true">
    <opc:Documentation>A 16-bit signed integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="UInt16" LengthInBits="16" ByteOrderSignificant="true">
    <opc:Documentation>A 16-bit unsigned integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Int32" LengthInBits="32" ByteOrderSignificant="true">
    <opc:Documentation>A 32-bit signed integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="UInt32" LengthInBits="32" ByteOrderSignificant="true">
    <opc:Documentation>A 32-bit unsigned integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Int64" LengthInBits="32" ByteOrderSignificant="true">
    <opc:Documentation>A 64-bit signed integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="UInt64" LengthInBits="64" ByteOrderSignificant="true">
    <opc:Documentation>A 64-bit unsigned integer.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Float" LengthInBits="32" ByteOrderSignificant="true">
    <opc:Documentation>An IEEE-754 single precision floating point
value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Double" LengthInBits="64" ByteOrderSignificant="true">
    <opc:Documentation>An IEEE-754 double precision floating point
value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Char" LengthInBits="8">
    <opc:Documentation>A 8-bit character value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:StructuredType Name="String">
    <opc:Documentation>A UTF-8 null terminated string value.</opc:Documentation>
    <opc:Field Name="Value" TypeName="Char" Terminator="00" />
  </opc:StructuredType>

  <opc:StructuredType Name="CharArray">
    <opc:Documentation>A UTF-8 string prefixed by its length in
characters.</opc:Documentation>
    <opc:Field Name="Length" TypeName="Int32" />
    <opc:Field Name="Value" TypeName="Char" LengthField="Length" />
  </opc:StructuredType>

  <opc:OpaqueType Name="WideChar" LengthInBits="16" ByteOrderSignificant="true">
    <opc:Documentation>A 16-bit character value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:StructuredType Name="WideString">
    <opc:Documentation>A UTF-16 null terminated string value.</opc:Documentation>
    <opc:Field Name="Value" TypeName="WideChar" Terminator="0000" />
  </opc:StructuredType>

  <opc:StructuredType Name="WideCharArray">
    <opc:Documentation>A UTF-16 string prefixed by its length in
characters.</opc:Documentation>
    <opc:Field Name="Length" TypeName="Int32" />
    <opc:Field Name="Value" TypeName="WideChar" LengthField="Length" />
  </opc:StructuredType>
```

```
  <opc:StructuredType Name="ByteString">
    <opc:Documentation>An array of bytes prefixed by its length.</opc:Documentation>
    <opc:Field Name="Length" TypeName="Int32" />
    <opc:Field Name="Value" TypeName="Byte" LengthField="Length" />
  </opc:StructuredType>

  <opc:OpaqueType Name="DateTime" LengthInBits="64" ByteOrderSignificant="true">
    <opc:Documentation>The number of 100 nanosecond intervals since January 01,
1601.</opc:Documentation>
  </opc:OpaqueType>

  <opc:StructuredType Name="Guid">
    <opc:Documentation>A 128-bit globally unique identifier.</opc:Documentation>
    <opc:Field Name="Data1" TypeName="UInt32" />
    <opc:Field Name="Data2" TypeName="UInt16" />
    <opc:Field Name="Data3" TypeName="UInt16" />
    <opc:Field Name="Data4" TypeName="Byte" Length="8" />
  </opc:StructuredType>

</opc:TypeDictionary>
```

Annex F
(normative)

User Authorization

F.1 Overview

OPC UA defines a standard approach for implementing role based security. *Servers* may choose to implement part or all of the mechanisms defined here. The OPC UA approach assigns *Permissions* to *Roles* for each *Node* in the *AddressSpace*. *Clients* are then granted *Roles* when they create a *Session* based on the information provided by the *Client*.

F.2 RoleSetType

F.2.1 RoleSetType Definition

The RoleSet *Object defined in* Table 10 is a *RoleSetType* which is formally defined in Table F.1.

Table F.1 – RoleSetType Definition

Attribute	**Value**				
BrowseName	RoleSetType				
IsAbstract	False				
References	***Node* Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of *BaseObjectType* defined in 6.2.					
HasComponent	Object	<RoleName>		RoleType	OptionalPlaceholder
HasComponent	Method	AddRole	Defined in F.2.2		Mandatory
HasComponent	Method	RemoveRole	Defined in F.2.3.		Mandatory

The *AddRole Method* allows configuration *Clients* to add a new *Role* to the *Server*.

The *RemoveRole Method* allows configuration *Clients* to remove a *Role* from the *Server*.

F.2.2 AddRole Method

This *Method* is used to add a *Role* to the *RoleSet Object.*

The combination of the NamespaceUri and *RoleName* parameters are used to construct the *BrowseName* for the new *Node*. The BrowseName shall be unique within the *RoleSet Object*.

This *Method* affects security and shall only be browseable and callable by authorized administrators.

Part 3 defines well-known *Roles*. If this *Method* is used to add a well-known *Role*, the name of the *Role* from Part 3 is used together with the OPC UA namespace URI. The *Server* shall use the *NodeIds* for the well-known *Roles* in this case. The *NodeIds* for the well-known *Roles* are defined in Part 6.

Signature

```
AddRole (
   [in]  String          RoleName
   [in]  String          NamespaceUri
   [out] NodeId          RoleNodeId
   );
```

Argument	Description
RoleName	The name of the *Role*.
NamespaceUri	The *NamespaceUri* qualifies the *RoleName*. If this value is null or empty then the resulting *BrowseName* will be qualified by the *Server's NamespaceUri*.
RoleNodeId	The *NodeId* assigned by the *Server* to the new *Node*.

Method Result Codes

ResultCode	Description
Bad_InvalidArgument	The *RoleName* or NamespaceUri is not valid. The text associated with the error shall indicate the exact problem.
Bad_NotSupported	The *Server* does not allow more *Roles* to be added.
Bad_UserAccessDenied	The caller does not have the necessary *Permissions*.

F.2.3 RemoveRole Method

This *Method* is used to remove a *Role* from the *RoleSet Object.*

The *RoleNodeId* is the *NodeId* of the *Role Object* to remove.

The *Server* may prohibit the removal of some *Roles* because they are necessary for the *Server* to function.

If a *Role* is removed all *Permissions* associated with the *Role* are deleted as well. Ideally these changes should take effect immediately, however, some lag may occur.

This Method affects security and shall only be browseable and callable by authorized administrators.

Signature

```
RemoveRole (
   [in]  NodeId RoleNodeId
   );
```

Argument	Description
RoleNodeId	The *NodeId* of the *Role Object*.

Method Result Codes

ResultCode	Description
Bad_NodeIdUnknown	The specified *Role Object* does not exist.
Bad_NotSupported	The *Server* does not allow the *Role Object* to be removed.
Bad_UserAccessDenied	The caller does not have the necessary *Permissions*.
Bad_RequestNotAllowed	The specified Role Object cannot be removed.

F.3 RoleType

F.3.1 RoleType Definition

Each *Role Object* has the *Properties* and *Methods* defined by the *RoleType* which is formally defined in Table F.2.

Table F.2 – RoleType Definition

Attribute	**Value**				
BrowseName	RoleType				
IsAbstract	False				
References	***Node* Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType					
HasProperty	*Variable*	Identities	IdentityMapping RuleType []	PropertyType	Mandatory
HasProperty	*Variable*	ApplicationsExclude	Boolean	PropertyType	Optional
HasProperty	*Variable*	Applications	String []	PropertyType	Optional
HasProperty	*Variable*	EndpointsExclude	Boolean	PropertyType	Optional
HasProperty	*Variable*	Endpoints	EndpointType []	PropertyType	Optional
HasComponent	Method	AddIdentity	Defined in F.3.3.		Optional
HasComponent	Method	RemoveIdentity	Defined in F.3.4.		Optional
HasComponent	Method	AddApplication	Defined in F.3.3.		Optional
HasComponent	Method	RemoveApplication	Defined in F.3.4.		Optional
HasComponent	Method	AddEndpoint	Defined in F.3.3.		Optional
HasComponent	Method	RemoveEndpoint	Defined in F.3.4.		Optional

The *Properties* and *Methods* of the *RoleType* contain sensitive security related information and shall only be browseable, writeable and callable by authorized administrators through an encrypted channel.

The *Identities Property* specifies the currently configured rules for mapping a *UserIdentityToken* to the *Role*. If this Property is an empty array, then the *Role* cannot be granted to any *Session*.

The *ApplicationsExclude Property* defines the *Applications Property* as an include list or exclude list. If this *Property* is not provided or has a value of *FALSE* then only *Application Instance Certificates* included in the *Applications Property* shall be included in this *Role*. All other *Application Instance Certificates* shall not be included in this *Role*. If this *Property* has a value of *TRUE* then all *Application Instance Certificates* included in the *Applications Property* shall be excluded from this *Role*. All other *Application Instance Certificates* shall be included in this *Role*.

The *Applications Property* specifies the *Application Instance Certificates* of *Clients* which shall be included or excluded from this *Role*. Each element in the array is an *ApplicationUri* from a *Client Certificate* which is trusted by the *Server*.

The *EndpontsExclude Property* defines the *Endpoints Property* as an include list or exclude list. If this *Property* is not provided or has a value of *FALSE* then only *Endpoints* included in the *Endpoints Property* shall be included in this *Role*. All other *Endpoints* shall not be include this *Role*. If this *Property* has a value of *TRUE* then all *Endpoints* included in the *Endpoints Property* shall be excluded from this *Role*. All other *Endpoints* shall be included in this *Role*.

The *Endpoints Property* specifies the *Endpoints which shall be included or excluded from this* Role. The value is an *EndpointType* array which contains one or more *Endpoint* descriptions. The *EndpointType DataType* is defined in 12.22.

The *AddIdentity Method* adds a rule used to map a *UserIdentityToken* to the *Role*. If the *Server* does not allow changes to the mapping rules, then the Method is not present. A *Server* should prevent certain rules from being added to particular *Roles*. For example, a *Server* should refuse to allow an ANONYMOUS_5 (see F.3.2) mapping rule to be added to *Roles* with administrator privileges.

The *RemoveIdentity Method* removes a mapping rule used to map a *UserIdentityToken* to the *Role*. If the *Server* does not allow changes to the mapping rules, then the *Method* is not present.

The *AddApplication Method* adds an *Application Instance Certificate* to the list of. If the *Server* does not enforce application restrictions or does not allow changes to the mapping rules for the *Role* the Method is not present.

The *RemoveApplication Method* removes an *Application Instance Certificate* from the list of applications. If the *Server* does not enforce application restrictions or does not allow changes to the mapping rules for the *Role* the Method is not present.

F.3.2 IdentityMappingRuleType

The *IdentityMappingRuleType* structure defines a single rule for selecting a *UserIdentityToken*. The structure is described in Table F.3.

Table F.3 – IdentityMappingRuleType

Name	Type	Description
IdentityMappingRuleType	Structure	Specifies a rule used to map a *UserIdentityToken* to a *Role*.
criteriaType	Enumeration IdentityMappingType	The type of criteria contained in the rule. USERNAME_1 The rule specifies a UserName from a *UserNameIdentityToken*; THUMBPRINT_2 The rule specifies the *Thumbprint* of a User or CA *Certificate*; ROLE_3 The rule is a *Role* specified in an *Access Token*; GROUPID_4 The rule is a user group specified in the *Access Token*; ANONYMOUS_5 The rule specifies *Anonymous UserIdentityToken*; AUTHENTICATED_USER_6 The rules specify any non-*Anonymous UserIdentityToken*;
criteria	String	The criteria which the *UserIdentityToken* must meet for a *Session* to be mapped to the *Role*. The meaning of the criteria depends on the *mappingType*. The criteria are a “” for ANONYMOUS_5 and AUTHENTICATED_USER_6

If the criteriaType is USERNAME_1, the criteria is a name of a user known to the *Server*, For example, the user could be the name of a local operating system account.

If the criteriaType is THUMBPRINT_2, the criteria is a thumbprint of a *Certificate* of a user or CA which is trusted by the *Server*.

If the criteriaType is ROLE_3, the criteria is a name of a restriction found in the *Access Token*. For example, the *Role* “subscriber” may only be allowed to access *PubSub* related *Nodes*.

If the criteriaType is GROUPID_4, the criteria is a generic text identifier for a user group specific to the *Authorization Service.* For example, an *Authorization Service* providing access to an Active Directory may add one or more Windows Security Groups to the *Access Token*. Part 6 provides details on how groups are added to *Access Tokens*.

If the criteriaType is ANONYMOUS_5, the criteria is a null string which indicates no user credentials have been provided.

If the criteriaType is AUTHENTICATED_USER_6, the criteria is a null string which indicates any valid user credentials have been provided.

F.3.3 AddIdentity Method

This *Method* is used to add an identity mapping rule to a *Role*.

The *Client* shall use an encrypted channel and shall provide user credentials with administrator rights when invoking this *Method* on the *Server*.

Signature

```
AddIdentity (
   [in]  IdentityMappingRuleType Rule
   );
```

Argument	Description
Rule	The rule to add.

Method Result Codes

ResultCode	Description
Bad_InvalidArgument	The rule is not valid.
Bad_RequestNotAllowed	The rule cannot be added to the *Role* because of *Server* imposed restrictions.
Bad_NotSupported	The rule is not supported by the *Server*.
Bad_AlreadyExists	An equivalent rule already exists.

F.3.4 RemoveIdentity Method

This *Method* is used to remove an identity mapping rule from a *Role*.

The *Client* shall provide user credentials with administrator rights when invoking this *Method* on the *Server*.

Signature

```
RemoveIdentity (
   [in]  IdentityMappingRuleType Rule
   );
```

Argument	Description
Rule	The Rule to remove.

Method Result Codes

ResultCode	Description
Bad_NotFound	The rule does not exist.
Bad_UserAccessDenied	The session user is not allowed to configure the object.

F.3.5 AddApplication Method

This *Method* is used to add an application mapping rule to a *Role*.

The *Client* shall provide user credentials with administrator rights when invoking this *Method* on the *Server*.

Signature

```
AddApplication (
   [in]  String ApplicationUri
   );
```

Argument	Description
ApplicationUri	The *ApplicationUri* for the application.

Method Result Codes

ResultCode	Description
Bad_InvalidArgument	The *ApplicationUri* is not valid.
Bad_RequestNotAllowed	The mapping cannot be added to the *Role* because of *Server* imposed restrictions.
Bad_AlreadyExists	The ApplicationUri is already assigned to the *Role*.
Bad_UserAccessDenied	The session user is not allowed to configure the object.

F.3.6 RemoveApplication Method

This *Method* is used to remove an application mapping rule from a *Role*.

The *Client* shall provide user credentials with administrator rights when invoking this *Method* on the *Server*.

Signature

```
RemoveApplication (
   [in]  String ApplicationUri
   );
```

Argument	Description
ApplicationUri	The *ApplicationUri* for the application.

Method Result Codes

ResultCode	Description
Bad_NotFound	The ApplicationUri is not assigned to the *Role*.
Bad_UserAccessDenied	The session user is not allowed to configure the object.

F.3.7 AddEndpoint Method

This *Method* is used to add an endpoint mapping rule to a *Role*.

The *Client* shall provide user credentials with administrator rights when invoking this *Method* on the *Server*.

Signature

```
AddEndpoint (
    [in]  EndpointType Endpoint
    );
```

Argument	Description
Endpoint	The *Endpoint to add*.

Method Result Codes

ResultCode	Description
Bad_InvalidArgument	The *EndpointUrl* is not valid.
Bad_RequestNotAllowed	The mapping cannot be added to the *Role* because of *Server* imposed restrictions.
Bad_AlreadyExists	The *EndpointUrl* is already assigned to the *Role*.
Bad_UserAccessDenied	The session user is not allowed to configure the object.

F.3.8 RemoveEndpoint Method

This *Method* is used to remove an endpoint mapping rule from a *Role*.

The *Client* shall provide user credentials with administrator rights when invoking this *Method* on the *Server*.

Signature

```
RemoveEndpoint (
    [in]  EndpointType Endpoint
    );
```

Argument	Description
Endpoint	The *Endpoint to remove*.

Method Result Codes

ResultCode	Description
Bad_NotFound	The *EndpointUrl* is not assigned to the *Role*.
Bad_UserAccessDenied	The session user is not allowed to configure the object.

F.4 RoleMappingRuleChangedAuditEventType

This *Event* is raised when a mapping rule for a *Role* is changed.

This is the result of calling any of the add or remove *Methods* defined on the *RoleType*.

It shall be raised when the *AddIdentity, RemoveIdentity, AddApplication, RemoveApplication, AddEndpoint* or *RemoveEndpoint* Method causes an update to a *Role*.

Its representation in the *AddressSpace* is formally defined in Table F.4.

Table F.4 – RoleMappingRuleChangedAuditEventType Definition

Attribute	**Value**				
BrowseName	RoleMappingRuleChangedAuditEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateMethodEventType* defined in 6.4.27					

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantics are defined in 6.4.27.

_____