IEC 62541-5

Edition 3.0 2020-07
REDLINE VERSION

INTERNATIONAL STANDARD

colour
inside

OPC unified architecture –
Part 5: Information Model

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: sales@iec.ch.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC 62541-5

Edition 3.0 2020-07
REDLINE VERSION

INTERNATIONAL STANDARD

colour
inside

OPC unified architecture –
Part 5: Information Model

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.05

ISBN 978-2-8322-8656-2

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

OPC UNIFIED ARCHITECTURE –

Part 5: Information Model

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.

3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.

8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

This redline version of the official IEC Standard allows the user to identify the changes made to the previous edition. A vertical bar appears in the margin wherever a change has been made. Additions are in green text, deletions are in strikethrough red text.

International Standard IEC 62541-5 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2015. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) Added Annex F on User Authentication. Describes the Role Information Model that also allows configuration of Roles.
b) Added new data types: "Union", "Decimal", "OptionSet", "DateString", "TimeString", "DurationString", NormalizedString", "DecimalString", and "AudioDataType".
c) Added Method to request a state change in a Server.
d) Added Method to set Subscription to persistent mode.
e) Added Method to request resending of data from a Subscription.
f) Added concept allowing to temporarily create a file to write to or read from a server in C.4.
g) Added new Variable type to support Selection Lists.
h) Added optional properties to FiniteStateMachineType to expose currently available states and transitions.
i) Added UrisVersion Property to ServerType. This version information can be used for session-less service invocation.

The text of this standard is based on the following documents:

FDIS	Report on voting
65E/717/FDIS	65E/733/RVD

Full information on the voting for the approval of this International Standard can be found in the report on voting indicated in the above table.

This document has been drafted in accordance with the ISO/IEC Directives, Part 2.

Throughout this document and the other parts of the IEC 62541 series, certain document conventions are used:

Italics are used to denote a defined term or definition that appears in Clause 3 in one of the parts of the series.

Italics are also used to denote the name of a service input or output parameter or the name of a structure or element of a structure that are usually defined in tables.

The *italicized terms and names* are also often written in camel-case (the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound). For example the defined term is *AddressSpace* instead of Address Space. This makes it easier to understand that there is a single definition for *AddressSpace*, not separate definitions for Address and Space.

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, can be found on the IEC website.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under "http://webstore.iec.ch" in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

OPC UNIFIED ARCHITECTURE –

Part 5: Information Model

1 Scope

This part of IEC 62541 defines the Information Model of the OPC Unified Architecture. The Information Model describes standardized *Nodes* of a *Server*’s *AddressSpace*. These *Nodes* are standardized types as well as standardized 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.

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

IEC 62541-3, *OPC Unified Architecture – Part 3: Address Space Model*

IEC 62541-4, *OPC Unified Architecture – Part 4: Services*

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

IEC 62541-9, *OPC Unified Architecture – Part 9: Alarms and Conditions*

IEC 62541-10, *OPC Unified Architecture – Part 10: Programs*

IEC 62541-11, *OPC Unified Architecture – Part 11: Historical Access*

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

IETF RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
http://www.ietf.org/rfc/rfc2045.txt

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

IETF RFC 2047, Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text
http://www.ietf.org/rfc/rfc2047.txt

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/

IETF RFC 3629: UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc3629.txt

3 Terms, definitions, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1 and IEC 62541-3 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

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 Abbreviated terms

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 IEC 62541-3). 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 IEC 62541-3) 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 –TypeDefinitionTable

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 *TargetNode*.	*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*.	~~*Attributes*~~ *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 document are only symbolic names. IEC 62541-6 defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this document 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 document, 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 document is 0. The namespace for this NamespaceIndex is specified in IEC 62541-3.

Note that this document 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 document is specified in the tables defining the *Nodes*. The NamespaceIndex for all *BrowseNames* defined in this document is 0.

5 Common Attributes

5.1 General

For all *Nodes* specified in this document, 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 ~~vendor~~ server-specific.
Description	Optionally a ~~vendor~~ 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 IEC 62541-6.
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 ~~that are not said to be vendor-specific~~. 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 document.
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 depends 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 document, 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 ~~vendor~~ server-specific.

5.3 Variables

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

Table 5 – Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a ~~vendor~~server-specific minimum sampling interval is provided.
AccessLevel	The access level for *Variables* used for type definitions is ~~vendor~~server-specific, for all other *Variables* defined in this document, the access level shall allow ~~a current read~~ reading; other settings are ~~vendor~~ server-specific.
UserAccessLevel	The value for the *UserAccessLevel Attribute* is ~~vendor~~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 ~~vendor~~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 ~~vendor~~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 document, the *Attributes* named in Table 6 shall be set as specified in Table 6.

Table 6 – Common VariableType Attributes

Attributes	Value
Value	Optionally a ~~vendor~~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 ~~vendor~~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 document, 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 document 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 document 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 document. An example of extending the *ObjectTypes* is putting the standard *Property NodeVersion* defined in IEC 62541-3 into the *BaseObjectType*, stating that each *Object* of the *Server* will provide a *NodeVersion*.

In addition to the *ObjectTypes* in Clause 6, Annex B provides *ObjectTypes* for StateMachines, Annex C provides *ObjectTypes* Model for File Transfer and Annex F defines *ObjectTypes* for User Authorization.

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**	**ModellingRule**
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		
~~HasSubtype~~	~~ObjectType~~	~~DataTypeSystemType~~	~~Defined in 6.8~~		

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[a]	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[a]	-- ServerCapabilitiesType	Mandatory
HasComponent	Object	ServerDiagnostics[a]	-- ServerDiagnosticsType	Mandatory
HasComponent	Object	VendorServerInfo	-- VendorServerInfoType	Mandatory
HasComponent	Object	ServerRedundancy[a]	-- 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~~ [a] 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*. IEC 62541-4 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 IEC 62541-4. 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 IEC 62541-4.

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. ~~The concrete values are vendor-specific.~~ IEC 62541-4 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 IEC 62541-7 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~~13 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 IEC 62541-4.

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 IEC 62541-4.

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 IEC 62541-4.

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 IEC 62541-4.

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	~~Variable~~ Object	~~Vendor specific *Variables* of a subtype of the ServerVendorCapabilityType defined in 7.5~~ RoleSet	~~--~~ RoleSetType	Optional

ServerProfileArray lists the *Profiles* that the *Server* supports. See IEC 62541-7 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 IEC 62541-7.

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 IEC 62541-3 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 length of Strings supported by *Variables* of the *Server*. The *Server* might further restrict the String length for individual *Variables* without notice to the client. *Servers* may use the *Property MaxStringLength* defined in IEC 62541-3 on individual *DataVariables* to specify the length on individual values. The individual *Property* may have larger or smaller values than *MaxStringLength*.~~

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 IEC 62541-3 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 IEC 62541-3 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 remaining components of the *ServerCapabilitiesType* define the server-specific capabilities of the *Server*. Each is defined using a *HasComponent Reference* whose target is an instance of a vendor-defined subtype of the abstract *ServerVendorCapabilityType* (see 7.5). Each subtype of this type defines a specific *Server* capability. The *NodeIds* for these *Variables* and their *VariableTypes* are server-defined.~~

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.~~11~~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 ~~statistic~~ diagnostic information is collected, and setting the value to FALSE disables the collection without resetting the ~~statistic~~ diagnostic values.

~~Static diagnostic *Nodes* that always appear in the *AddressSpace* will return Bad_NotReadable when the *Value Attribute* of such a *Node* is read or subscribed to and diagnostics are turned off. Dynamic diagnostic *Nodes* (such as the *Session Nodes*) will not appear in the *AddressSpace* when diagnostics are turned off.~~

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 the 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* (IEC 62541-4) 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 IEC 62541-4 for the definition of redundancy in this document. 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*. ~~The *Client* is allowed to access the redundant *Server* only as described there, however, "hot" switchover implies the support of "warm" switchover and "warm" switchover implies the support of "cold" switchover. Support for HotAndMirrored redundancy implies the support of "hot" switchover, however, for Servers supporting HotandMirrored redundancy it is strongly recommended that *Clients* use the HotAndMirrored mechanisms.~~ Its intended use is defined in IEC 62541-4.

~~If the *Server* supports only a "cold" switchover, the *ServiceLevel Variable* of the *Server Object* should be considered to identify the primary *Server*. In this scenario, only the primary *Server* may be able to access the underlying system, because the underlying system may support access only from a single *Server*. In this case, all other *Servers* will be identified with a *ServiceLevel* of zero.~~

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	MaxNodesPerTranslateBrowsePathsToNodeIds	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*~~ using historyReadDetails RAW, PROCESSED, MODIFIED or ATTIME~~.

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

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*~~, and~~ or the DeleteMonitoredItems *Service*,
- the maximum size of the sum of the linksToAdd and ~~the~~ 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 IEC 62541-6.

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	RolePermissionType[]	PropertyType	Optional
HasProperty	Variable	DefaultUserRolePermissions	RolePermissionType[]	PropertyType	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 IEC 62541-4 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 IEC 62541-6. 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 document defines standard *EventTypes*. They are represented in the *AddressSpace* as *ObjectTypes*. The *EventTypes* are already defined in IEC 62541-3. The following subclauses specify their representation in the *AddressSpace*.

6.4.2 BaseEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-4, 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 to 1 000
MEDIUM HIGH	601 to 800
MEDIUM	401 to 600
MEDIUM LOW	201 to 400
LOW	1 to 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 to 1 000)	15	1 000
	14	955
	13	910
	12	865
	11	820
	10	775
	9	730
	8	685
MEDIUM (334 to 666)	7	650
	6	575
	5	500
	4	425
	3	350
LOW (1 to 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 IEC 62541-9.

6.4.3 AuditEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-3.

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* ~~should~~ 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 IEC 62541-3. 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 IEC 62541-3. 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.~~ There are no additional *Properties* defined for this *EventType*.~~ The *SourceNode* *Property* for *Events* of this type~~ should~~ shall be assigned to the *Server Object*. The *SourceName* for *Events* of this type~~ should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-6 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 IEC 62541-3. 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 ~~*AuditEventType*~~ *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 ~~*AuditEventType*~~ *AuditSecurityEventType*. Their semantic is defined in 6.4.4.

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

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

The *SessionId* ~~should~~ 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* ~~is set to~~ shall be null.

6.4.8 AuditCreateSessionEventType

This *EventType* is defined in IEC 62541-3. 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 ~~should~~ 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 IEC 62541-6 for details on thumbprints.

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

6.4.9 AuditUrlMismatchEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-3. 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 ~~should~~ 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ shall be assigned to the *Server Object*. The *SourceName* for *Events* of this type ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ 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 IEC 62541-3. 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 ~~should~~ shall be assigned to the *NodeId* that was changed. The *SourceName* for *Events* of this type ~~should~~ 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 IEC 62541-3. 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 ~~should~~ shall be "Attribute/Write". Their semantic is defined in 6.4.24.

AttributeId identifies the *Attribute* that was written ~~on~~. 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 ~~to the *SourceNode*~~. If the *IndexRange* is provided, only the values in the provided range are shown.

OldValue identifies the value that the ~~*SourceNode*~~ *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 IEC 62541-3. 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 IEC 62541-11 representing the different possibilities to manipulate historical data.

6.4.27 AuditUpdateMethodEventType

This *EventType* is defined in IEC 62541-3. 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 ~~should~~ shall be assigned to the *NodeId* of the *Object* that the *Method* resides on. The *SourceName* for *Events* of this type ~~should~~ 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 ~~should~~ 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 ~~should~~ shall be the

String part of the *BrowseName* of the *View*; for the whole *AddressSpace* it ~~should~~ shall be "Server".

6.4.32 GeneralModelChangeEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-3. 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 ~~should~~ 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 ~~should~~ shall be the *String* part of the *BrowseName* of the *View*, for the whole *AddressSpace* it ~~should~~ 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. IEC 62541-4 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 IEC 62541-4>" 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 IEC 62541-3. 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 IEC 62541-3.

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 IEC 62541-3. This *ObjectType* is used as type for the *DataTypeEncodings*. The use of the *DataTypeEncodingType* with *DataTypeDictionaries* is defined in Annex D. 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 DataTypeSystemType~~

~~*DataTypeSystems* are defined in IEC 62541-3. This *ObjectType* is used as type for the *DataTypeSystems*. There are no *References* specified for this *ObjectType*. It is formally defined in Table 59.~~

~~Table 59 – DataTypeSystemType Definition~~

~~Attribute~~	**~~Value~~**				
~~BrowseName~~	~~DataTypeSystemType~~				
~~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 IEC 62541.

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 document 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 IEC 62541-3, 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~~	~~DataTypeDictionaryType~~	~~Defined in 7.6~~
~~HasSubtype~~	~~VariableType~~	~~DataTypeDescriptionType~~	~~Defined in 7.7~~
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 DataTypeDictionaryType~~

~~*DataTypeDictionaries* are defined in IEC 62541-3. This *VariableType* is used as the type for the *DataTypeDictionaries*. There are no *References* specified for this *VariableType*. It is formally defined in Table 65.~~

~~**Table 65 – 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~~

~~The *Property* DataTypeVersion is defined in IEC 62541-3. The NamespaceUri is the URI for the namespace described by the *Value Attribute* of the DataTypeDictionary.~~

~~7.7 DataTypeDescriptionType~~

~~*DataTypeDescriptions* are defined in IEC 62541-3. This *VariableType* is used as the type for the *DataTypeDescriptions*. There are no *References* specified for this *VariableType*. It is formally defined in Table 66.~~

~~**Table 66 – 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 defined in IEC 62541-3.~~

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[a]	BuildInfo	BuildInfoType	Mandatory
HasComponent	Variable	SecondsTillShutdown	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	ShutdownReason	LocalizedText	BaseDataVariableType	Mandatory
~~NOTE~~ [a] 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	~~VariableType~~ 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	~~VariableType~~ 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	~~TotalRequestsCount~~ TotalRequestCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	~~UnauthorizedRequestsCount~~ UnauthorizedRequestCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	ReadCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	HistoryReadCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	WriteCount	ServiceCounterDataType BaseDataVariableType	Mandatory

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	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 BaseDataVariableType	Mandatory
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

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	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**	**BrowseName**	**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 and the *OptionSetValues Property* contains the human-readable representation for each bit of the bit mask set to true. The order of the bits of the bit mask points to a position of the array, i.e. the first bit (least significant bit) points to the first entry in the array, etc.~~

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.

~~The OptionSetValues array contains an empty LocalizedText for each bit that has no specific meaning. The *VariableType* is formally defined in Table 76.~~

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**	**BrowseName**	**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*. Text code defined in IETF RFC 2045, IETF RFC 2046 and IETF RFC 2047 shall be used 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	AudiDataType				
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.

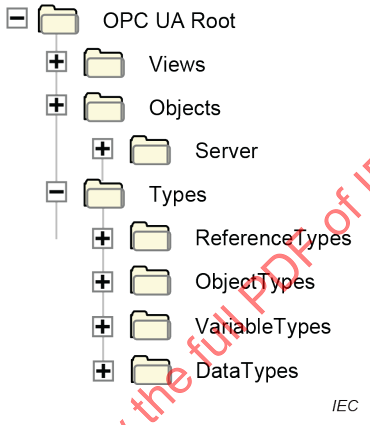


Figure 1 – Standard AddressSpace structure

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".

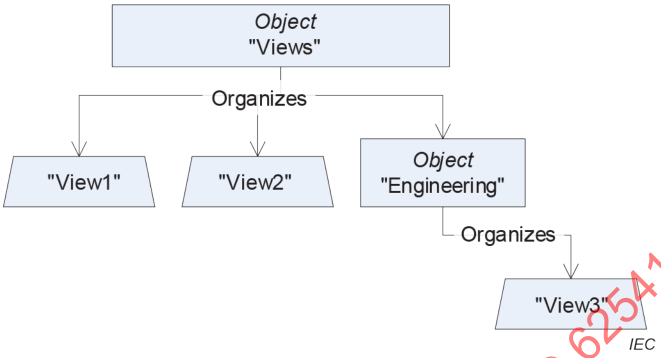


Figure 2 – Views organization

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.

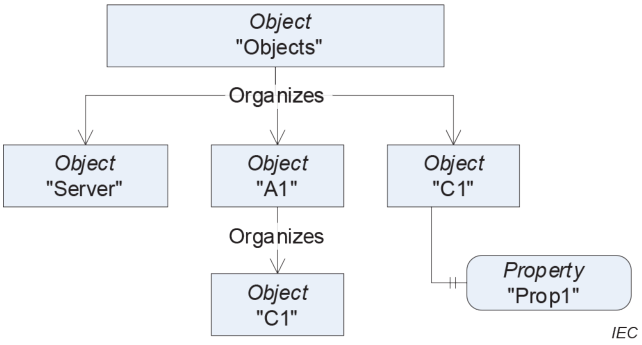


Figure 3 – Objects organization

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 Clause 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*.

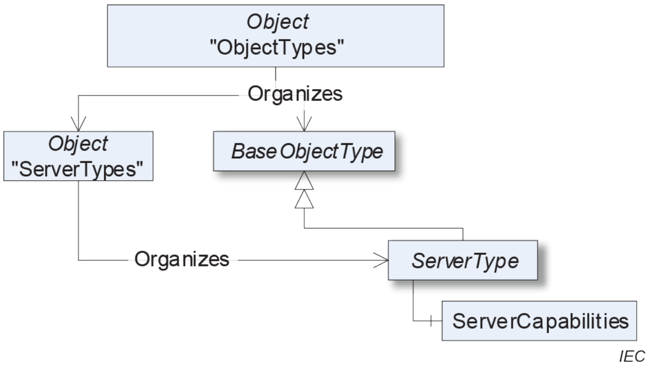


Figure 4 – ObjectTypes organization

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*.

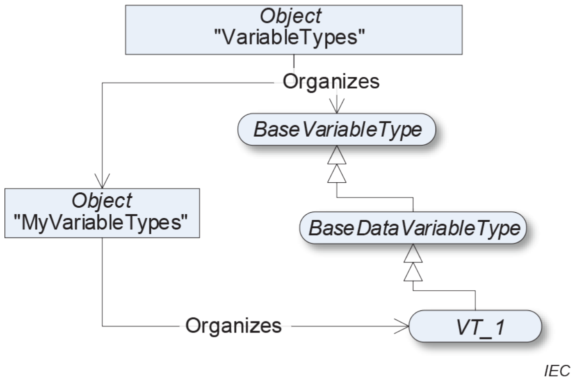


Figure 5 – VariableTypes organization

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*.

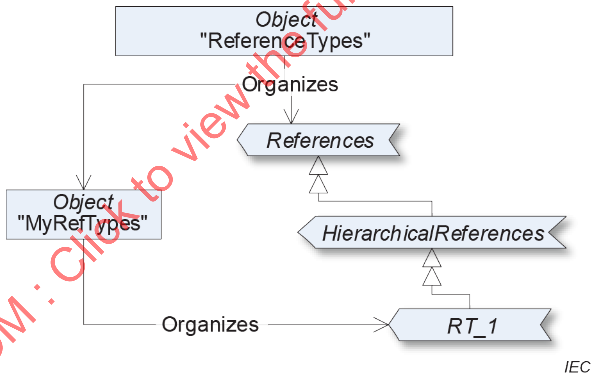


Figure 6 – ReferenceType definitions

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

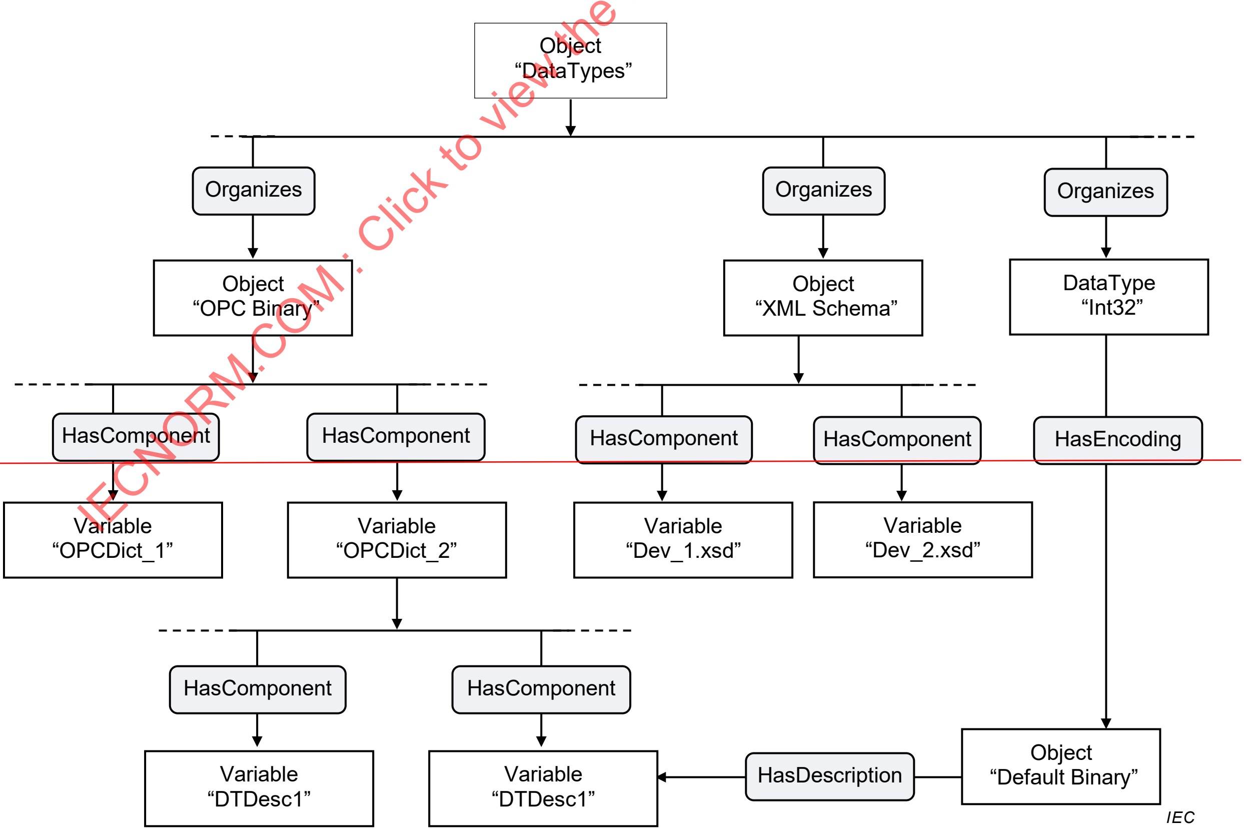
Attribute	**Value**		
BrowseName	ReferenceTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ReferenceType	References	Defined in 11.1

8.2.9 DataTypes

~~This standard *Object* is the browse entry point for *DataTypes* that the *Server* wishes to expose in the *AddressSpace*. 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 7 illustrates this hierarchy using the "OPC Binary" and "XML Schema" standard *DataTypeSystems* as examples. Other *DataTypeSystems* may be defined under this *Object*.~~



~~**Figure 7 – 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*.~~

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~~	~~Object~~	~~OPC Binary~~	~~Defined in 8.2.10~~
~~Organizes~~	~~Object~~	~~XML Schema~~	~~Defined in 8.2.11~~
Organizes	DataType	BaseDataType	Defined in 12.2

~~8.2.10 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 IEC 62541-3. OPC Binary uses XML to describe complex binary data values. The "*OPC Binary*" *Object* is formally defined in Table 87.~~

~~Table 87 – OPC Binary Definition~~

~~Attribute~~	**~~Value~~**		
~~BrowseName~~	~~OPC Binary~~		
~~References~~	**~~NodeClass~~**	**~~BrowseName~~**	**~~Comment~~**
~~HasTypeDefinition~~	~~ObjectType~~	~~DataTypeSystemType~~	~~Defined in 6.8~~

~~8.2.11 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 88.~~

~~Table 88 – XML Schema Definition~~

~~Attribute~~	~~Value~~		
~~BrowseName~~	~~XML Schema~~		
~~References~~	~~NodeClass~~	~~BrowseName~~	~~Comment~~
~~HasTypeDefinition~~	~~ObjectType~~	~~DataTypeSystemType~~	~~Defined in 6.8~~

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*.

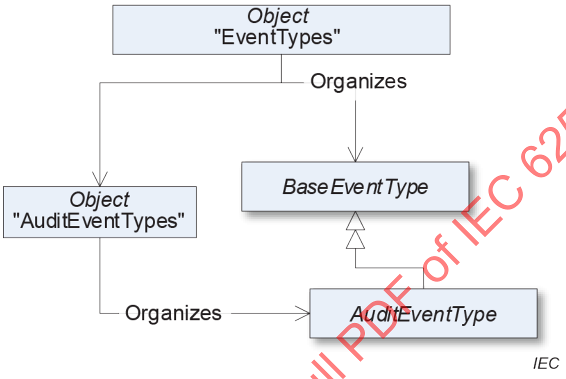


Figure 7 – EventTypes organization

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.

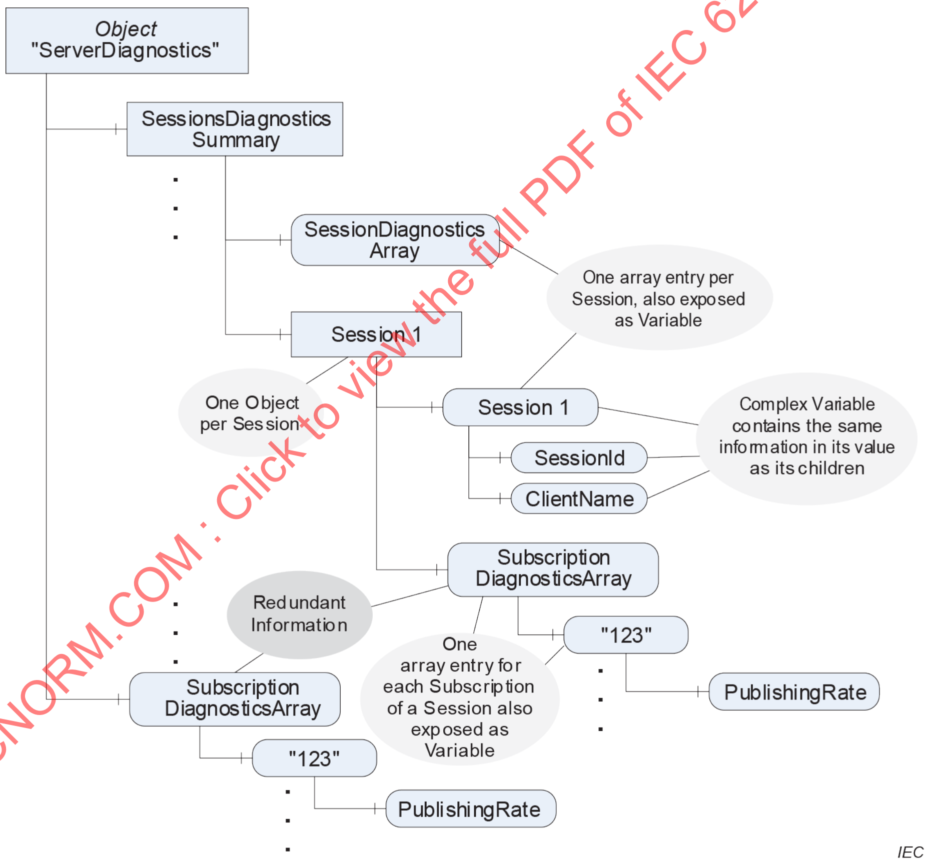

Object
"ServerDiagnostics"
SessionsDiagnostics Summary
SessionDiagnostics Array
One array entry per Session, also exposed as Variable
Session 1
One Object per Session
Session 1
Complex Variable contains the same information in its value as its children
SessionId
ClientName
Subscription DiagnosticsArray
Redundant Information
"123"
One array entry for each Subscription of a Session also exposed as Variable
PublishingRate
Subscription DiagnosticsArray
"123"
PublishingRate
IEC

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		
~~HasProperty~~	~~Variable~~	~~ServerArray~~	~~String[]~~	~~PropertyType~~	~~Mandatory~~
~~HasProperty~~	~~Variable~~	~~NamespaceArray~~	~~String[]~~	~~PropertyType~~	~~Mandatory~~
~~HasComponent~~	~~Variable~~	~~ServerStatus~~[a]	~~ServerStatusDataType~~	~~ServerStatusType~~	~~Mandatory~~
~~HasProperty~~	~~Variable~~	~~ServiceLevel~~	~~Byte~~	~~PropertyType~~	~~Mandatory~~
~~HasComponent~~	~~Object~~	~~ServerCapabilities~~[a]	~~--~~	~~ServerCapabilities~~	~~Mandatory~~
~~HasComponent~~	~~Object~~	~~ServerDiagnostics~~[a]	~~--~~	~~ServerDiagnosticsType~~	~~Mandatory~~
~~HasComponent~~	~~Object~~	~~VendorServerInfo~~	~~--~~	~~vendor-specific~~[b]	~~Mandatory~~
~~HasComponent~~	~~Object~~	~~ServerRedundancy~~[a]	~~--~~	~~depends on supported redundancy~~[c]	~~Mandatory~~

[a] ~~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.~~

[b] ~~Shall be the *VendorServerInfo ObjectType* or one of its subtypes.~~

[c] ~~Shall be the *ServerRedundancyType* or one of its subtypes.~~

8.4 ModellingRule Objects

8.4.1 ExposesItsArray

The *ModellingRule ExposesItsArray* is defined in IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-4.

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 IEC 62541-4
Bad_UserAccessDenied	Defined in IEC 62541-4 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 IEC 62541-4.

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 IEC 62541-4
Bad_UserAccessDenied	Defined in IEC 62541-4 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 IEC 62541-4.

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 IEC 62541-4
Bad_InvalidState	Defined in IEC 62541-4 This is returned when a *Subscription* already contains *MonitoredItems*.
Bad_UserAccessDenied	Defined in IEC 62541-4 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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~~	~~HasDescription~~	~~Defined in 11.14~~
HasSubtype	ReferenceType	GeneratesEvent	Defined in 11.16

11.4 HasChild

This standard *ReferenceType* is defined in IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 HasDescription~~

~~This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 110.~~

~~**Table 110 – HasDescription ReferenceType**~~

~~Attributes~~	~~Value~~		
~~BrowseName~~	~~HasDescription~~		
~~InverseName~~	~~DescriptionOf~~		
~~Symmetric~~	~~False~~		
~~IsAbstract~~	~~False~~		
~~**References**~~	~~**NodeClass**~~	~~**BrowseName**~~	~~**Comment**~~

11.14 HasEventSource

This standard *ReferenceType* is defined in IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3

IEC 62541-3 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 115.

Table 115 – IEC 62541-3 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
~~Time~~

BrowseName
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	~~TRUE~~ 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 IEC 62541-4

IEC 62541-4 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 130.

Table 130 – IEC 62541-4 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 IEC 62541-4.

The *AddNodesItem* is a structure that is defined as the type of the nodesToAdd parameter of the AddNodes *Service* in IEC 62541-4.

The *AddReferencesItem* is a structure that is defined as the type of the referencesToAdd parameter of the AddReferences *Service* in IEC 62541-4.

The *DeleteNodesItem* is a structure that is defined as the type of the nodesToDelete parameter of the DeleteNodes *Service* in IEC 62541-4.

The *DeleteReferencesItem* is a structure that is defined as the type of the referencesToDelete parameter of the DeleteReferences *Service* in IEC 62541-4.

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 IEC 62541-4.
WARM_2	Warm means that the server supports warm redundancy as defined in IEC 62541-4.
HOT_3	Hot means that the server supports hot redundancy as defined in IEC 62541-4.
TRANSPARENT_4	Transparent means that the server supports transparent redundancy as defined in IEC 62541-4.
HOT_AND_MIRRORED_5	HotAndMirrored means that the server supports HotAndMirrored redundancy as defined in IEC 62541-4.

See IEC 62541-4 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* ~~has shut down~~ initiated a shutdown or is in the process of shutting down. ~~Depending on the implementation, this might or might not be visible to *Clients*.~~ This *ServerState* is intended as an indication to *Clients* connected to the *Server* to orderly disconnect from the *Server* before the *Server* completes the shutdown.
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 ~~servers~~ 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 IEC 62541-4, 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 IEC 62541-4, 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* shutdown is initated, the state changes to SHUTDOWN_4 and the actual shutdown 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.
~~currentPublishTimerExpirations~~	~~UInt32~~	~~The number of publish timer expirations when there are data to be sent, but there are no publish requests for this session. The value shall be 0 if there are no data to be sent or publish requests queued.~~

Name	Type	Description
~~totalRequestsCount~~ totalRequestCount	ServiceCounter DataType	Counter of all *Services*, identifying the number of received requests of any *Services* on the session.
~~unauthorizedRequestsCount~~ unauthorizedRequestCount	UInt32	Counter of all *Services*, identifying the number of *Service* requests that were rejected due to authorization failure
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.

Name	Type	Description
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 ~~should not~~ shall only be ~~made~~ accessible ~~to all users, but only to~~ 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 ~~(user name and password, X.509, Kerberos)~~ 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~~, for example~~. The String shall be "XML", "JSON" or "UA Binary".
transportProtocol	String	~~Which transport protocol is used For example TCP or HTTP.~~ 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 IEC 62541-6.
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>ModelChangeStructureDataType</td><td>structure</td><td></td></tr>
<tr><td>affected</td><td>NodeId</td><td><i>NodeId</i> of the <i>Node</i> that was changed. The client should assume that the <i>affected Node</i> has been created or deleted, had a <i>Reference</i> added or deleted, or the <i>DataType</i> has changed as described by the <i>verb</i>.</td></tr>
<tr><td>affectedType</td><td>NodeId</td><td>If the <i>affected Node</i> was an <i>Object</i> or <i>Variable</i>, <i>affectedType</i> contains the <i>NodeId</i> of the <i>TypeDefinitionNode</i> of the <i>affected Node</i>. 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 <i>verb</i> 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 <i>affected Node</i> has been added.</td></tr>
<tr><td>NodeDeleted</td><td>1</td><td>Indicates the <i>affected Node</i> has been deleted.</td></tr>
<tr><td>ReferenceAdded</td><td>2</td><td>Indicates a <i>Reference</i> has been added. The affected <i>Node</i> may be either a <i>SourceNode</i> or <i>TargetNode</i>. Note that an added bidirectional <i>Reference</i> is reflected by two ~~ChangeStructures~~ changes.</td></tr>
<tr><td>ReferenceDeleted</td><td>3</td><td>Indicates a <i>Reference</i> has been deleted. The affected <i>Node</i> may be either a <i>SourceNode</i> or <i>TargetNode</i>. Note that a deleted bidirectional <i>Reference</i> is reflected by two ~~ChangeStructures~~ changes.</td></tr>
<tr><td>DataTypeChanged</td><td>4</td><td>This verb may be used only for affected <i>Nodes</i> that are <i>Variables</i> or <i>VariableTypes</i>. It indicates that the <i>DataType Attribute</i> 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 IEC 62541-3 for details).
Note that all <i>verbs</i> shall always be considered in the context where the ModelChangeStructureDataType is used. A NodeDeleted may indicate that a <i>Node</i> was removed from a view but still exists in other <i>Views</i>.</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 IEC 62541-4.
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

Annex A 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*.

Annex A 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 IEC 62541-3.

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

The following clauses describe the design decisions made while modelling the *Server Object*. General *DataTypes*, *VariableTypes* and *ObjectTypes* such as the *EventTypes* described in this document 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 document and the *NodeId* is based on the symbolic name in IEC 62541-6. 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 IEC 62541-3 about the semantic difference between *Properties* and *DataVariables* are not applicable for the information provided about the *Server* the rules described in IEC 62541-3 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 IEC 62541, 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 document. 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 document, 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 document, clients will look there for server-specific information.

Annex B
(normative)

StateMachines

B.1 General

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

Annex B is an integral part of this document, that is, the types defined in Annex B 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 Annex B, 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 Annex B is to provide an appropriate foundation for state machines needed for IEC 62541-9 and IEC 62541-10. 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 Annex B can be extended to support such concepts.

The following clauses describe examples of state machines, define state machines in the context of Annex B 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 Annex B 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".

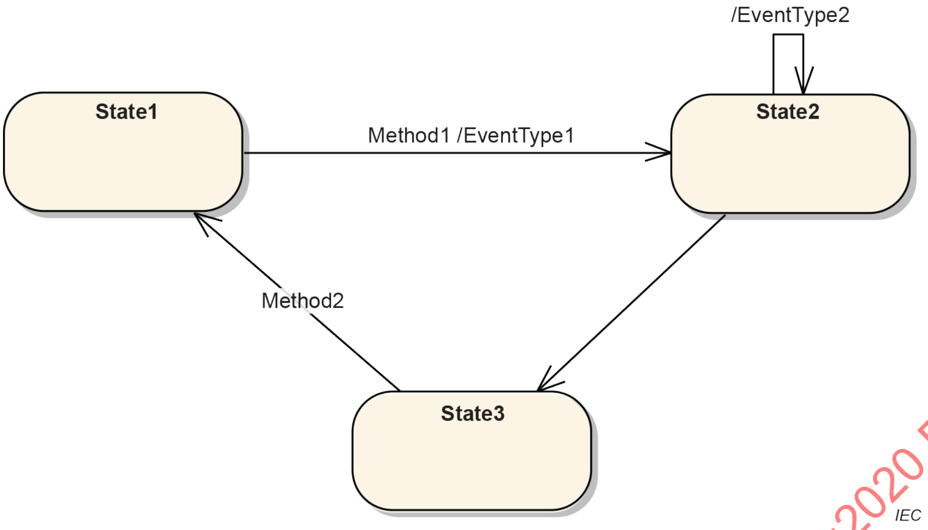


Figure B.1 – Example of a simple state machine

For simplicity, the state machines described in Annex B will only support causes in form of specifying *Methods* that ~~have to~~ shall 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.

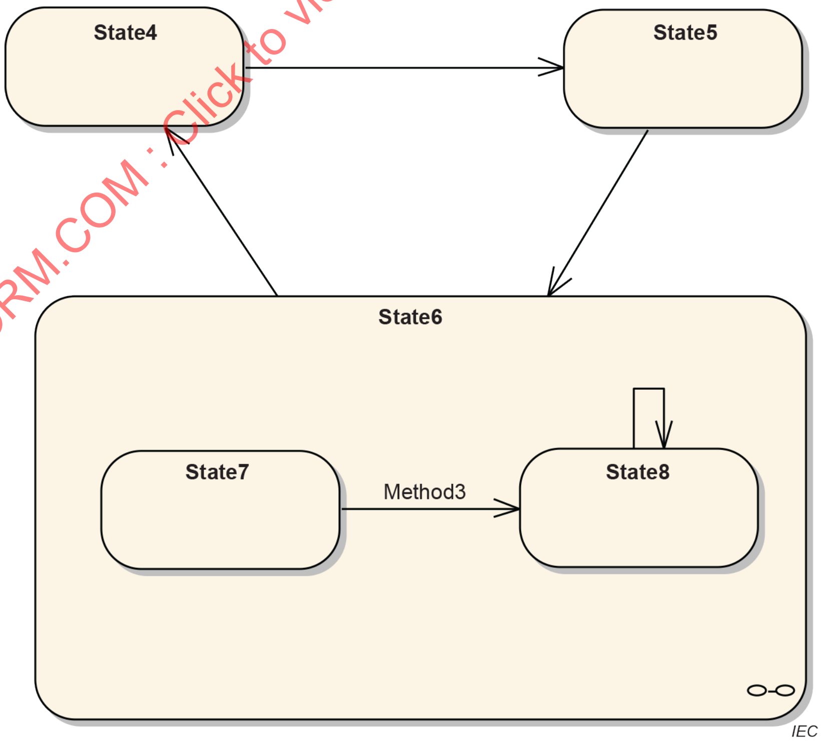


Figure B.2 – Example of a state machine having a sub-machine

B.3 Definition of state machine

The infrastructure of state machines defined in Annex B only deals with the basics of state machines needed to support IEC 62541-9 and IEC 62541-10. The intention is to keep the basic simple but extensible.

For the state machines defined in Annex B 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 Annex B 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 Annex B 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 Annex B 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 Annex B 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 IEC 62541-9 and IEC 62541-10 for additional examples of *StateMachines*.

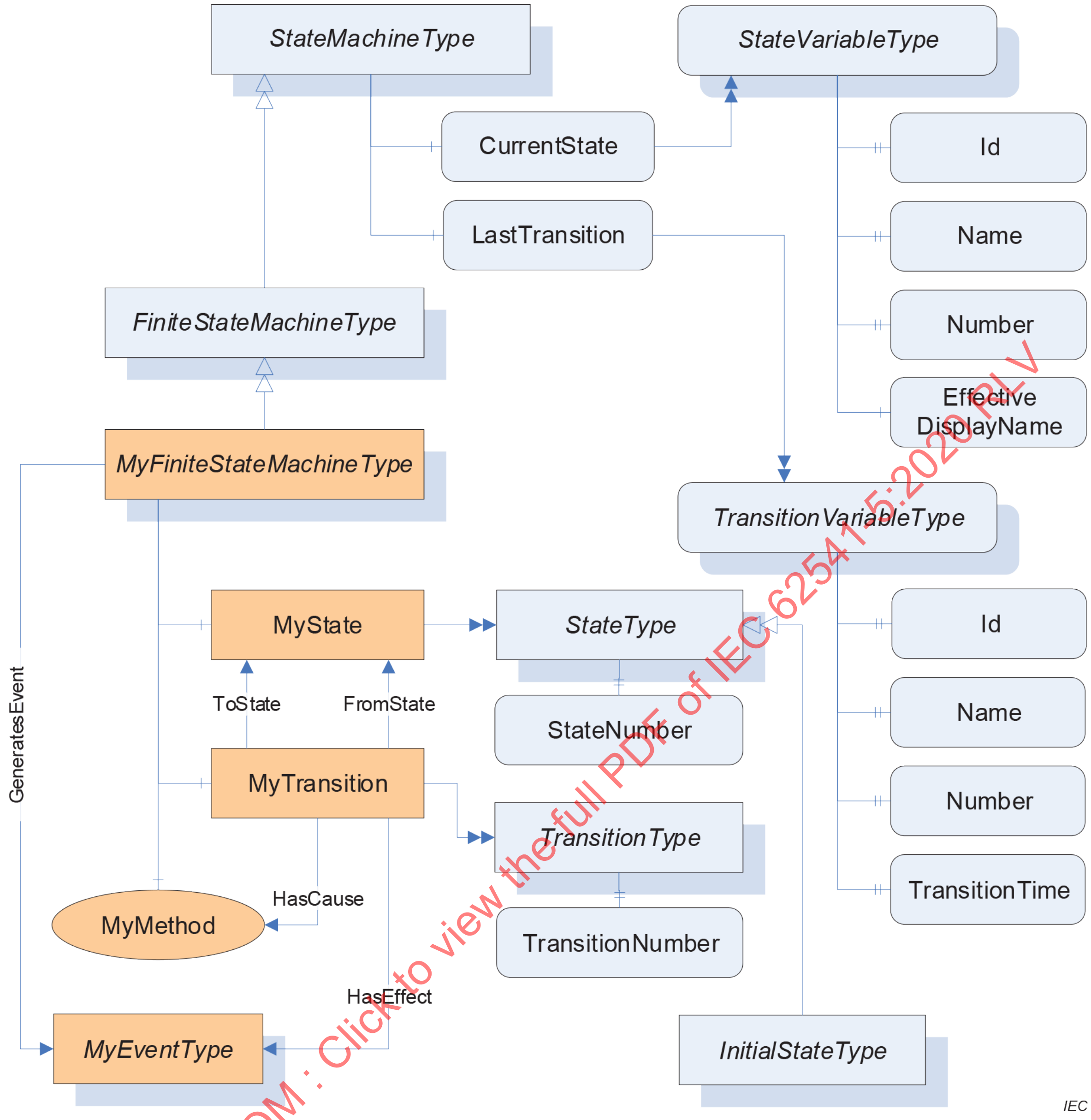


Figure B.3 – The StateMachine Information Model

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 IEC 62541-4. 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.

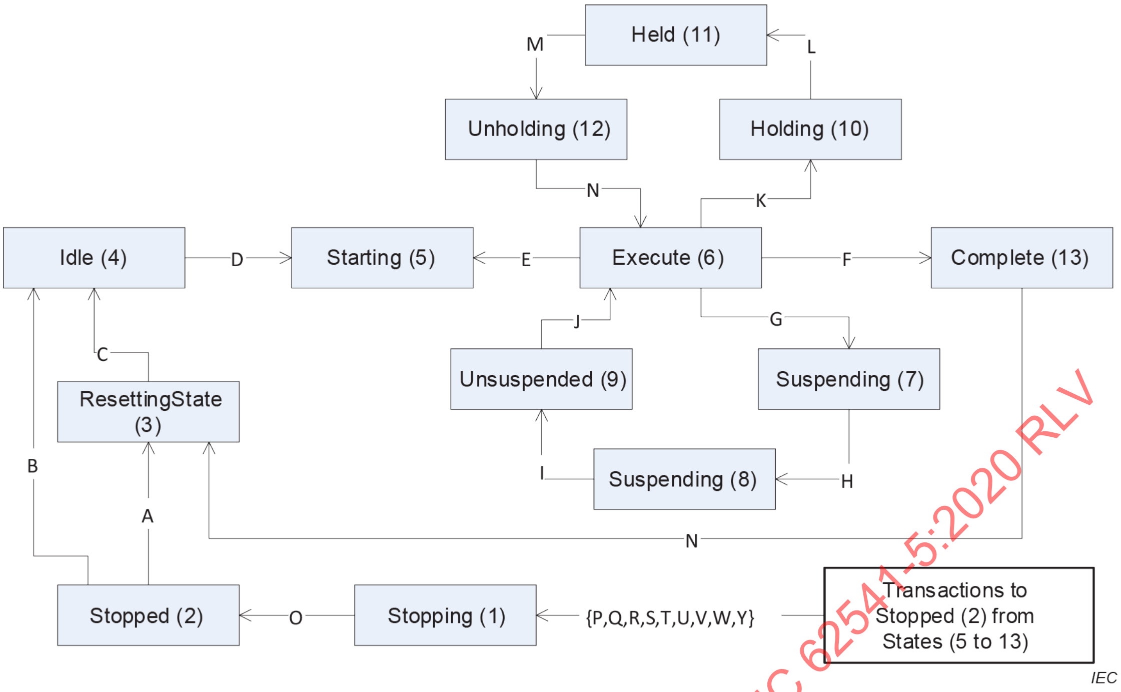


Figure B.4 – Example of a FiniteStateMachine type

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.

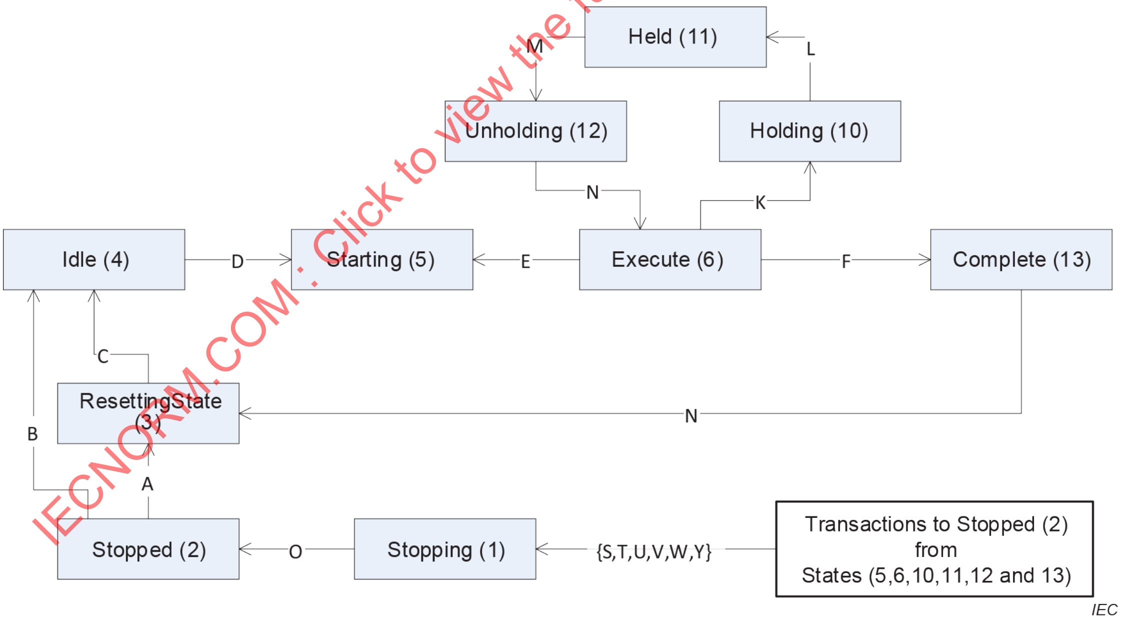


Figure B.5 – Example of a FiniteStateMachine instance

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 in B.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.

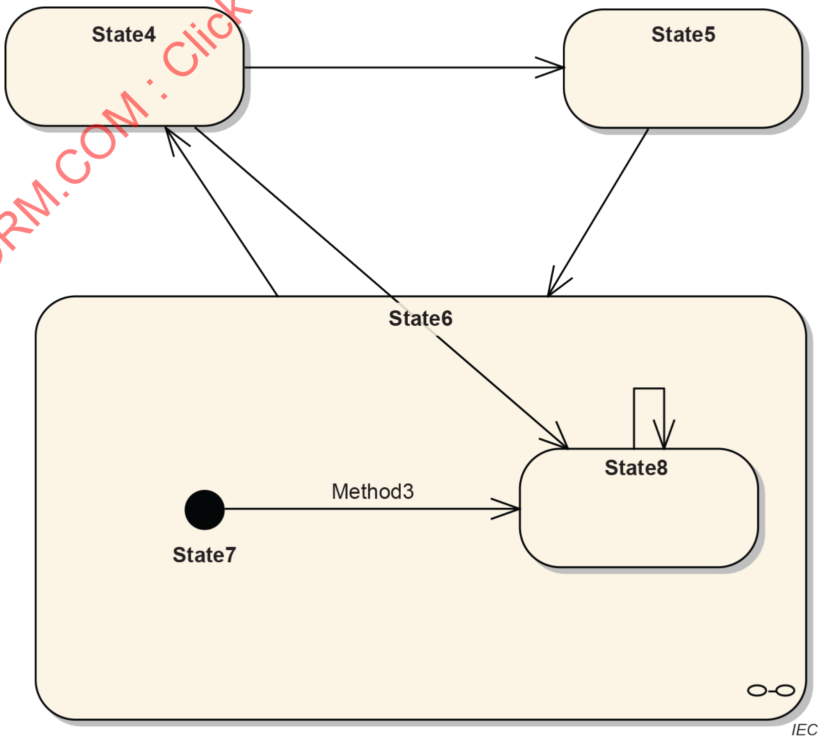


Figure B.6 – Example of an initial State in a sub-machine

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 Annex B 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 Annex B 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~~ shall be used. If the Transition occurs between a *StateMachine* and a *SubStateMachine*, then the *NodeId* of the *StateMachine* ~~has to~~ shall 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~~ shall 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

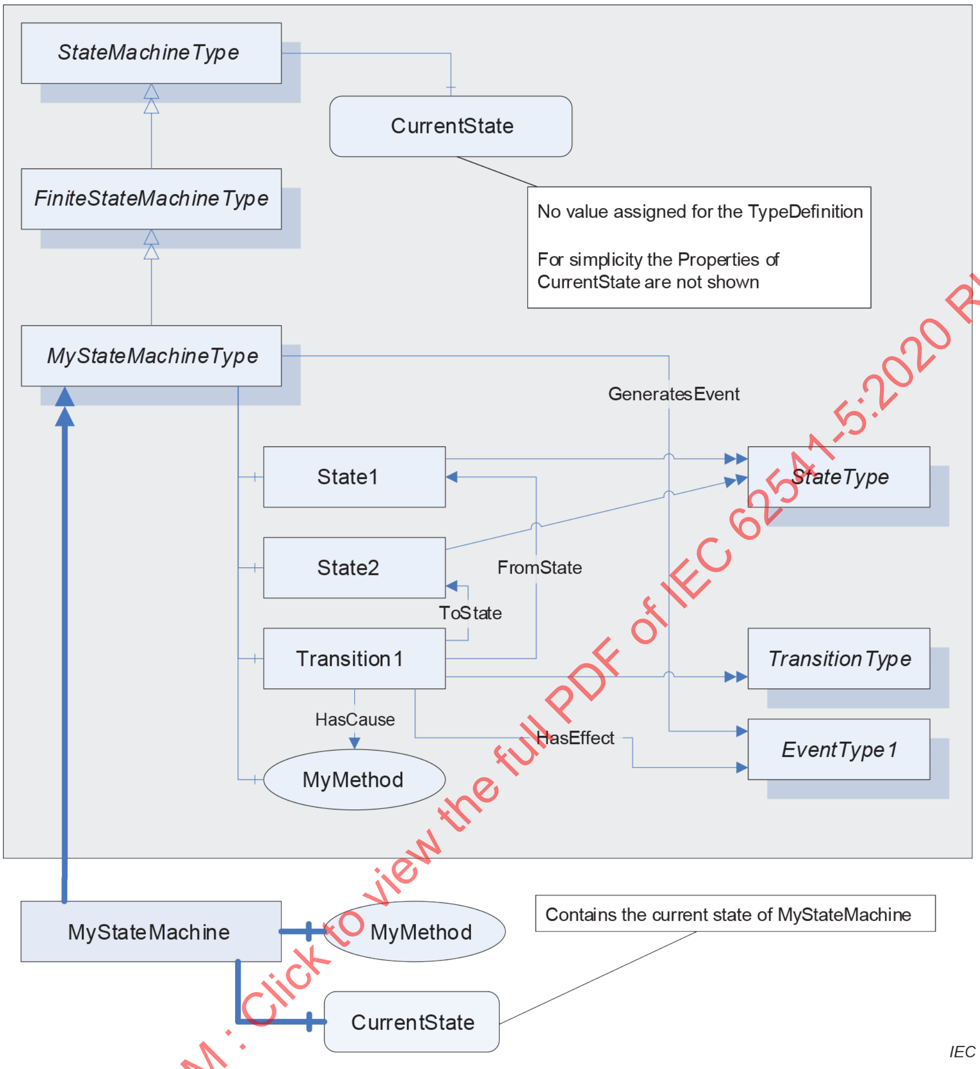


Figure B.7 – Example of a StateMachineType using inheritance

In Figure B.7 an example of a *StateMachine* is given using the Notation defined in IEC 62541-3. 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~~ SubStateMachine using inheritance

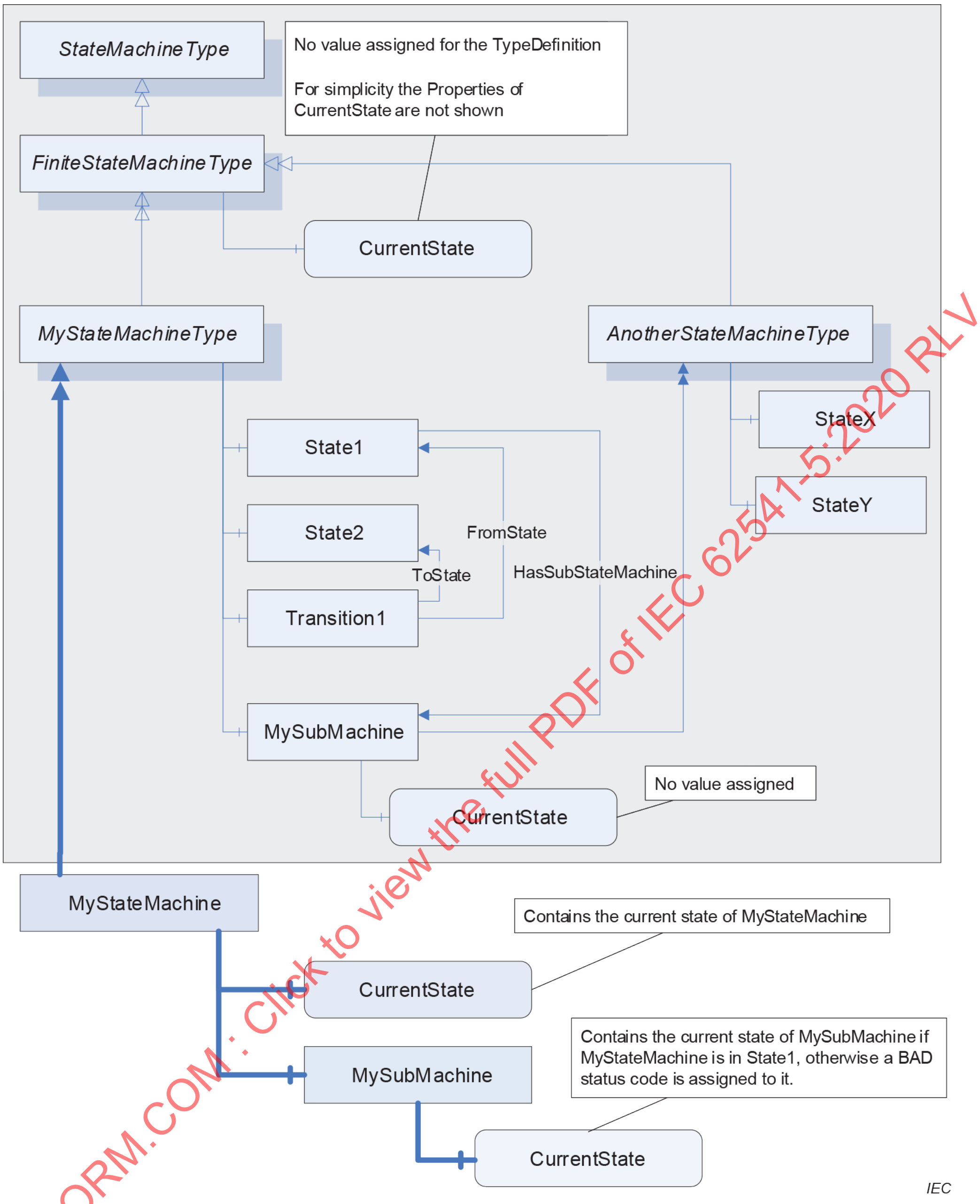

StateMachineType
No value assigned for the TypeDefinition
For simplicity the Properties of CurrentState are not shown
FiniteStateMachineType
CurrentState
MyStateMachineType
AnotherStateMachineType
StateX
StateY
State1
State2
FromState
ToState
HasSubStateMachine
Transition1
MySubMachine
No value assigned
CurrentState
MyStateMachine
Contains the current state of MyStateMachine
CurrentState
MySubMachine
Contains the current state of MySubMachine if MyStateMachine is in State1, otherwise a BAD status code is assigned to it.
CurrentState
IEC

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

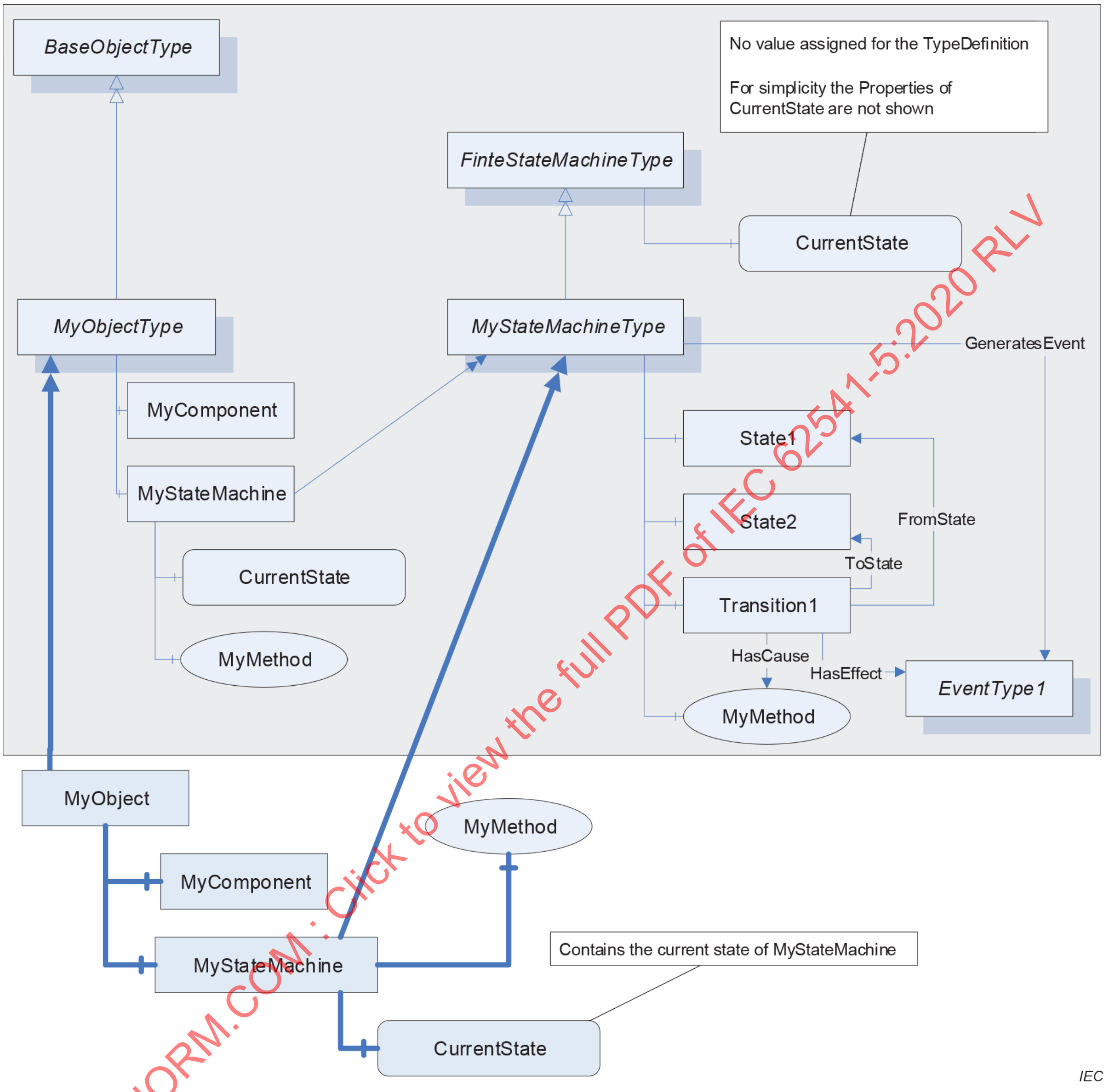


Figure B.9 – Example of a StateMachineType using containment

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.

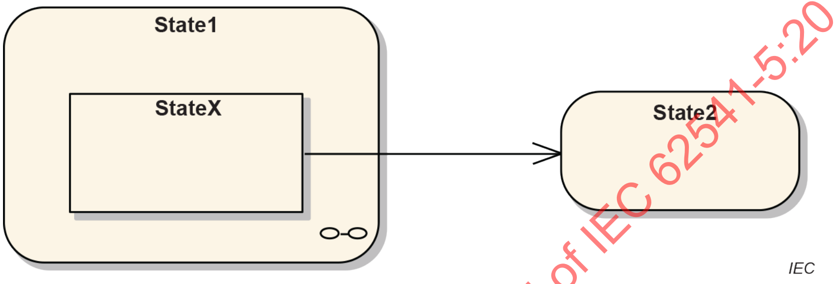


Figure B.10 – Example of a ~~state machine~~ StateMachine with Transitions from sub-states

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.

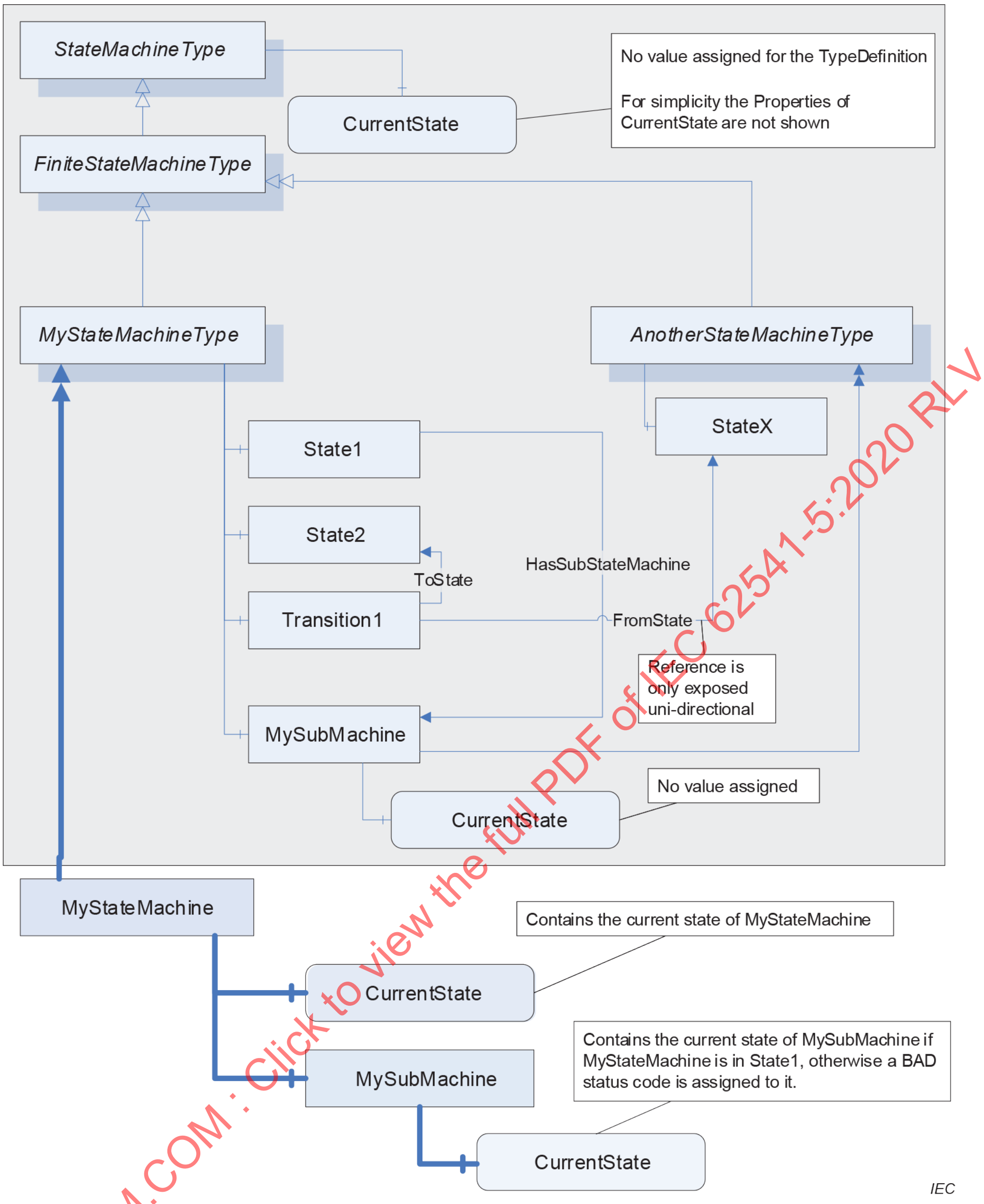

StateMachineType
CurrentState
No value assigned for the TypeDefinition
For simplicity the Properties of CurrentState are not shown
FiniteStateMachineType
MyStateMachineType
AnotherStateMachineType
StateX
State1
State2
ToState
HasSubStateMachine
Transition1
FromState
Reference is only exposed uni-directional
MySubMachine
No value assigned
CurrentState
MyStateMachine
Contains the current state of MyStateMachine
CurrentState
MySubMachine
Contains the current state of MySubMachine if MyStateMachine is in State1, otherwise a BAD status code is assigned to it.
CurrentState
IEC

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

Annex C
(normative)

File Transfer

C.1 Overview

Annex C 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 ~~an~~ 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.

~~The *Services* defined in the NodeManagement Service Set can be used to create or delete files in an *AddressSpace*. 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. Deleting the OPC UA *Object* does not imply that the file is deleted from disk and a deletion from disk does not imply that the OPC UA *Object* is deleted.~~

~~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.~~

In addition to representing individual files Annex C 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 Annex C.

Annex C is an integral part of this document, that is, the types defined in Annex C 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

C.2.1 General

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.2.2		Mandatory
HasComponent	Method	Close	Defined in C.2.3		Mandatory
HasComponent	Method	Read	Defined in C.2.4		Mandatory
HasComponent	Method	Write	Defined in C.2.5		Mandatory
HasComponent	Method	GetPosition	Defined in C.2.6		Mandatory
HasComponent	Method	SetPosition	Defined in C.2.7		Mandatory

Size defines the size of the file in Bytes. When a file is opened for write ~~and the fileHandle is still valid~~ 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 IETF RFC 2046.

Note that all *Methods* on a file require a fileHandle, which is returned in the *Open Method*.

C.2.2 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 <i>mode</i> 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 IEC 62541-4 for a general description. File might be locked and thus not readable.
Bad_NotWritable	See IEC 62541-4 for a general description.
Bad_InvalidState	See IEC 62541-4 for a general description. The file is locked and thus not writable.
Bad_InvalidArgument~~s~~	See IEC 62541-4 for a general description. Mode setting is invalid.
Bad_NotFound	See IEC 62541-4 for a general description.
Bad_UnexpectedError	See IEC 62541-4 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.2.3 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 IEC 62541-4 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.2.4 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 only all data till the end of the file are returned. If the specified length is longer than the maximum allowed message size of the communication, only those data fitting into the message size are returned.~~ 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 IEC 62541-4 for a general description. Invalid file handle in call or non-positive length.
Bad_UnexpectedError	See IEC 62541-4 for a general description.
Bad_InvalidState	See IEC 62541-4 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.2.5 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 effect on the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.
Bad_NotWritable	See IEC 62541-4 for a general description. File might be locked and thus not writable.
Bad_InvalidState	See IEC 62541-4 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.2.6 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 IEC 62541-4 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.2.7 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 IEC 62541-4 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**	**ModellingRule**
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.

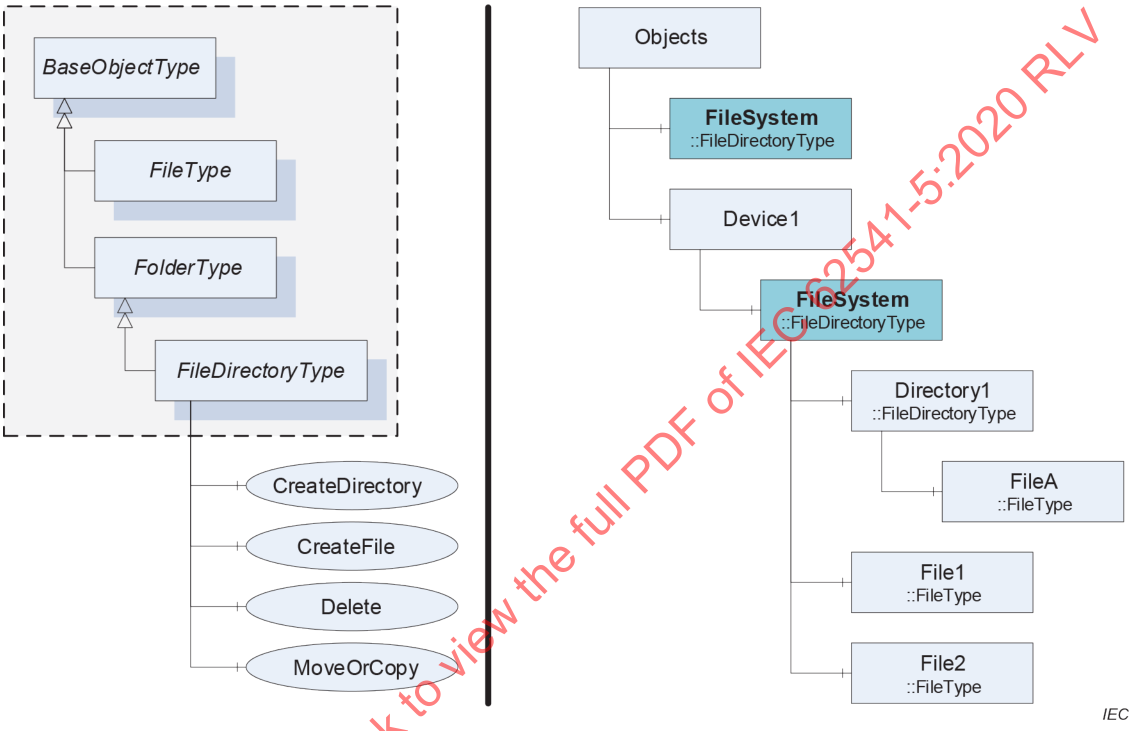


Figure C.1 – FileSystem example

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 IEC 62541-4 for a general description. A directory with the name already exists.
Bad_UserAccessDenied	See IEC 62541-4 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 IEC 62541-4 for a general description. A file with the name already exists.
Bad_UserAccessDenied	See IEC 62541-4 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 IEC 62541-4 for a general description. A file or directory with the provided NodeId is not organized by this object.
Bad_InvalidState	See IEC 62541-4 for a general description. The file or directory is locked and thus cannot be deleted.
Bad_UserAccessDenied	See IEC 62541-4 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 IEC 62541-4 for a general description. A file or directory with the name already exists.
Bad_NotFound	See IEC 62541-4 for a general description. A file or directory with the provided NodeId is not organized by this object.
Bad_InvalidState	See IEC 62541-4 for a general description. The file or directory is locked and thus cannot be moved or copied.
Bad_UserAccessDenied	See IEC 62541-4 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 browseable 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**	**ModellingRule**
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>		FileTransferStateMachineType	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 asynchronously 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 write 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.

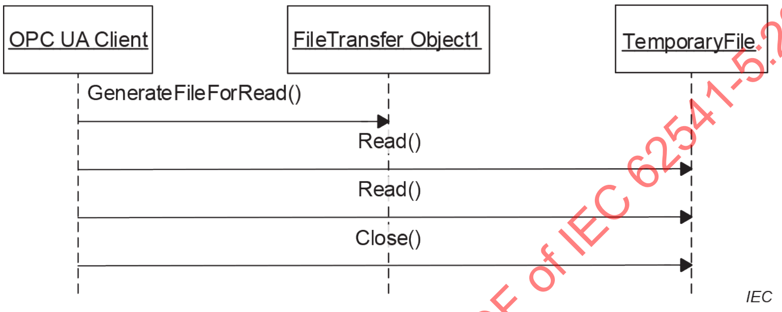


Figure C.2 – Read file transfer example sequence

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.

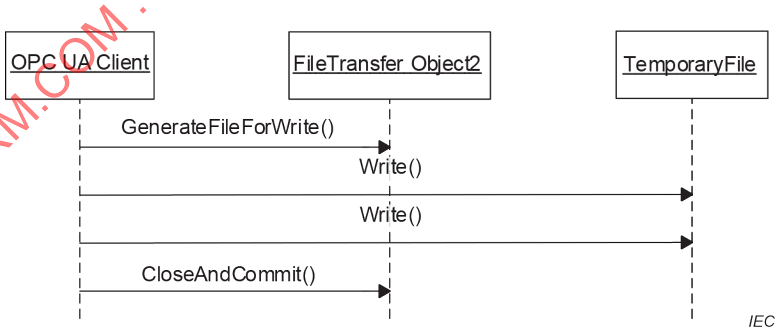


Figure C.3 – Write file transfer example sequence

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 asynchronously, 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 IEC 62541-4 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 IEC 62541-4 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 asynchronously, 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 IEC 62541-4 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.

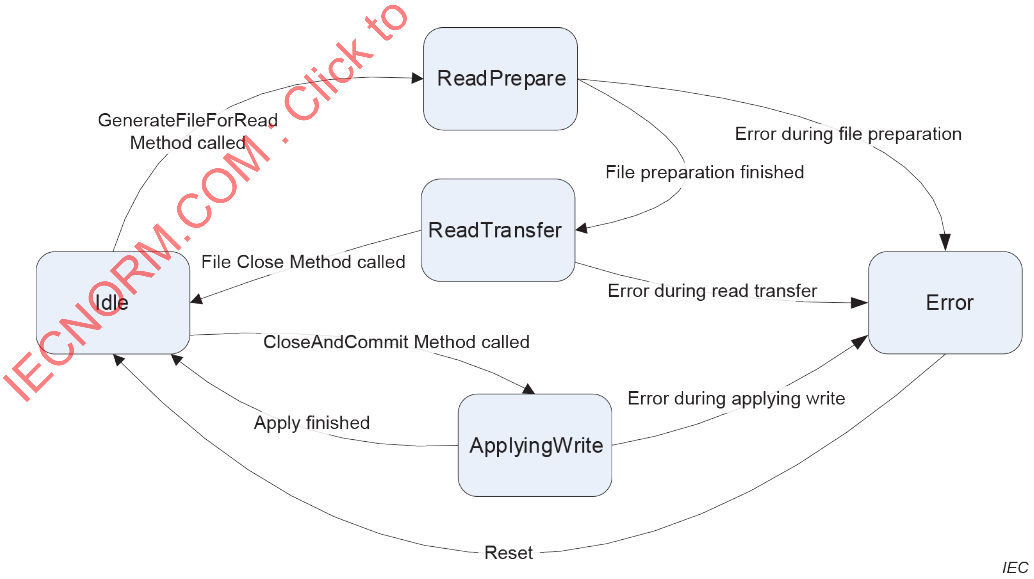

ReadPrepare
GenerateFileForRead
Method called
Error during file preparation
File preparation finished
ReadTransfer
File Close Method called
Idle
Error during read transfer
Error
CloseAndCommit Method called
Error during applying write
Apply finished
ApplyingWrite
Reset
IEC

Figure C.4 – File transfer States

The *FileTransferStateMachineType* and the related type are illustrated in Figure C.5.

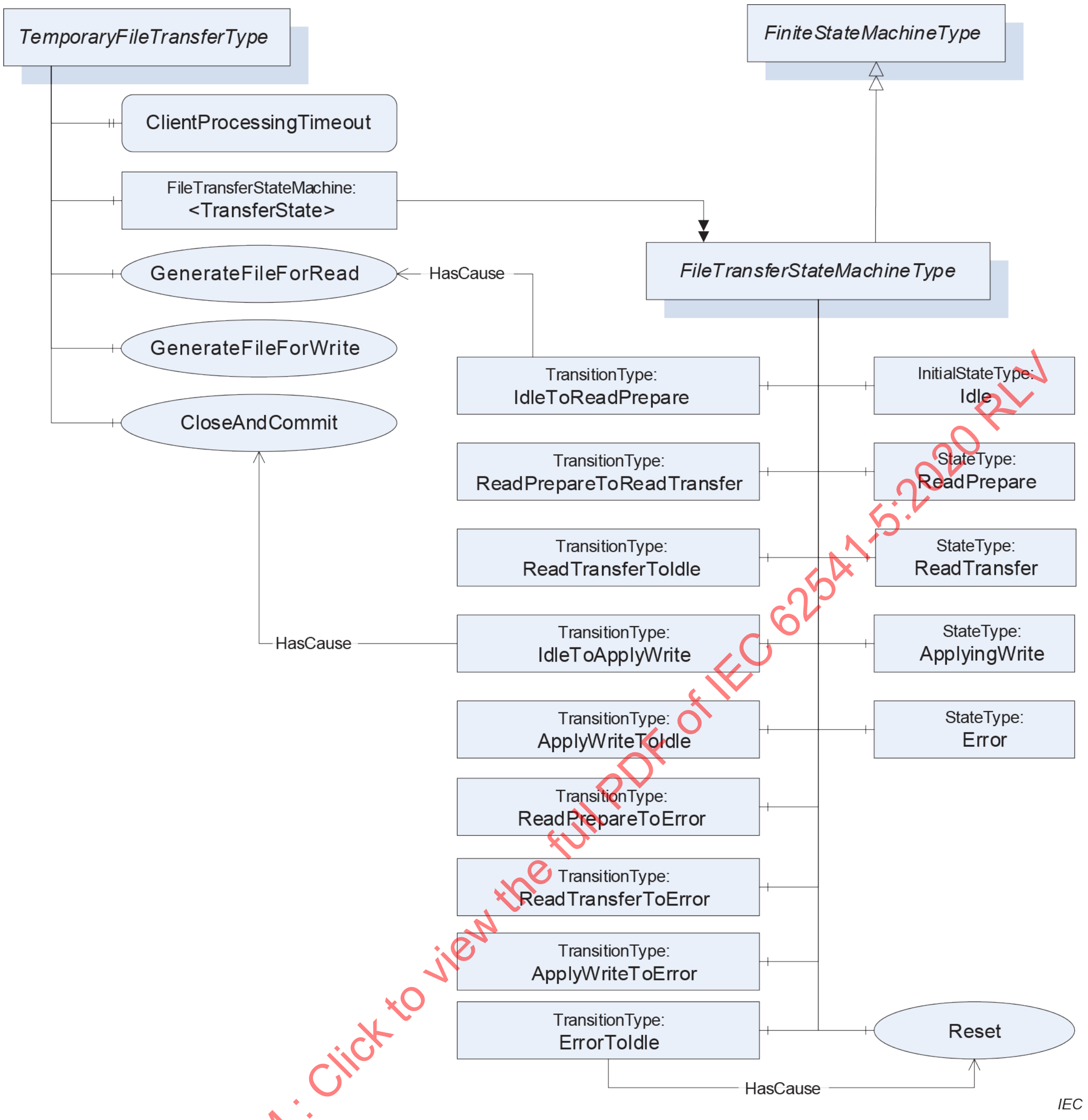


Figure C.5 – FileTransferStateMachineType

This *ObjectType* defines the StateMachine for asynchronously processing of temporary file transfers. It is formally defined in Table C.17. The transitions are formally defined in Table C.18.

Table C.17 – FileTransferStateMachineType

Attribute	**Value**				
BrowseName	FileTransferStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
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

Annex D defines a way to provide encoding information for custom *DataTypes*. In previous releases of the specification this approach was defined in IEC 62541-3. In IEC 62541-3 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

IEC 62541-3 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*.

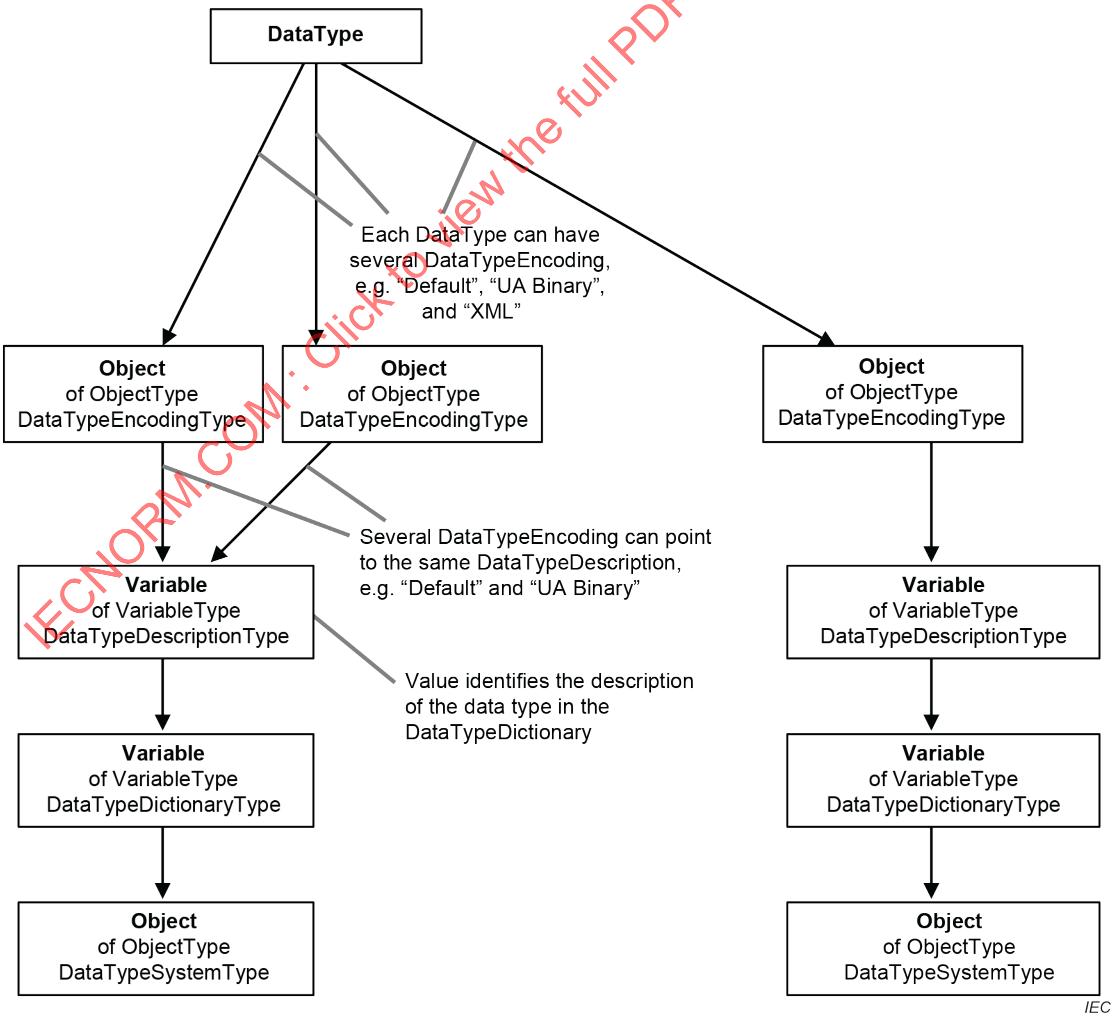

DataType
Each DataType can have several DataTypeEncoding, e.g. "Default", "UA Binary", and "XML"
Object of ObjectType DataTypeEncodingType
Object of ObjectType DataTypeEncodingType
Object of ObjectType DataTypeEncodingType
Several DataTypeEncoding can point to the same DataTypeDescription, e.g. "Default" and "UA Binary"
Variable of VariableType DataTypeDescriptionType
Variable of VariableType DataTypeDescriptionType
Value identifies the description of the data type in the DataTypeDictionary
Variable of VariableType DataTypeDictionaryType
Variable of VariableType DataTypeDictionaryType
Object of ObjectType DataTypeSystemType
Object of ObjectType DataTypeSystemType
IEC

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 recognize the *DataTypeSystem* to parse any of the type description information. OPC UA *Clients* that do not recognize 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 of how *DataTypes* are modelled in the *AddressSpace*.

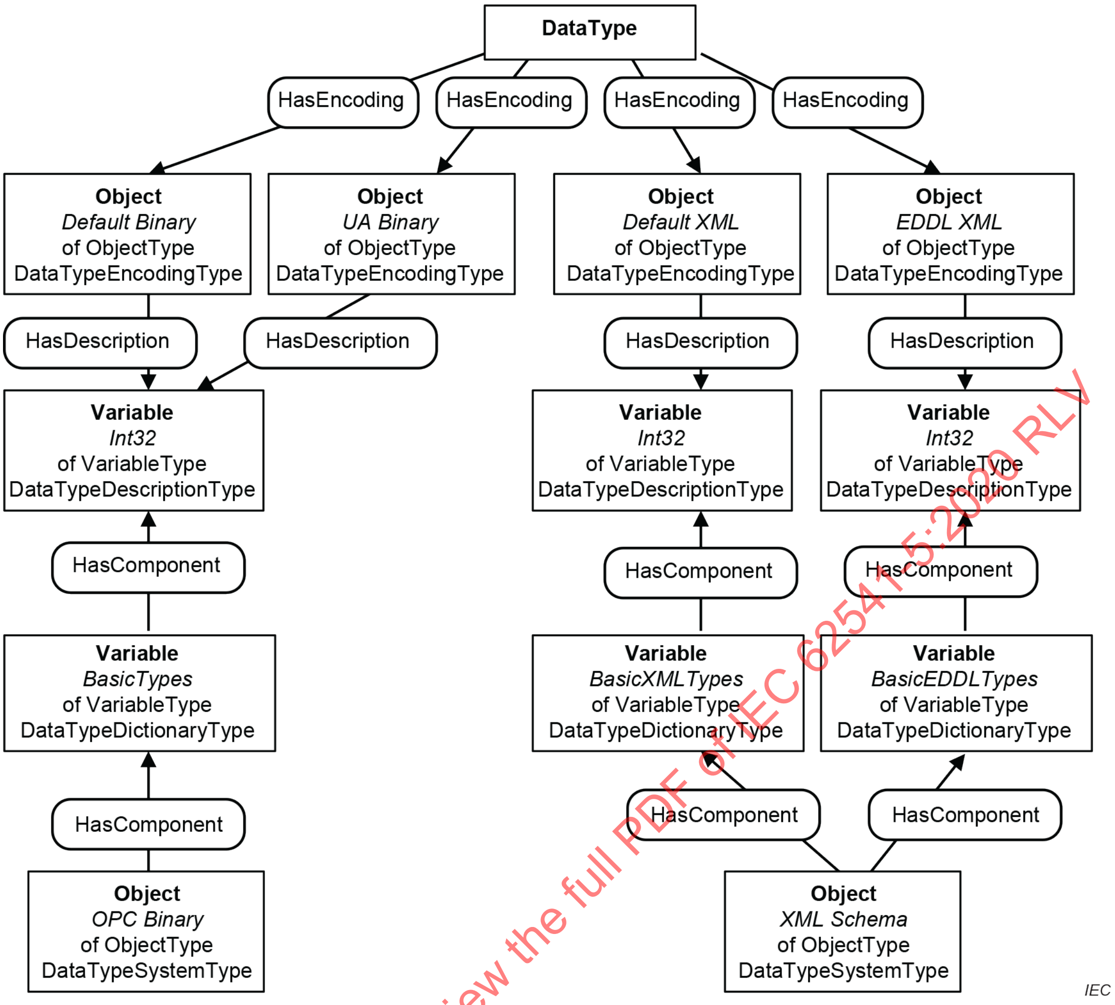


Figure D.2 – Example of DataType modelling

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 document 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 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*.

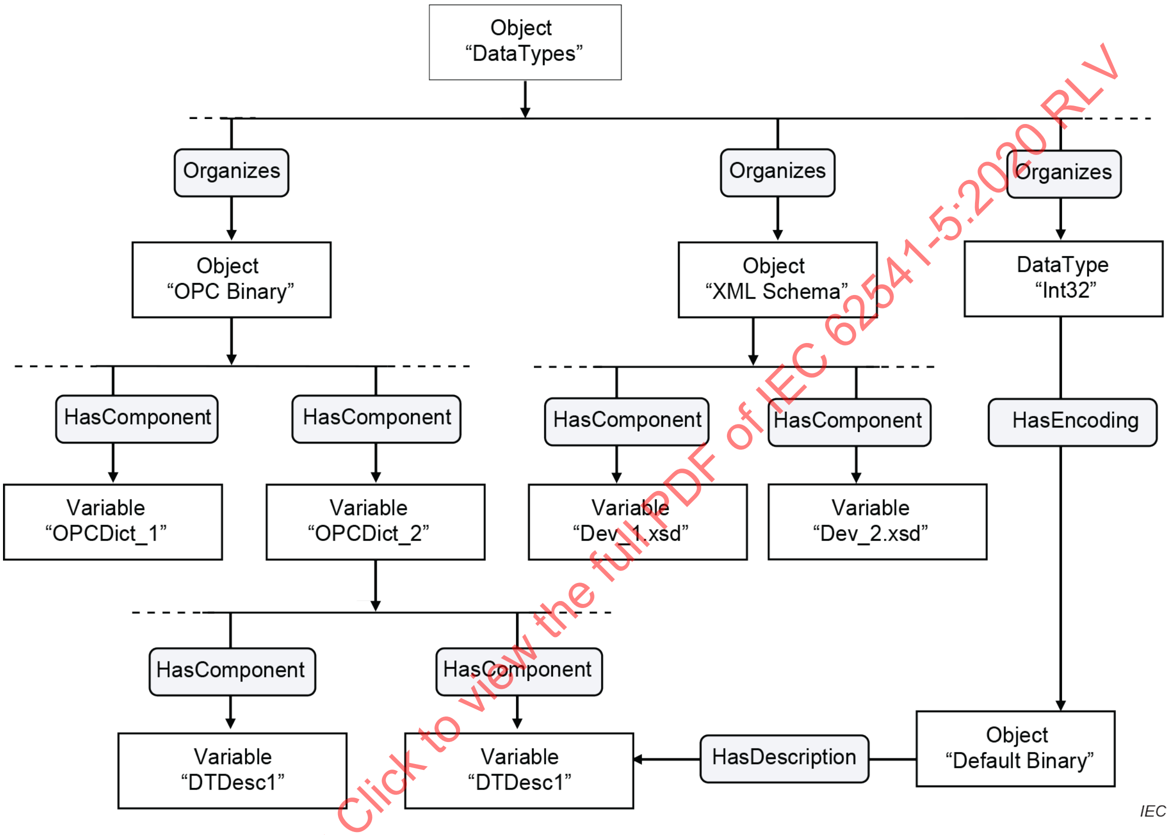


Figure D.3 – DataTypes organization

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 an *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	String				
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 IEC 62541-3. 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 E 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 illustrates the structure of an OPC Binary *TypeDictionary*.

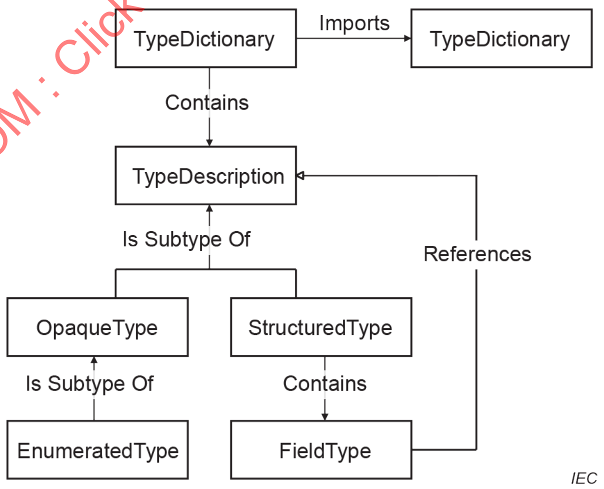


Figure E.1 – OPC Binary Dictionary structure

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 specify 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*.
SwitchValue	xs:unsignedInt	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.

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></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:
<table>
<tr><td>Equal</td><td><i>SwitchField</i> is equal to the <i>SwitchValue</i>.</td></tr>
<tr><td>GreaterThan</td><td><i>SwitchField</i> is greater than the <i>SwitchValue</i>.</td></tr>
<tr><td>LessThan</td><td><i>SwitchField</i> is less than the <i>SwitchValue</i>.</td></tr>
<tr><td>GreaterThanOrEqual</td><td><i>SwitchField</i> is greater than or equal to the <i>SwitchValue</i>.</td></tr>
<tr><td>LessThanOrEqual</td><td><i>SwitchField</i> is less than or equal to the <i>SwitchValue</i>.</td></tr>
<tr><td>NotEqual</td><td><i>SwitchField</i> is not equal to the <i>SwitchValue</i>.</td></tr>
</table>
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 <i>TypeDescription</i> 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 <i>TypeDescription</i> referenced by this field shall either have a fixed byte order (i.e. byte order is not significant or explicitly specified) or the containing <i>StructuredType</i> 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 <i>TypeDictionary</i> may add their own attributes to any <i>FieldType</i> 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 of possible 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 recognize 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	An 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 nanosecond 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

E.4.1 A 128-bit signed integer

```
<opc:OpaqueType Name="Int128" LengthInBits="128" ByteOrderSignificant="true">
  <opc:Documentation>A 128-bit signed integer.</opc:Documentation>
</opc:OpaqueType>
```

E.4.2 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.

E.4.3 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.

E.4.4 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.

E.4.5 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.

E.4.6 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 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.

E.4.7 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.

E.4.8 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 ISO/IEC/IEEE 60559:2011 single precision floating point
value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Double" LengthInBits="64" ByteOrderSignificant="true">
    <opc:Documentation>An ISO/IEC/IEEE 60559:2011 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	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
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 is 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.

IEC 62541-3 defines well-known *Roles*. If this *Method* is used to add a well-known *Role*, the name of the *Role* from IEC 62541-3 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 IEC 62541-6.

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	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					
HasProperty	*Variable*	Identities	IdentityMappingRuleType []	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 *EndpointsExclude 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 included 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 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.

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 Identity Mapping Type	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* shall meet for a *Session* to be mapped to the *Role*. The meaning of the criteria depends on the *mappingType*. The criteria are an empty string 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

<table>
<tr><th>Attribute</th><th colspan="5">Value</th></tr>
<tr><td>BrowseName</td><td colspan="5">RoleMappingRuleChangedAuditEventType</td></tr>
<tr><td>IsAbstract</td><td colspan="5">True</td></tr>
<tr><td><b>References</b></td><td><b>NodeClass</b></td><td><b>BrowseName</b></td><td><b>DataType</b></td><td><b>TypeDefinition</b></td><td><b>ModellingRule</b></td></tr>
<tr><td colspan="6">Subtype of the <i>AuditUpdateMethodEventType</i> defined in 6.4.27</td></tr>
</table>

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantics are defined in 6.4.27.

IEC 62541-5

Edition 3.0 2020-07

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour
inside

OPC unified architecture –
Part 5: Information Model

Architecture unifiée OPC –
Partie 5: Modèle d'information

IEC 62541-5:2020-07(en-fr)

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

OPC UNIFIED ARCHITECTURE –

Part 5: Information Model

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62541-5 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2015. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) Added Annex F on User Authentication. Describes the Role Information Model that also allows configuration of Roles.
b) Added new data types: "Union", "Decimal", "OptionSet", "DateString", "TimeString", "DurationString", NormalizedString", "DecimalString", and "AudioDataType".
c) Added Method to request a state change in a Server.
d) Added Method to set Subscription to persistent mode.

e) Added Method to request resending of data from a Subscription.

f) Added concept allowing to temporarily create a file to write to or read from a server in C.4.

g) Added new Variable type to support Selection Lists.

h) Added optional properties to FiniteStateMachineType to expose currently available states and transitions.

i) Added UrisVersion Property to ServerType. This version information can be used for session-less service invocation.

The text of this standard is based on the following documents:

FDIS	Report on voting
65E/717/FDIS	65E/733/RVD

Full information on the voting for the approval of this International Standard can be found in the report on voting indicated in the above table.

This document has been drafted in accordance with the ISO/IEC Directives, Part 2.

Throughout this document and the other parts of the IEC 62541 series, certain document conventions are used:

Italics are used to denote a defined term or definition that appears in Clause 3 in one of the parts of the series.

Italics are also used to denote the name of a service input or output parameter or the name of a structure or element of a structure that are usually defined in tables.

The *italicized terms and names* are also often written in camel-case (the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound). For example the defined term is *AddressSpace* instead of Address Space. This makes it easier to understand that there is a single definition for *AddressSpace*, not separate definitions for Address and Space.

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, can be found on the IEC website.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under "http://webstore.iec.ch" in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

OPC UNIFIED ARCHITECTURE –

Part 5: Information Model

1 Scope

This part of IEC 62541 defines the Information Model of the OPC Unified Architecture. The Information Model describes standardized *Nodes* of a *Server*'s *AddressSpace*. These *Nodes* are standardized types as well as standardized 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.

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

IEC 62541-3, *OPC Unified Architecture – Part 3: Address Space Model*

IEC 62541-4, *OPC Unified Architecture – Part 4: Services*

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

IEC 62541-9, *OPC Unified Architecture – Part 9: Alarms and Conditions*

IEC 62541-10, *OPC Unified Architecture – Part 10: Programs*

IEC 62541-11, *OPC Unified Architecture – Part 11: Historical Access*

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

IETF RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
http://www.ietf.org/rfc/rfc2045.txt

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

IETF RFC 2047, Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text
http://www.ietf.org/rfc/rfc2047.txt

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/

IETF RFC 3629: UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc3629.txt

3 Terms, definitions, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1 and IEC 62541-3 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

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 Abbreviated terms

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 IEC 62541-3). 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 IEC 62541-3) 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 –TypeDefinitionTable

<table>
<tr><th>Attribute</th><th colspan="5">Value</th></tr>
<tr><td>Attribute name</td><td colspan="5">Attribute value. If it is an optional Attribute that is not set "--" will be used.</td></tr>
<tr><td></td><td colspan="5"></td></tr>
<tr><td><b>References</b></td><td><b>NodeClass</b></td><td><b>BrowseName</b></td><td><b>DataType</b></td><td><b>TypeDefinition</b></td><td><b>ModellingRule</b></td></tr>
<tr><td><i>ReferenceType</i> name</td><td><i>NodeClass</i> of the <i>TargetNode</i>.</td><td><i>BrowseName</i> of the target <i>Node</i>. If the <i>Reference</i> is to be instantiated by the server, then the value of the target Node’s BrowseName is "--".</td><td><i>DataType</i> of the referenced <i>Node</i>, only applicable for <i>Variables</i>.</td><td><i>TypeDefinition</i> of the referenced <i>Node</i>, only applicable for <i>Variables</i> and <i>Objects</i>.</td><td>Referenced <i>ModellingRule</i> of the referenced <i>Object</i>.</td></tr>
<tr><td colspan="6">NOTE Notes referencing footnotes of the table content.</td></tr>
</table>

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 document are only symbolic names. IEC 62541-6 defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this document 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 document, 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 document is 0. The namespace for this NamespaceIndex is specified in IEC 62541-3.

Note that this document 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 document is specified in the tables defining the *Nodes*. The NamespaceIndex for all *BrowseNames* defined in this document is 0.

5 Common Attributes

5.1 General

For all *Nodes* specified in this document, 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 IEC 62541-6.
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 document.
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 depends 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 document, 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 document, 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 document, 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 document, 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 document, 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 document 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 document 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 document. An example of extending the *ObjectTypes* is putting the standard *Property NodeVersion* defined in IEC 62541-3 into the *BaseObjectType*, stating that each *Object* of the *Server* will provide a *NodeVersion*.

In addition to the *ObjectTypes* in Clause 6, Annex B provides *ObjectTypes* for StateMachines, Annex C provides *ObjectTypes* Model for File Transfer and Annex F defines *ObjectTypes* for User Authorization.

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**	**ModellingRule**
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[a]	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[a]	-- ServerCapabilitiesType	Mandatory
HasComponent	Object	ServerDiagnostics[a]	-- ServerDiagnosticsType	Mandatory
HasComponent	Object	VendorServerInfo	-- VendorServerInfoType	Mandatory
HasComponent	Object	ServerRedundancy[a]	-- 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

[a] 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*. IEC 62541-4 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 IEC 62541-4. 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 IEC 62541-4.

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. IEC 62541-4 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 IEC 62541-7 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 6.3.13 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 IEC 62541-4.

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 IEC 62541-4.

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 IEC 62541-4.

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 IEC 62541-4.

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 IEC 62541-7 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 IEC 62541-7.

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 IEC 62541-3 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 IEC 62541-3 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 IEC 62541-3 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 the 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* (IEC 62541-4) 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 IEC 62541-4 for the definition of redundancy in this document. 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 IEC 62541-4.

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	MaxNodesPerTranslateBrowsePathsToNodeIds	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 IEC 62541-6.

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[]	PropertyType	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 IEC 62541-4 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 IEC 62541-6. 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 document defines standard *EventTypes*. They are represented in the *AddressSpace* as *ObjectTypes*. The *EventTypes* are already defined in IEC 62541-3. The following subclauses specify their representation in the *AddressSpace*.

6.4.2 BaseEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-4, 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 to 1 000
MEDIUM HIGH	601 to 800
MEDIUM	401 to 600
MEDIUM LOW	201 to 400
LOW	1 to 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 to 1 000)	15	1 000
	14	955
	13	910
	12	865
	11	820
	10	775
	9	730
	8	685
MEDIUM (334 to 666)	7	650
	6	575
	5	500
	4	425
	3	350
LOW (1 to 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 IEC 62541-9.

6.4.3 AuditEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-3.

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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-6 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-6 for details on thumbprints.

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

6.4.9 AuditUrlMismatchEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 37.

Table 37 – AuditCertificateUntrustedEventType definition

<table>
<tr><th colspan="2">Attribute</th><th colspan="4">Value</th></tr>
<tr><td colspan="2">BrowseName</td><td colspan="4">AuditCertificateUntrustedEventType</td></tr>
<tr><td colspan="2">IsAbstract</td><td colspan="4">True</td></tr>
<tr><th>References</th><th>NodeClass</th><th>BrowseName</th><th>DataType</th><th>TypeDefinition</th><th>ModellingRule</th></tr>
<tr><td colspan="6">Subtype of the <i>AuditCertificateEventType</i> defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.</td></tr>
</table>

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 IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 38.

Table 38 – AuditCertificateRevokedEventType definition

<table>
<tr><th colspan="2">Attribute</th><th colspan="4">Value</th></tr>
<tr><td colspan="2">BrowseName</td><td colspan="4">AuditCertificateRevokedEventType</td></tr>
<tr><td colspan="2">IsAbstract</td><td colspan="4">True</td></tr>
<tr><th>References</th><th>NodeClass</th><th>BrowseName</th><th>DataType</th><th>TypeDefinition</th><th>ModellingRule</th></tr>
<tr><td colspan="6">Subtype of the <i>AuditCertificateEventType</i> defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.</td></tr>
</table>

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 IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 39.

Table 39 – AuditCertificateMismatchEventType definition

<table>
<tr><th colspan="2">Attribute</th><th colspan="4">Value</th></tr>
<tr><td colspan="2">BrowseName</td><td colspan="4">AuditCertificateMismatchEventType</td></tr>
<tr><td colspan="2">IsAbstract</td><td colspan="4">True</td></tr>
<tr><th>References</th><th>NodeClass</th><th>BrowseName</th><th>DataType</th><th>TypeDefinition</th><th>ModellingRule</th></tr>
<tr><td colspan="6">Subtype of the <i>AuditCertificateEventType</i> defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.</td></tr>
</table>

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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-11 representing the different possibilities to manipulate historical data.

6.4.27 AuditUpdateMethodEventType

This *EventType* is defined in IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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. IEC 62541-4 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 IEC 62541-4>" 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 IEC 62541-3. 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 IEC 62541-3.

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 IEC 62541-3. This *ObjectType* is used as type for the *DataTypeEncodings*. The use of the *DataTypeEncodingType* with *DataTypeDictionaries* is defined in Annex D. 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 IEC 62541.

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 document 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 IEC 62541-3, 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[a]	BuildInfo	BuildInfoType	Mandatory
HasComponent	Variable	SecondsTillShutdown	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	ShutdownReason	LocalizedText	BaseDataVariableType	Mandatory
[a] 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

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	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 BaseDataVariableType	Mandatory
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

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	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**	**BrowseName**	**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**	**BrowseName**	**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*. Text code defined in IETF RFC 2045, IETF RFC 2046 and IETF RFC 2047 shall be used 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	AudiDataType				
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.

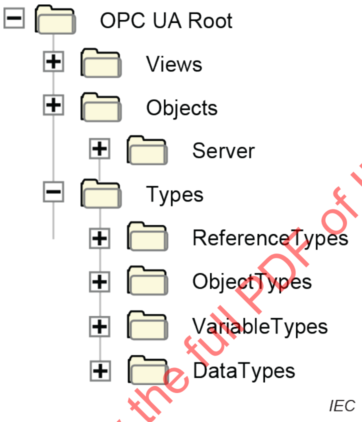


Figure 1 – Standard AddressSpace structure

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".

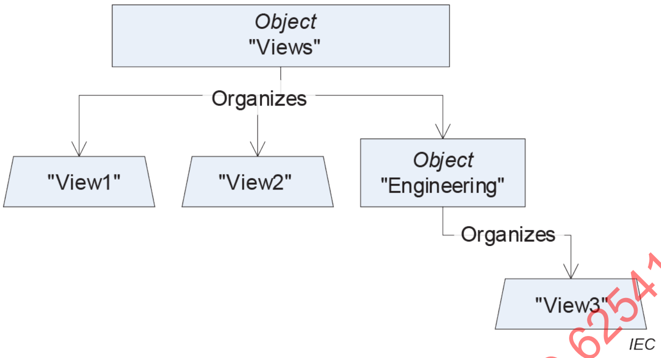


Figure 2 – Views organization

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.

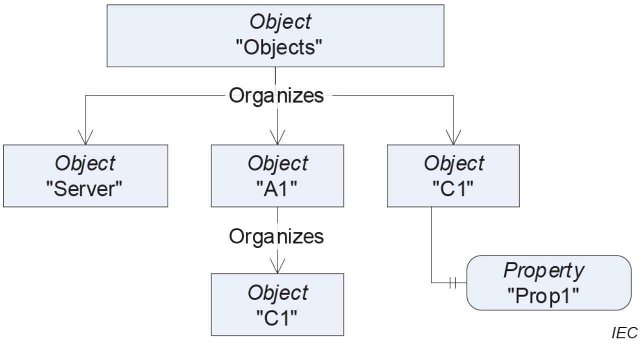


Figure 3 – Objects organization

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 Clause 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*.

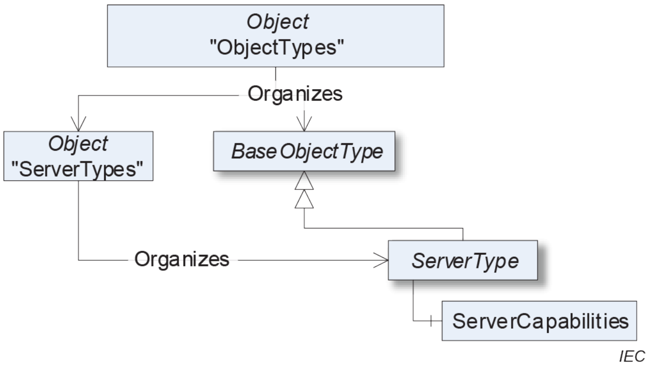


Figure 4 – ObjectTypes organization

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*.

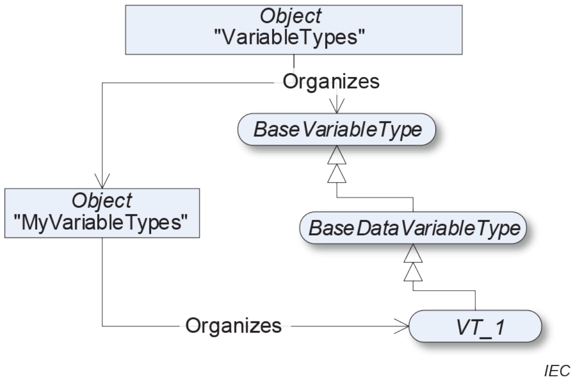


Figure 5 – VariableTypes organization

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*.

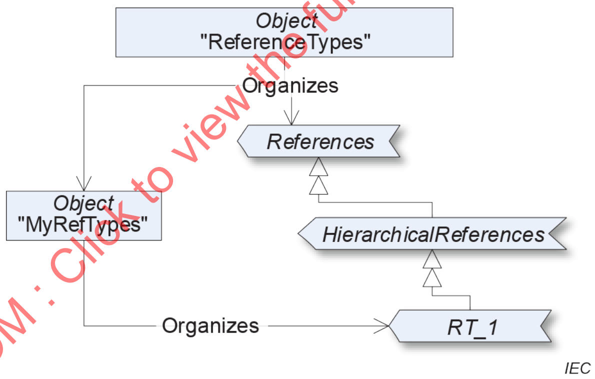


Figure 6 – ReferenceType definitions

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*.

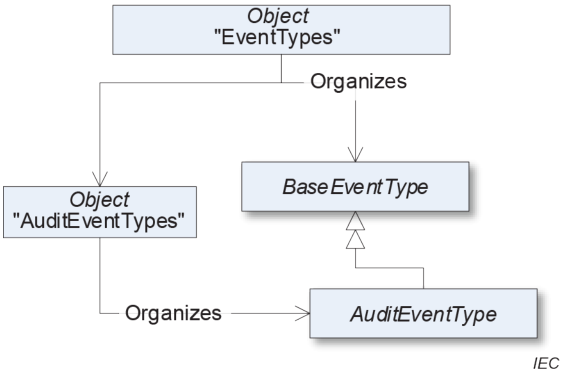


Figure 7 – EventTypes organization

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.

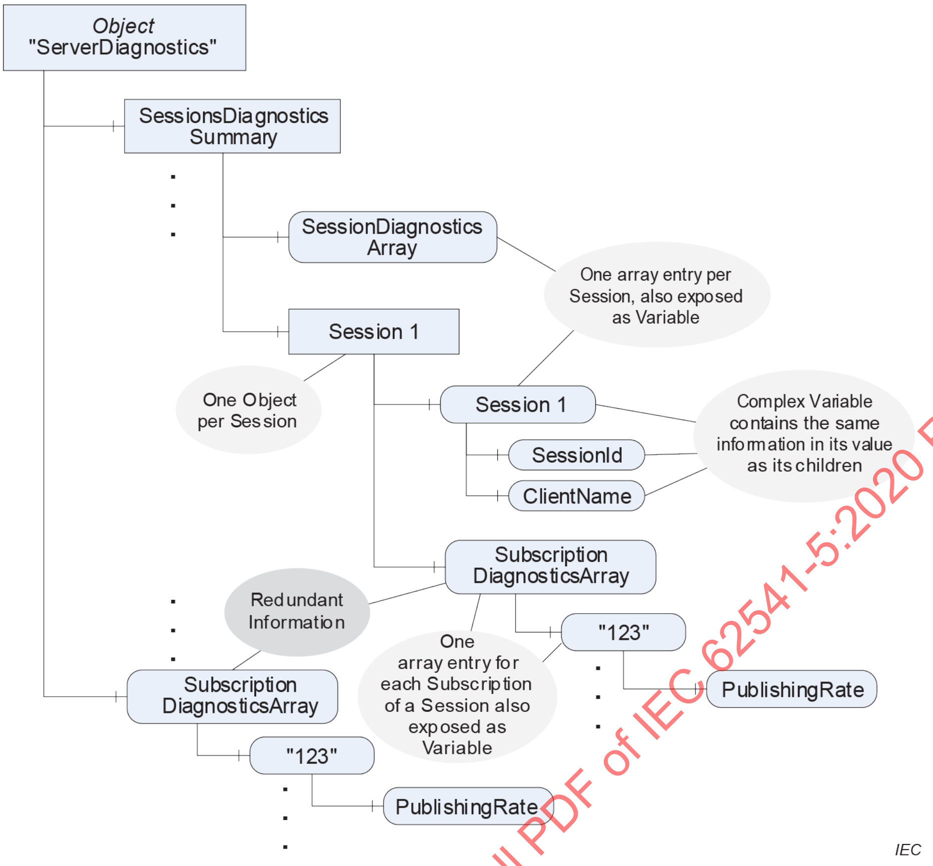

Object "ServerDiagnostics"
SessionsDiagnostics Summary
SessionDiagnostics Array
One array entry per Session, also exposed as Variable
Session 1
One Object per Session
Session 1
Complex Variable contains the same information in its value as its children
SessionId
ClientName
Subscription DiagnosticsArray
Redundant Information
"123"
PublishingRate
One array entry for each Subscription of a Session also exposed as Variable
Subscription DiagnosticsArray
"123"
PublishingRate
IEC

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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-4.

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 IEC 62541-4
Bad_UserAccessDenied	Defined in IEC 62541-4 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 IEC 62541-4.

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 IEC 62541-4
Bad_UserAccessDenied	Defined in IEC 62541-4 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 IEC 62541-4.

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 IEC 62541-4
Bad_InvalidState	Defined in IEC 62541-4 This is returned when a *Subscription* already contains *MonitoredItems*.
Bad_UserAccessDenied	Defined in IEC 62541-4 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3. 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 IEC 62541-3

IEC 62541-3 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 115.

Table 115 – IEC 62541-3 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

BrowseName
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 IEC 62541-4

IEC 62541-4 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 130.

Table 130 – IEC 62541-4 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 IEC 62541-4.

The *AddNodesItem* is a structure that is defined as the type of the nodesToAdd parameter of the AddNodes *Service* in IEC 62541-4.

The *AddReferencesItem* is a structure that is defined as the type of the referencesToAdd parameter of the AddReferences *Service* in IEC 62541-4.

The *DeleteNodesItem* is a structure that is defined as the type of the nodesToDelete parameter of the DeleteNodes *Service* in IEC 62541-4.

The *DeleteReferencesItem* is a structure that is defined as the type of the referencesToDelete parameter of the DeleteReferences *Service* in IEC 62541-4.

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 IEC 62541-4.
WARM_2	Warm means that the server supports warm redundancy as defined in IEC 62541-4.
HOT_3	Hot means that the server supports hot redundancy as defined in IEC 62541-4.
TRANSPARENT_4	Transparent means that the server supports transparent redundancy as defined in IEC 62541-4.
HOT_AND_MIRRORED_5	HotAndMirrored means that the server supports HotAndMirrored redundancy as defined in IEC 62541-4.

See IEC 62541-4 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 shutdown 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 shutdown.
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 IEC 62541-4, 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 IEC 62541-4, 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* shutdown is initated, the state changes to SHUTDOWN_4 and the actual shutdown 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
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.

Name	Type	Description
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 IEC 62541-6.
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>ModelChangeStructureDataType</td><td>structure</td><td></td></tr>
<tr><td>affected</td><td>NodeId</td><td><i>NodeId</i> of the <i>Node</i> that was changed. The client should assume that the <i>affected Node</i> has been created or deleted, had a <i>Reference</i> added or deleted, or the <i>DataType</i> has changed as described by the <i>verb</i>.</td></tr>
<tr><td>affectedType</td><td>NodeId</td><td>If the <i>affected Node</i> was an <i>Object</i> or <i>Variable</i>, <i>affectedType</i> contains the <i>NodeId</i> of the <i>TypeDefinitionNode</i> of the <i>affected Node</i>. 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 <i>verb</i> 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 <i>affected Node</i> has been added.</td></tr>
<tr><td>NodeDeleted</td><td>1</td><td>Indicates the <i>affected Node</i> has been deleted.</td></tr>
<tr><td>ReferenceAdded</td><td>2</td><td>Indicates a <i>Reference</i> has been added. The affected <i>Node</i> may be either a <i>SourceNode</i> or <i>TargetNode</i>. Note that an added bidirectional <i>Reference</i> is reflected by two changes.</td></tr>
<tr><td>ReferenceDeleted</td><td>3</td><td>Indicates a <i>Reference</i> has been deleted. The affected <i>Node</i> may be either a <i>SourceNode</i> or <i>TargetNode</i>. Note that a deleted bidirectional <i>Reference</i> is reflected by two changes.</td></tr>
<tr><td>DataTypeChanged</td><td>4</td><td>This verb may be used only for affected <i>Nodes</i> that are <i>Variables</i> or <i>VariableTypes</i>. It indicates that the <i>DataType Attribute</i> 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 IEC 62541-3 for details).
Note that all <i>verbs</i> shall always be considered in the context where the ModelChangeStructureDataType is used. A NodeDeleted may indicate that a <i>Node</i> was removed from a view but still exists in other <i>Views</i>.</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 IEC 62541-4.
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

Annex A 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*.

Annex A 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 IEC 62541-3.

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

The following clauses describe the design decisions made while modelling the *Server Object*. General *DataTypes*, *VariableTypes* and *ObjectTypes* such as the *EventTypes* described in this document 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 document and the *NodeId* is based on the symbolic name in IEC 62541-6. 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 IEC 62541-3 about the semantic difference between *Properties* and *DataVariables* are not applicable for the information provided about the *Server* the rules described in IEC 62541-3 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 IEC 62541, 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 document. 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 document, 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 document, clients will look there for server-specific information.

Annex B
(normative)

StateMachines

B.1 General

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

Annex B is an integral part of this document, that is, the types defined in Annex B 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 Annex B, 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 Annex B is to provide an appropriate foundation for state machines needed for IEC 62541-9 and IEC 62541-10. 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 Annex B can be extended to support such concepts.

The following clauses describe examples of state machines, define state machines in the context of Annex B 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 Annex B 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".

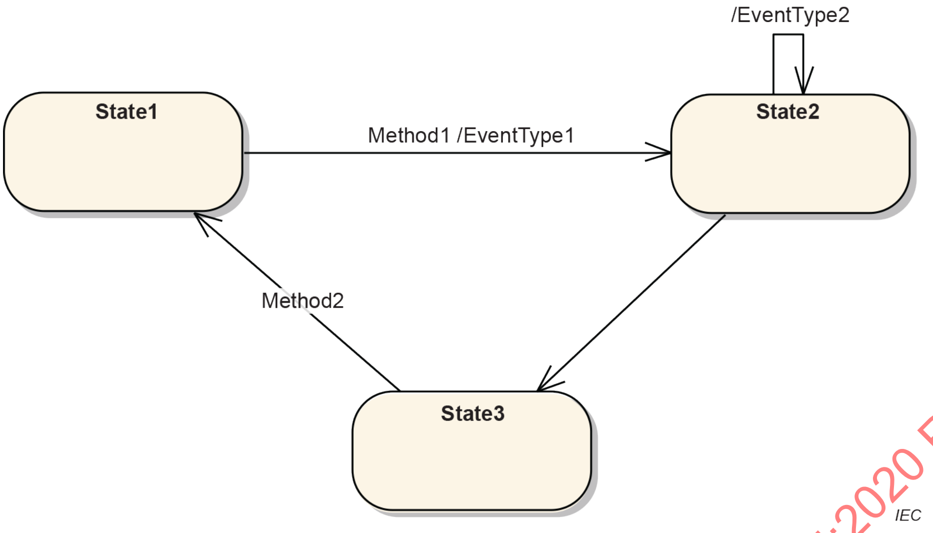


Figure B.1 – Example of a simple state machine

For simplicity, the state machines described in Annex B will only support causes in form of specifying *Methods* that shall 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.

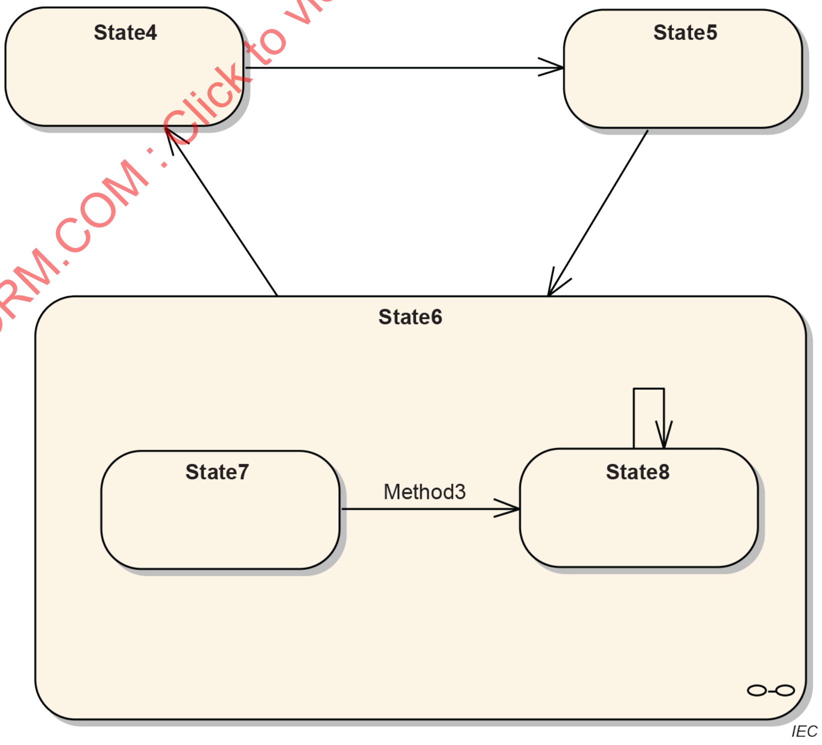


Figure B.2 – Example of a state machine having a sub-machine

B.3 Definition of state machine

The infrastructure of state machines defined in Annex B only deals with the basics of state machines needed to support IEC 62541-9 and IEC 62541-10. The intention is to keep the basic simple but extensible.

For the state machines defined in Annex B 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 Annex B 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 Annex B 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 Annex B 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 Annex B 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 IEC 62541-9 and IEC 62541-10 for additional examples of *StateMachines*.

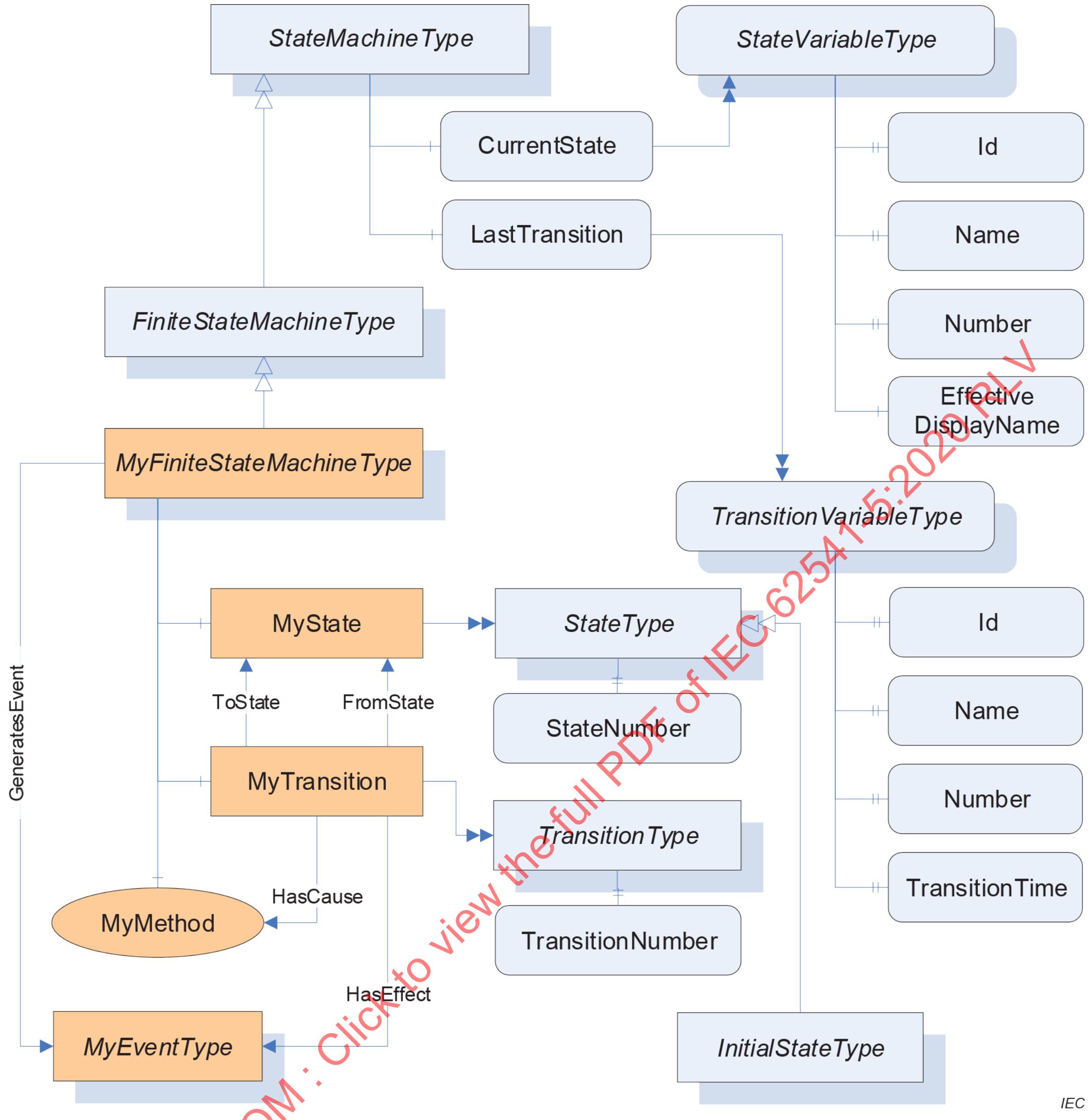


Figure B.3 – The StateMachine Information Model

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 IEC 62541-4. 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.

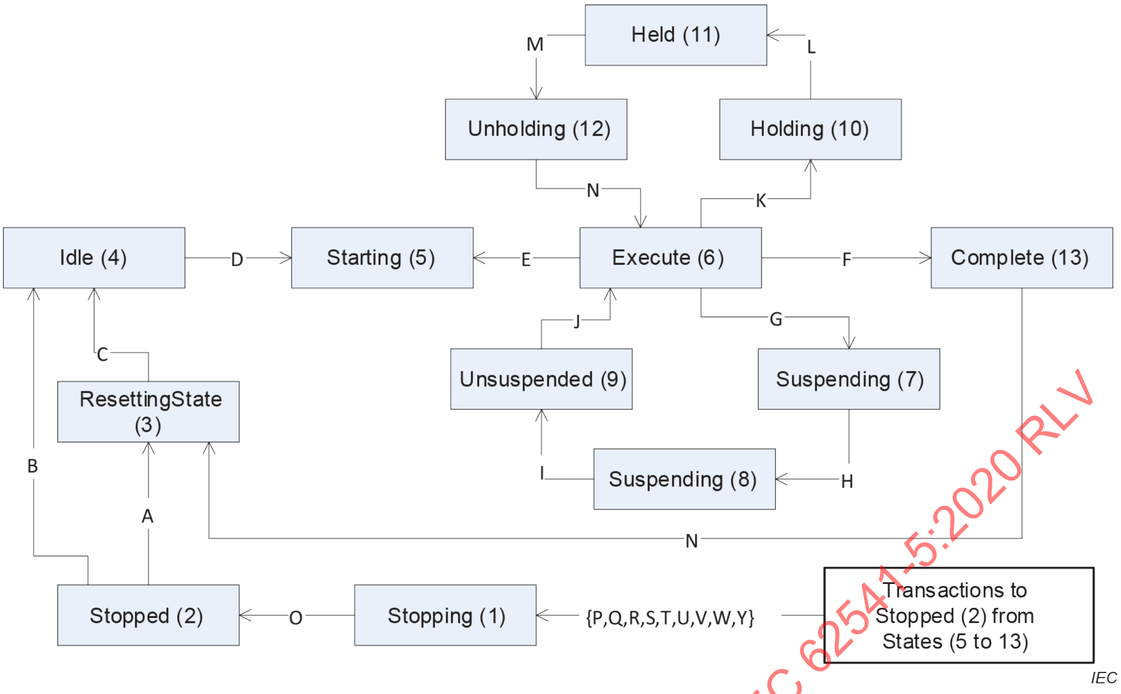


Figure B.4 – Example of a FiniteStateMachine type

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.

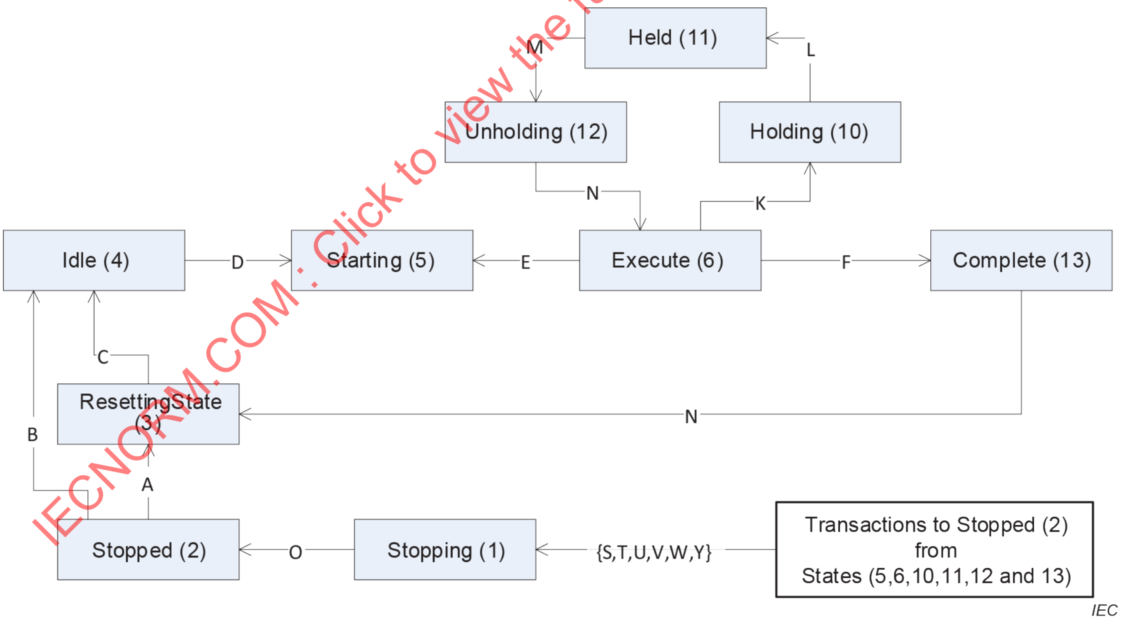


Figure B.5 – Example of a FiniteStateMachine instance

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 in B.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.

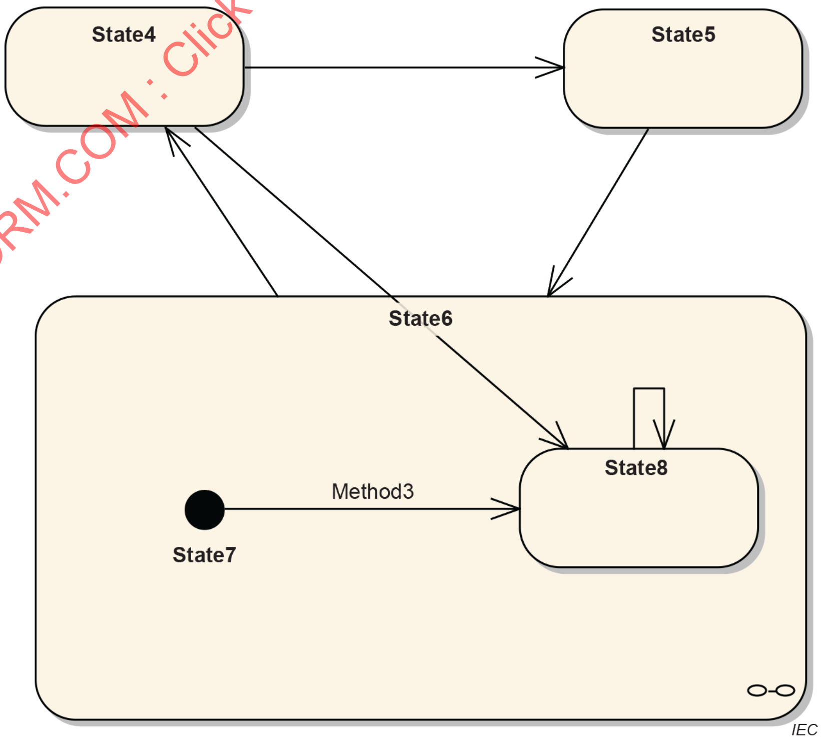


Figure B.6 – Example of an initial State in a sub-machine

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 Annex B 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 Annex B 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* shall be used. If the Transition occurs between a *StateMachine* and a *SubStateMachine*, then the *NodeId* of the *StateMachine* shall 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* shall 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

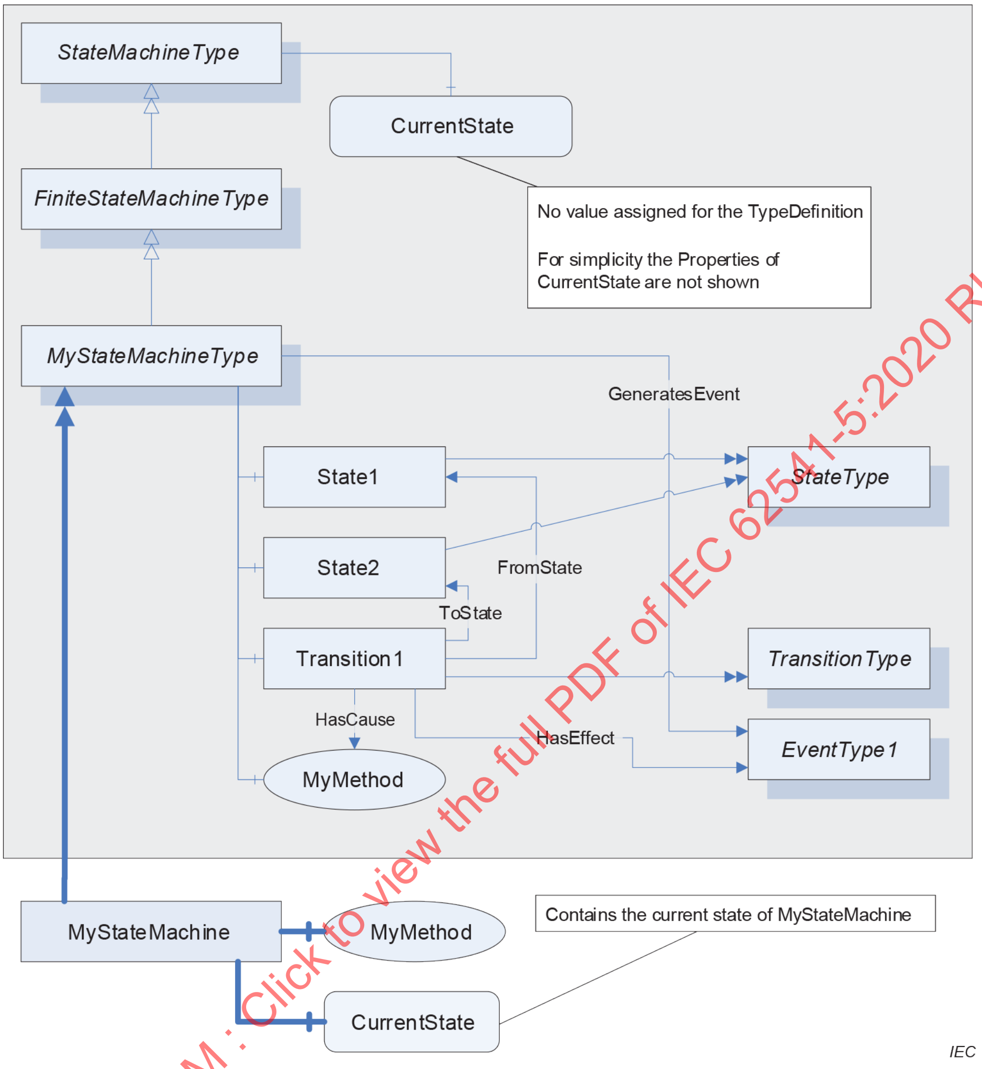


Figure B.7 – Example of a StateMachineType using inheritance

In Figure B.7 an example of a *StateMachine* is given using the Notation defined in IEC 62541-3. 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 SubStateMachine using inheritance

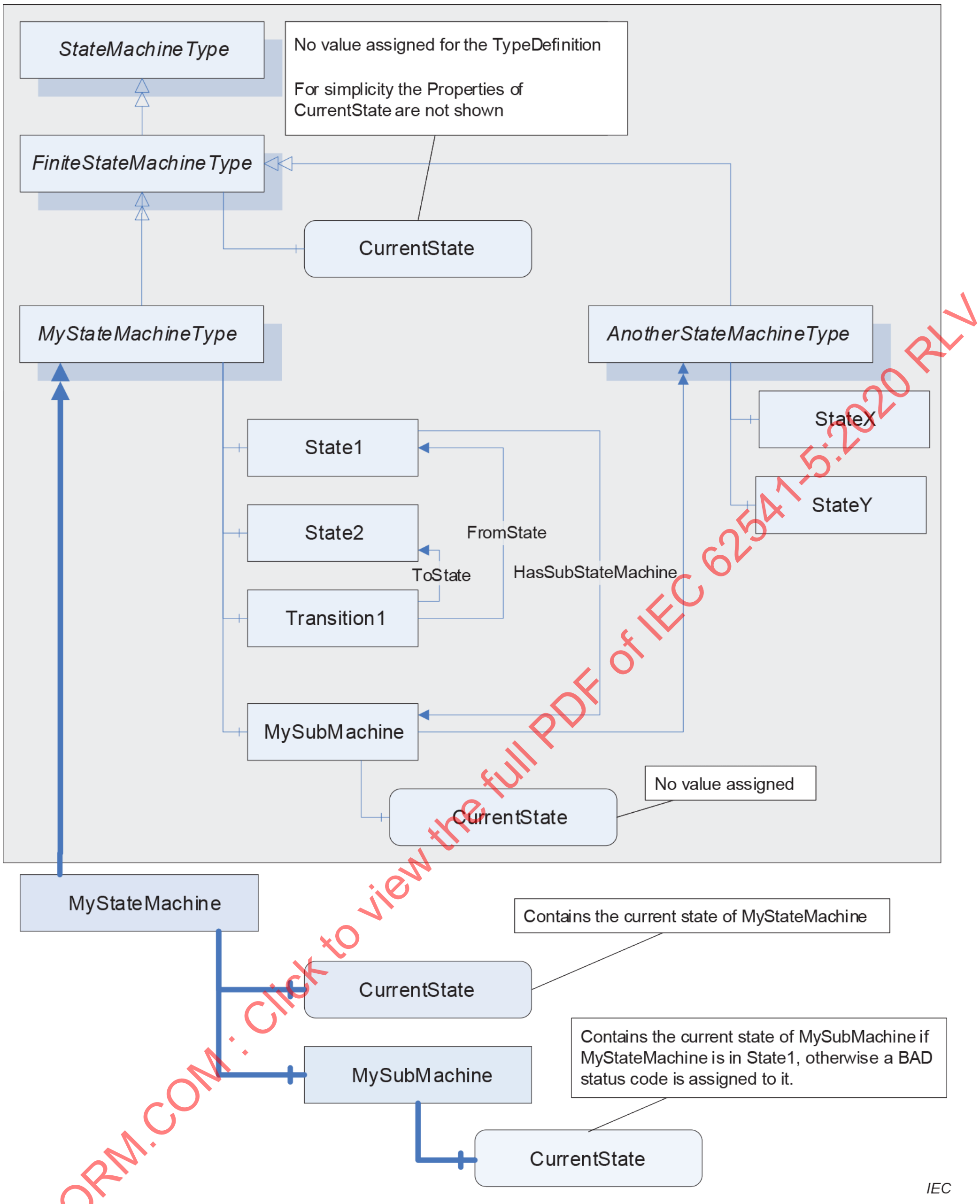

StateMachineType
No value assigned for the TypeDefinition
For simplicity the Properties of CurrentState are not shown
FiniteStateMachineType
CurrentState
MyStateMachineType
AnotherStateMachineType
StateX
StateY
State1
State2
FromState
ToState
HasSubStateMachine
Transition1
MySubMachine
No value assigned
CurrentState
MyStateMachine
Contains the current state of MyStateMachine
CurrentState
MySubMachine
Contains the current state of MySubMachine if MyStateMachine is in State1, otherwise a BAD status code is assigned to it.
CurrentState
IEC

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

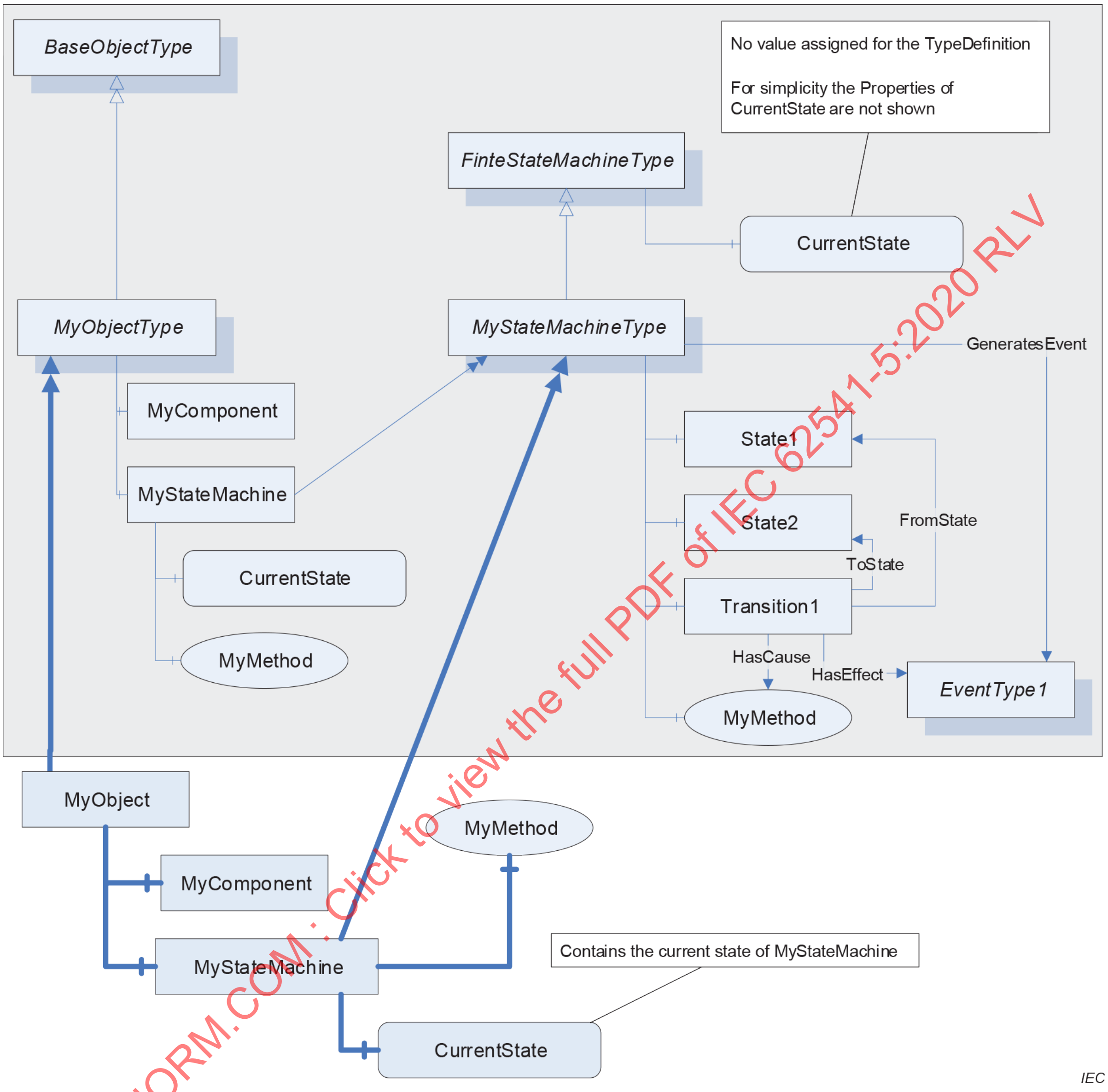


Figure B.9 – Example of a StateMachineType using containment

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.

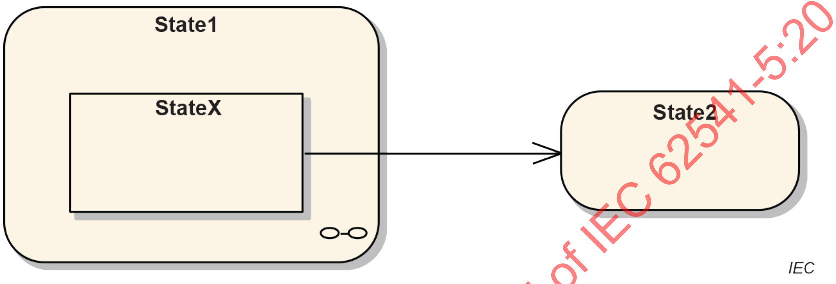


Figure B.10 – Example of a StateMachine with Transitions from sub-states

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.

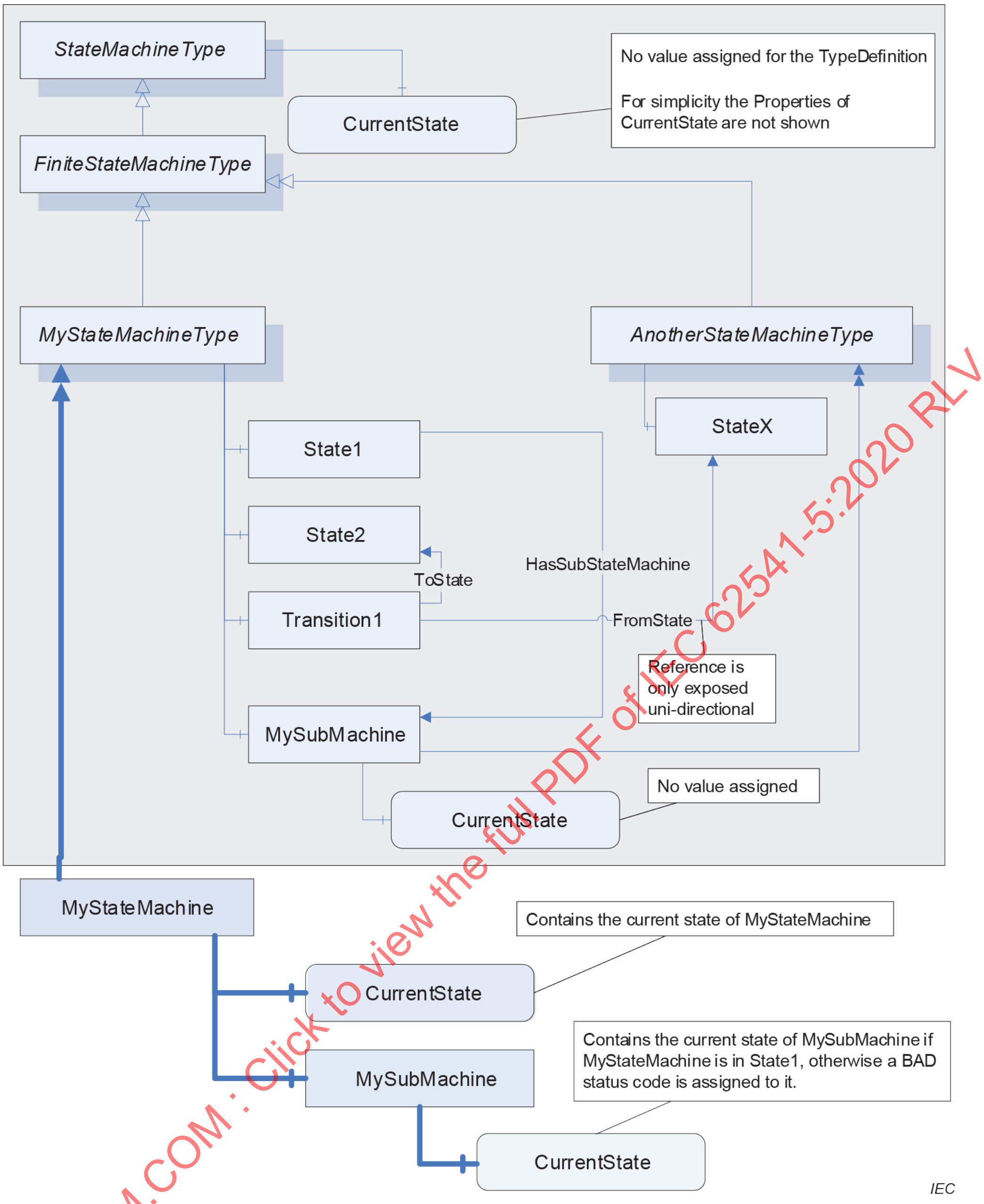

StateMachineType
CurrentState
No value assigned for the TypeDefinition
For simplicity the Properties of CurrentState are not shown
FiniteStateMachineType
MyStateMachineType
AnotherStateMachineType
StateX
State1
State2
ToState
HasSubStateMachine
Transition1
FromState
Reference is only exposed uni-directional
MySubMachine
No value assigned
CurrentState
MyStateMachine
Contains the current state of MyStateMachine
CurrentState
MySubMachine
Contains the current state of MySubMachine if MyStateMachine is in State1, otherwise a BAD status code is assigned to it.
CurrentState
IEC

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

Annex C
(normative)

File Transfer

C.1 Overview

Annex C 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 Annex C 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 Annex C.

Annex C is an integral part of this document, that is, the types defined in Annex C 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

C.2.1 General

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.2.2		Mandatory
HasComponent	Method	Close	Defined in C.2.3		Mandatory
HasComponent	Method	Read	Defined in C.2.4		Mandatory
HasComponent	Method	Write	Defined in C.2.5		Mandatory
HasComponent	Method	GetPosition	Defined in C.2.6		Mandatory
HasComponent	Method	SetPosition	Defined in C.2.7		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 IETF RFC 2046.

Note that all *Methods* on a file require a fileHandle, which is returned in the *Open Method*.

C.2.2 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 <i>mode</i> 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 IEC 62541-4 for a general description. File might be locked and thus not readable.
Bad_NotWritable	See IEC 62541-4 for a general description.
Bad_InvalidState	See IEC 62541-4 for a general description. The file is locked and thus not writable.
Bad_InvalidArgument	See IEC 62541-4 for a general description. Mode setting is invalid.
Bad_NotFound	See IEC 62541-4 for a general description.
Bad_UnexpectedError	See IEC 62541-4 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.2.3 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 IEC 62541-4 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.2.4 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 IEC 62541-4 for a general description. Invalid file handle in call or non-positive length.
Bad_UnexpectedError	See IEC 62541-4 for a general description.
Bad_InvalidState	See IEC 62541-4 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.2.5 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 effect on the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.
Bad_NotWritable	See IEC 62541-4 for a general description. File might be locked and thus not writable.
Bad_InvalidState	See IEC 62541-4 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.2.6 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 IEC 62541-4 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.2.7 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 IEC 62541-4 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**	**ModellingRule**
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.

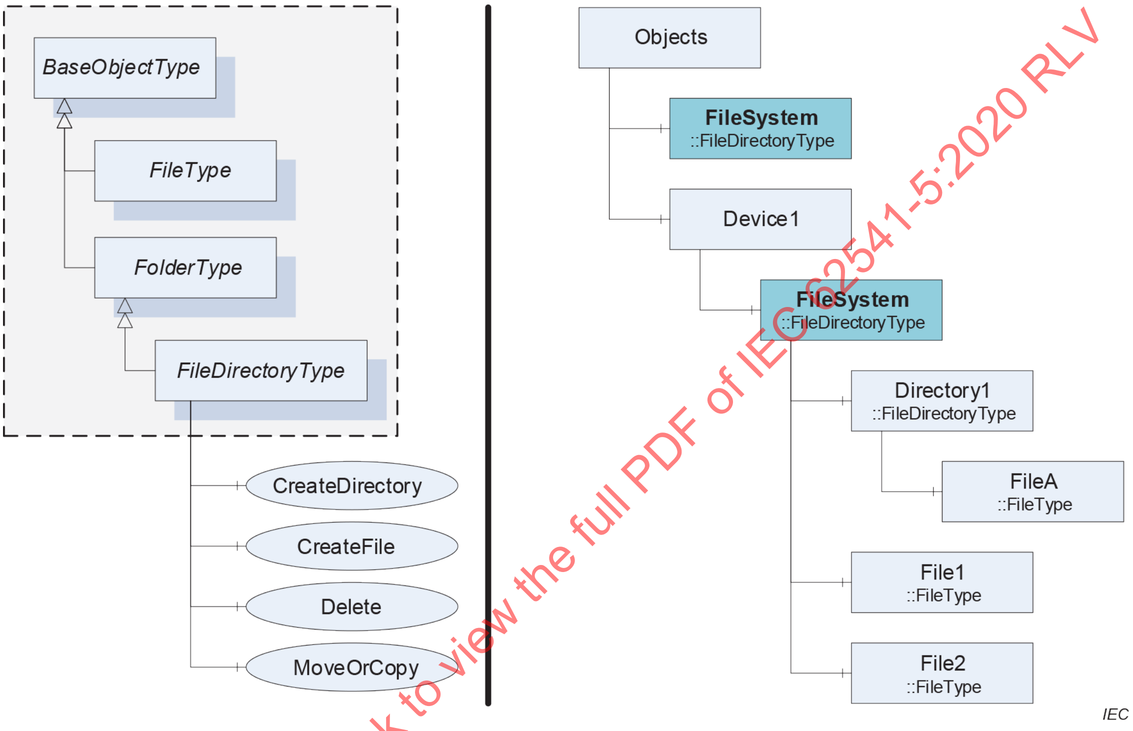


Figure C.1 – FileSystem example

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 IEC 62541-4 for a general description. A directory with the name already exists.
Bad_UserAccessDenied	See IEC 62541-4 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 IEC 62541-4 for a general description. A file with the name already exists.
Bad_UserAccessDenied	See IEC 62541-4 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 IEC 62541-4 for a general description. A file or directory with the provided NodeId is not organized by this object.
Bad_InvalidState	See IEC 62541-4 for a general description. The file or directory is locked and thus cannot be deleted.
Bad_UserAccessDenied	See IEC 62541-4 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 IEC 62541-4 for a general description. A file or directory with the name already exists.
Bad_NotFound	See IEC 62541-4 for a general description. A file or directory with the provided NodeId is not organized by this object.
Bad_InvalidState	See IEC 62541-4 for a general description. The file or directory is locked and thus cannot be moved or copied.
Bad_UserAccessDenied	See IEC 62541-4 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 browseable 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**	**ModellingRule**
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>		FileTransferStateMachineType	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 asynchronously 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 write 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.

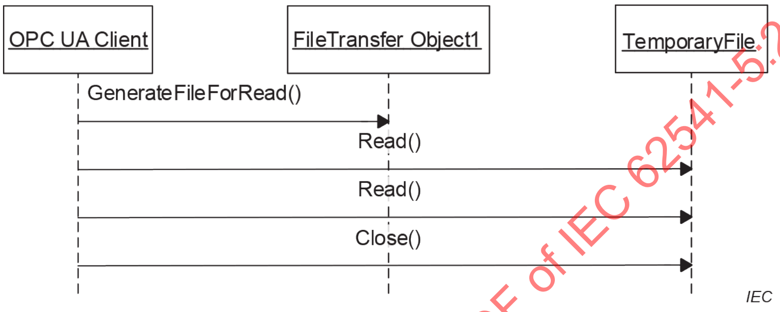


Figure C.2 – Read file transfer example sequence

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.

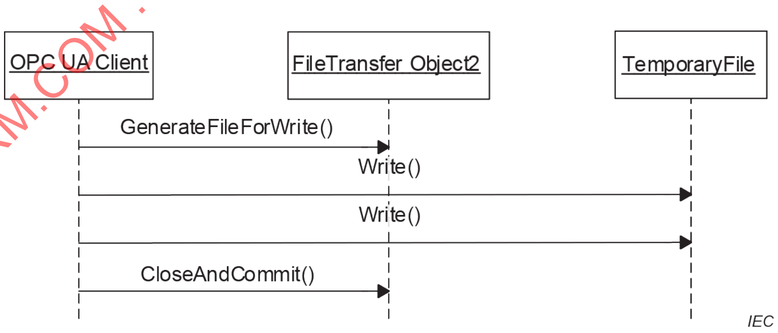


Figure C.3 – Write file transfer example sequence

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 asynchronously, 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 IEC 62541-4 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 IEC 62541-4 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 asynchronously, 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 IEC 62541-4 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.

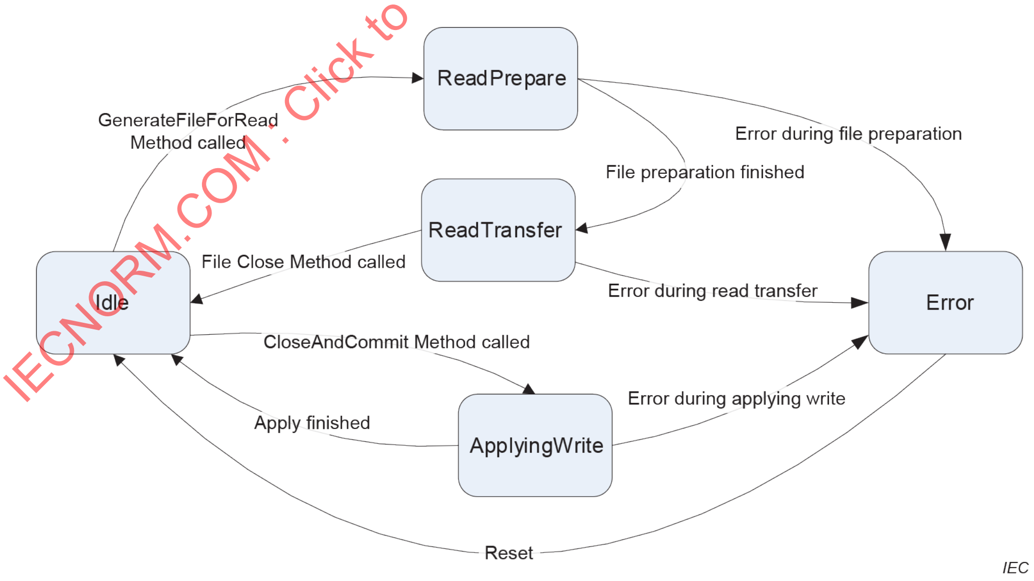

ReadPrepare
GenerateFileForRead
Method called
Error during file preparation
File preparation finished
ReadTransfer
File Close Method called
Idle
Error during read transfer
Error
CloseAndCommit Method called
Error during applying write
Apply finished
ApplyingWrite
Reset
IEC

Figure C.4 – File transfer States

The *FileTransferStateMachineType* and the related type are illustrated in Figure C.5.

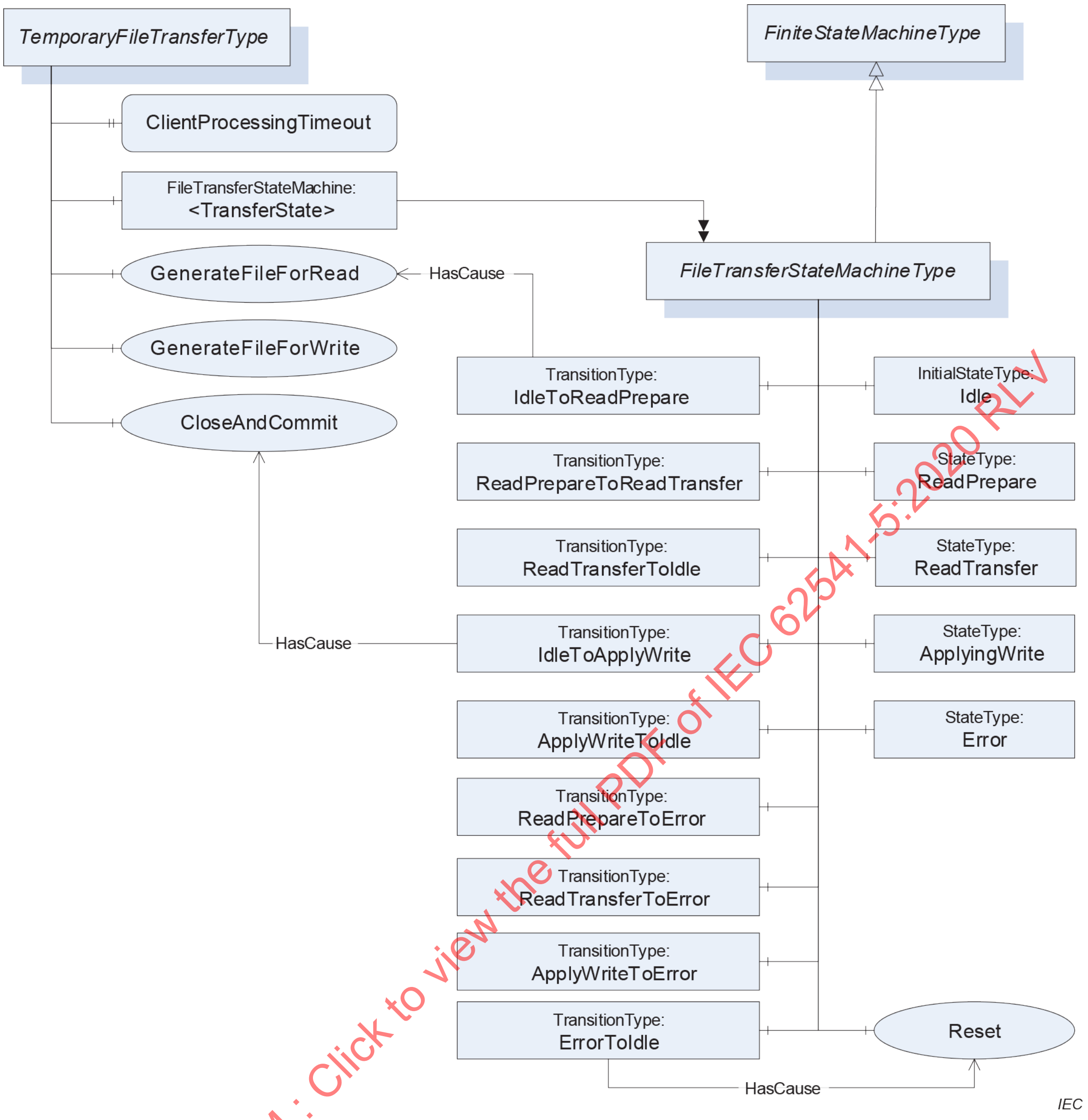


Figure C.5 – FileTransferStateMachineType

This *ObjectType* defines the StateMachine for asynchronously processing of temporary file transfers. It is formally defined in Table C.17. The transitions are formally defined in Table C.18.

Table C.17 – FileTransferStateMachineType

Attribute	Value				
BrowseName	FileTransferStateMachineType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
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

Annex D defines a way to provide encoding information for custom *DataTypes*. In previous releases of the specification this approach was defined in IEC 62541-3. In IEC 62541-3 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

IEC 62541-3 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*.

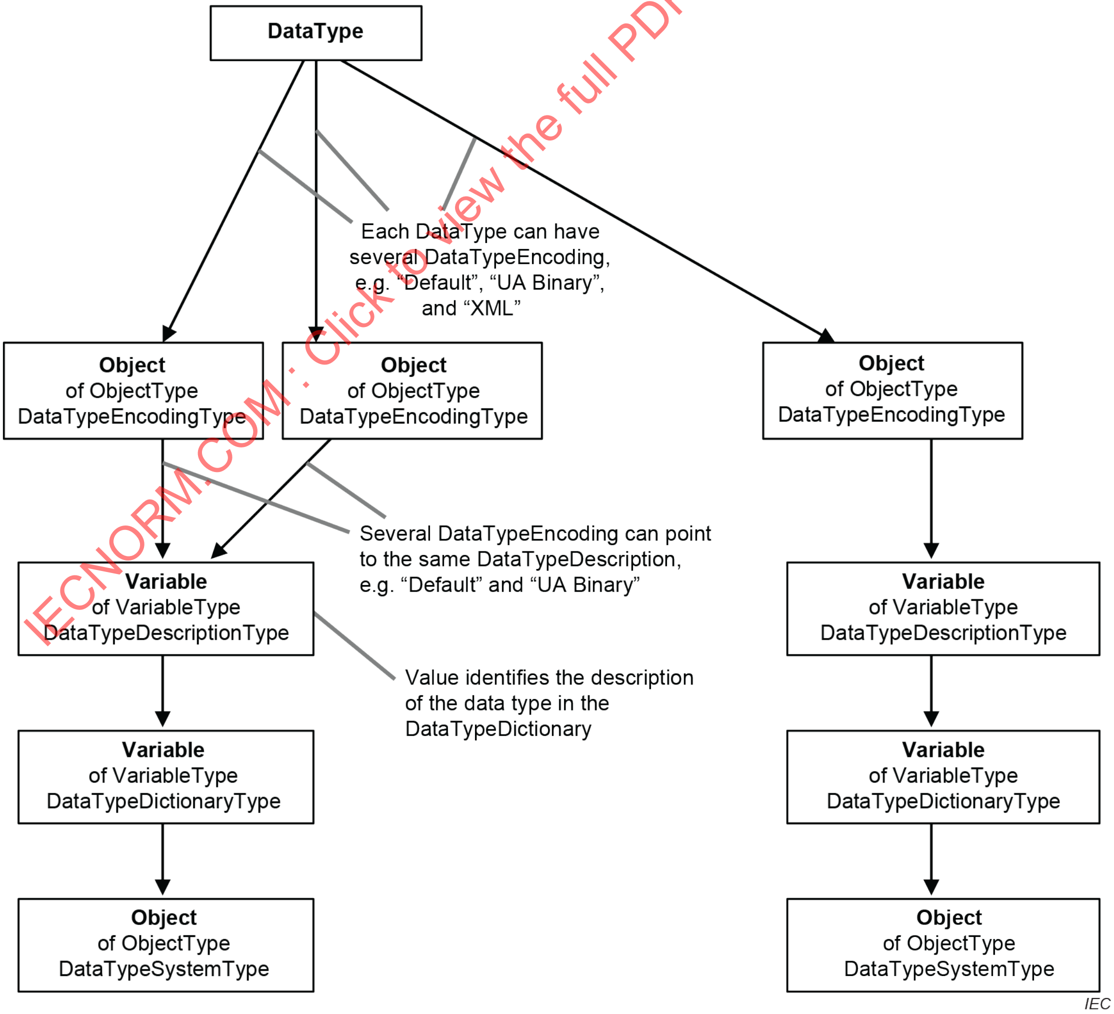

DataType
Each DataType can have several DataTypeEncoding, e.g. “Default”, “UA Binary”, and “XML”
Object of ObjectType DataTypeEncodingType
Object of ObjectType DataTypeEncodingType
Object of ObjectType DataTypeEncodingType
Several DataTypeEncoding can point to the same DataTypeDescription, e.g. “Default” and “UA Binary”
Variable of VariableType DataTypeDescriptionType
Variable of VariableType DataTypeDescriptionType
Value identifies the description of the data type in the DataTypeDictionary
Variable of VariableType DataTypeDictionaryType
Variable of VariableType DataTypeDictionaryType
Object of ObjectType DataTypeSystemType
Object of ObjectType DataTypeSystemType
IEC

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 recognize the *DataTypeSystem* to parse any of the type description information. OPC UA *Clients* that do not recognize 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 of how *DataTypes* are modelled in the *AddressSpace*.

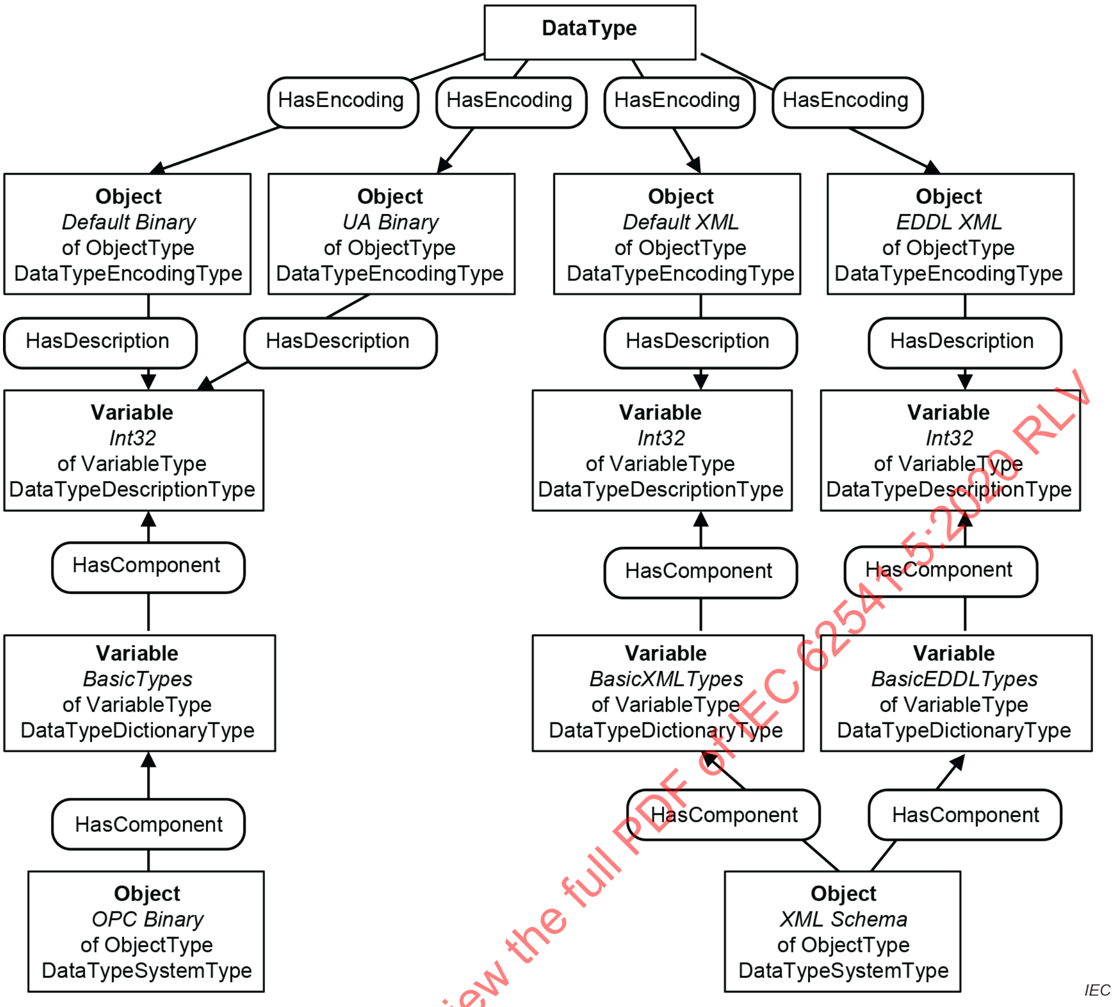


Figure D.2 – Example of DataType modelling

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 document 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 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*.

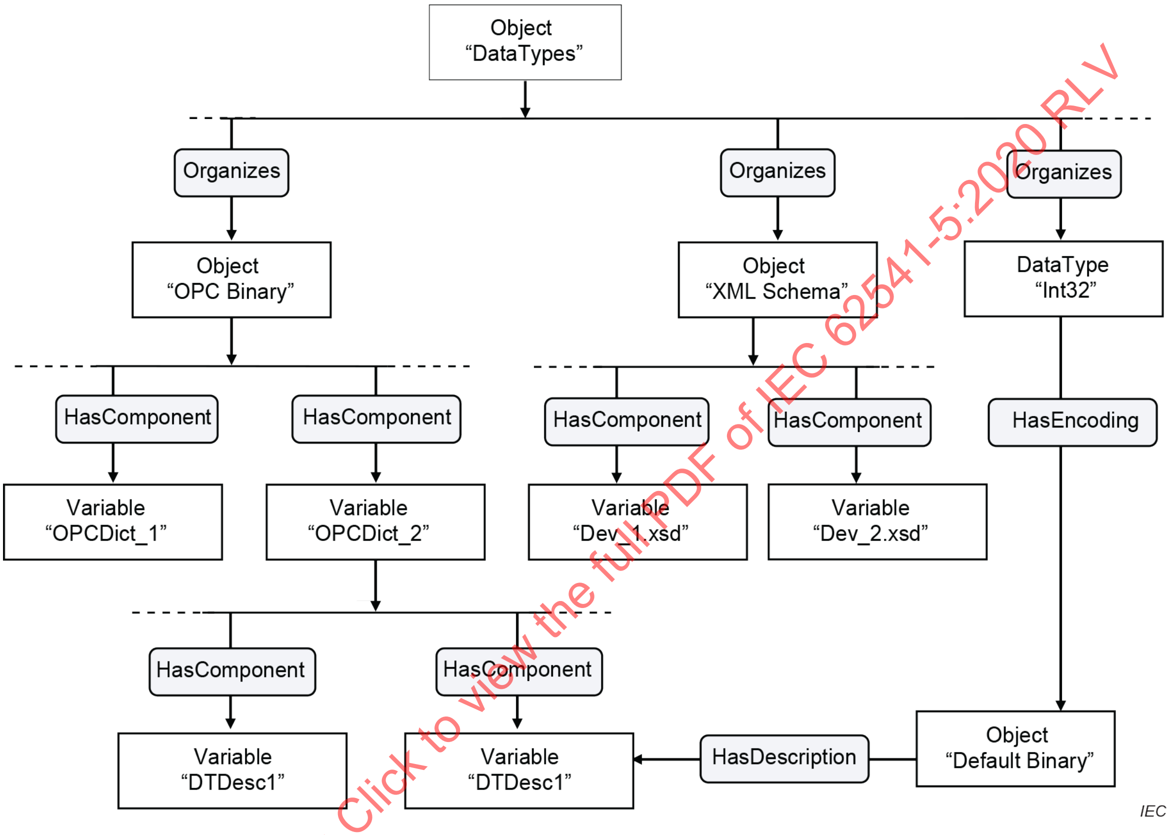


Figure D.3 – DataTypes organization

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 an *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	String				
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 IEC 62541-3. 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 E 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 illustrates the structure of an OPC Binary *TypeDictionary*.

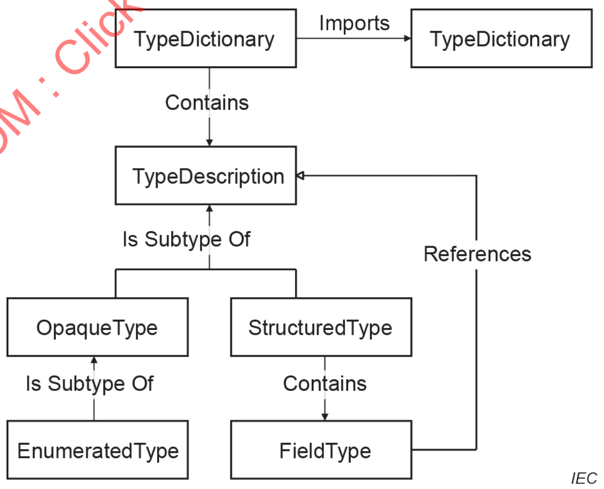


Figure E.1 – OPC Binary Dictionary structure

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 specify 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*.
SwitchValue	xs:unsignedInt	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.

Name	Type	Description
SwitchOperand	xs:string	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.
Terminator	xs:hexBinary	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: Field Data Type Terminator Byte Order Hexadecimal String Char tab character not applicable 09 WideChar tab character BigEndian 0009 WideChar tab character LittleEndian 0900 Int16 1 BigEndian 0001 Int16 1 LittleEndian 0100
anyAttribute	*	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.

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 of possible 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 recognize 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	An 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 nanosecond 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

E.4.1 A 128-bit signed integer

```
<opc:OpaqueType Name="Int128" LengthInBits="128" ByteOrderSignificant="true">
  <opc:Documentation>A 128-bit signed integer.</opc:Documentation>
</opc:OpaqueType>
```

E.4.2 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.

E.4.3 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.

E.4.4 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.

E.4.5 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.

E.4.6 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 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.

E.4.7 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.

E.4.8 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 ISO/IEC/IEEE 60559:2011 single precision floating point
value.</opc:Documentation>
  </opc:OpaqueType>

  <opc:OpaqueType Name="Double" LengthInBits="64" ByteOrderSignificant="true">
    <opc:Documentation>An ISO/IEC/IEEE 60559:2011 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	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
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 is 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.

IEC 62541-3 defines well-known *Roles*. If this *Method* is used to add a well-known *Role*, the name of the *Role* from IEC 62541-3 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 IEC 62541-6.

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	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					
HasProperty	*Variable*	Identities	IdentityMappingRuleType []	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 *EndpointsExclude 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 included 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 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.

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 Identity Mapping Type	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* shall meet for a *Session* to be mapped to the *Role*. The meaning of the criteria depends on the *mappingType*. The criteria are an empty string 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.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

____

ARCHITECTURE UNIFIÉE OPC –

Partie 5: Modèle d'information

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62541-5 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Cette troisième édition annule et remplace la deuxième édition parue en 2015. Cette édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

a) ajout de l'Annexe F concernant l'Authentification des utilisateurs; description du Modèle d'information de rôle qui autorise également la configuration des Rôles;
b) ajout de nouveaux types de données: "Union", "Decimal", "OptionSet", "DateString", "TimeString", "DurationString", "NormalizedString", "DecimalString" et "AudioDataType";

c) ajout d'une méthode afin de demander un changement d'état dans un Serveur;
d) ajout d'une méthode afin de définir un Abonnement en mode persistant;
e) ajout d'une méthode afin de demander le renvoi de données à partir d'un Abonnement;
f) ajout en C.4 d'un concept permettant la création temporaire d'un fichier pour l'écriture ou la lecture sur un serveur;
g) ajout d'un nouveau type de Variable afin de prendre en charge les Listes de sélection;
h) ajout de propriétés facultatives au FiniteStateMachineType afin de présenter les états et transitions actuellement disponibles;
i) ajout d'une propriété UrisVersion au ServerType. Ces informations de version peuvent être utilisées pour l'invocation de services sans session.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/717/FDIS	65E/733/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette Norme internationale.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2.

Tout au long du présent document et des autres parties de la série IEC 62541, certaines conventions documentaires sont utilisées:

Le format *italique* est utilisé pour mettre en évidence un terme défini ou une définition qui apparaît à l'Article 3 dans l'une des parties de la série.

Le format *italique* est également utilisé pour mettre en évidence le nom d'un paramètre d'entrée ou de sortie de service, ou le nom d'une structure ou d'un élément de structure habituellement défini dans les tableaux.

Par ailleurs, les *termes* et les *noms en italique* sont souvent écrits en camel-case (pratique qui consiste à joindre, sans espace, les éléments des mots ou expressions composés, la première lettre de chaque élément étant en majuscule). Par exemple, le terme défini est *AddressSpace* et non Espace d'adressage. Cela permet de mieux comprendre qu'il existe une définition unique pour *AddressSpace*, et non deux définitions distinctes pour Espace et pour Adressage.

Une liste de toutes les parties de la série IEC 62541, publiées sous le titre général *Acrhitecture unifiée OPC*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de ce document ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives au document recherché. A cette date, le document sera

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

ARCHITECTURE UNIFIÉE OPC –

Partie 5: Modèle d'information

1 Domaine d'application

La présente partie de l'IEC 62541 définit le Modèle d'information de l'Architecture unifiée OPC. Le Modèle d'information décrit des *Nœuds* normalisés de l'*AddressSpace* d'un *Serveur*. Ces *Nœuds* sont des types normalisés ainsi que des instances normalisées utilisés pour le diagnostic ou comme des points d'entrée à des *Nœuds* spécifiques au serveur. Ainsi, le Modèle d'information définit l'*AddressSpace* d'un *Serveur* OPC UA vide. Cependant, tous les *Serveurs* ne sont pas supposés fournir la totalité de ces *Nœuds*.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1*, OPC Unified Architecture – Part 1: Overview and Concepts* (disponible en anglais seulement)

IEC 62541-3, *Architecture unifiée OPC – Partie 3:* Modèle d'espace d'adressage

IEC 62541-4, *Architecture unifiée OPC – Partie 4: Services*

IEC 62541-6, *Architecture unifiée OPC – Partie 6: Mappings*

IEC 62541-7, *Architecture unifiée OPC – Partie 7: Profils*

IEC 62541-9, *Architecture unifiée OPC – Partie 9: Alarmes et conditions*

IEC 62541-10, *Architecture unifiée OPC – Partie 10: Programmes*

IEC 62541-11, *Architecture unifiée OPC – Partie 11: Accès à l'historique*

ISO/IEC/IEEE 60559:2011, *Information technology – Microprocessor Systems – Floating-Point arithmetic* (disponible en anglais seulement)

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

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

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

IETF RFC 2047: Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text
http://www.ietf.org/rfc/rfc2047.txt

XML Schema Tome 1: Structures
http://www.w3.org/TR/xmlschema-1/

XML Schema Tome 2: Datatypes
http://www.w3.org/TR/xmlschema-2/

Xpath: XML Path Language
http://www.w3.org/TR/xpath/

IETF RFC 3629: UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc3629.txt

3 Termes, définitions, termes abrégés et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1 et l'IEC 62541-3 ainsi que les suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1.1
ClientUserId
chaîne qui identifie l'utilisateur du client demandant une action

Note 1 à l'article: Le *ClientUserId* est obtenu, directement ou indirectement, de l'*UserIdentityToken* passé par le *Client* dans l'appel de *Service ActivateSession*. Voir 6.4.3 pour plus d'informations.

3.2 Termes abrégés

UA	Unified Architecture (Architecture unifiée)
XML	eXtensible Markup Language (Langage de balisage extensible)

3.3 Conventions pour les descriptions de Nœuds

Les définitions des *Nœuds* sont spécifiées à l'aide de tableaux (voir Tableau 2).

Les *Attributs* sont définis par la fourniture du nom de l'*Attribut* et d'une valeur, ou d'une description de la valeur.

Les *Références* sont définies par la fourniture du nom du *ReferenceType*, du *BrowseName*, du *TargetNode* et de sa *NodeClass*.

- Si le *TargetNode* est une composante du *Nœud* défini dans le tableau, les *Attributs* du *Nœud* composé sont définis dans la même rangée du tableau.
- Le *DataType* est uniquement spécifié pour les *Variables*; "[<number>]" indique une matrice unidimensionnelle, alors que pour les matrices multidimensionnelles, l'expression est répétée pour chaque dimension (par exemple, [2][3] pour une matrice à deux dimensions). Pour toutes les matrices, *ArrayDimensions* est établi comme identifié par des valeurs <number>. Si aucun <number> n'est défini, la dimension correspondante est définie sur 0, indiquant une taille inconnue. Si aucun nombre n'est fourni, *ArrayDimensions* peut être exclu. L'absence de crochets identifie un *DataType* scalaire et le *ValueRank* est défini sur la valeur correspondante (voir IEC 62541-3). De plus, *ArrayDimensions* est défini sur nul ou est exclu. Si elle peut être Any ou ScalarOrOneDimension, la valeur est placée dans "{<value>}" et, ainsi, "{Any}" ou

"{ScalarOrOneDimension}" et le *ValueRank* sont définis sur la valeur correspondante (voir IEC 62541-3) et *ArrayDimensions* est défini sur nul ou est exclu. Des exemples sont donnés dans le Tableau 1.

Tableau 1 – Exemples de DataTypes

Notation	DataType	ValueRank	ArrayDimensions	Description
Int32	Int32	−1	exclu ou nul	Valeur Int32 scalaire.
Int32[]	Int32	1	exclu ou {0}	Matrice unidimensionnelle de Int32 de taille inconnue.
Int32[][]	Int32	2	exclu ou {0,0}	Matrice bidimensionnelle de Int32 de taille inconnue pour les deux dimensions.
Int32[3][]	Int32	2	{3,0}	Matrice bidimensionnelle de Int32 de taille 3 pour la première dimension et de taille inconnue pour la seconde dimension.
Int32[5][3]	Int32	2	{5,3}	Matrice bidimensionnelle de Int32 de taille 5 pour la première dimension et de taille 3 pour la seconde dimension.
Int32{Any}	Int32	−2	exclu ou nul	Int32 dans le cas d'un scalaire ou d'une matrice d'une quelconque dimension.
Int32{ScalarOrOneDimension}	Int32	−3	exclu ou nul	Int32 dans le cas d'une matrice unidimensionnelle ou d'un scalaire.

- La TypeDefinition est spécifiée pour les *Objets* et les *Variables*.
- La colonne TypeDefinition spécifie un nom symbolique pour un *NodeId*, c'est-à-dire les points de *Nœud* spécifiés avec une *Référence HasTypeDefinition* au *Nœud* correspondant.
- La *ModellingRule* de la composante référencée est fournie par la spécification du nom symbolique de la règle dans la colonne *ModellingRule*. Dans l'*AddressSpace*, le *Nœud* doit utiliser une *Référence HasModellingRule* pour pointer vers l'*Objet ModellingRule* correspondant.

Si le *NodeId* d'un *DataType* est fourni, le nom symbolique du *Nœud* représentant le *DataType* doit être utilisé.

Les *Nœuds* de toutes les autres *NodeClasses* ne peuvent pas être définis dans le même tableau; par conséquent, seuls le *ReferenceType* utilisé, sa *NodeClass* et son *BrowseName* sont spécifiés. Une référence à une autre partie de ce document pointe sur leur définition.

Le Tableau 2 représente la table correspondante. Si aucune composante n'est fournie, les colonnes DataType, TypeDefinition et ModellingRule peuvent être exclues et seule la colonne Commentaires est introduite pour pointer vers la définition du *Nœud*.

Tableau 2 – Table de définition des types

Attribut	**Valeur**				
Nom d'attribut	Valeur d'attribut. S'il s'agit d'un attribut facultatif qui n'est pas défini, "--" est utilisé.				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Nom du *ReferenceType*	*NodeClass* du *TargetNode*.	*BrowseName* du *Nœud* cible. Si la *Référence* doit être instanciée par le serveur, la valeur du BrowseName du Nœud cible est alors "--".	Le *DataType* du *Nœud* référencé, applicable seulement aux *Variables*.	La *TypeDefinition* du *Nœud* référencé, applicable seulement aux *Variables* et *Objets*.	*ModellingRule* référencée de l'*Objet* référencé.
NOTE Notes référençant les notes de bas de tableau du contenu du tableau.					

Les composantes de *Nœuds* peuvent être complexes, c'est-à-dire contenir elles-mêmes des composantes. Les *TypeDefinition*, *NodeClass*, *DataType* et *ModellingRule* peuvent résulter des définitions de type, et le nom symbolique peut être créé comme défini en 4.1. Par conséquent, celles qui contiennent des composantes ne sont pas spécifiées de manière explicite; elles sont implicitement spécifiées par les définitions de type.

4 NodeIds et BrowseNames

4.1 NodeIds

Les *NodeIds* de tous les *Nœuds* décrits dans le présent document sont uniquement des noms symboliques. L'IEC 62541-6 définit les *NodeIds* réels.

Le nom symbolique de chaque *Nœud* défini dans le présent document est son *BrowseName*, ou, lorsqu'il fait partie d'un autre *Nœud*, le *BrowseName* de l'autre *Nœud*, un ".", et son propre *BrowseName*. Dans ce cas, "fait partie de" signifie que l'ensemble a une *Référence HasProperty* ou *HasComponent* à cette partie. Sachant que tous les *Nœuds* ne faisant pas partie d'un autre *Nœud* ont un nom unique dans le présent document, le nom symbolique est unique. Par exemple, le *ServerType* défini en 6.3.1 a le nom symbolique "ServerType". Une de ses *InstanceDeclarations* est identifiée comme "ServerType.ServerCapabilities". Sachant que cet *Objet* est complexe, une autre *InstanceDeclaration* du *ServerType* est "ServerType.ServerCapabilities.MinSupportedSampleRate". L'*Objet Serveur* défini en 8.3.2 est fondé sur le ServerType et a le nom symbolique "Serveur". Par conséquent, l'instance fondée sur l'*InstanceDeclaration* décrite ci-dessus a le nom symbolique "Server.ServerCapabilities.MinSupportedSampleRate".

Le NamespaceIndex pour tous les *NodeIds* définis dans le présent document est 0. L'espace de noms pour ce NamespaceIndex est spécifié dans l'IEC 62541-3.

Le présent document définit non seulement des *Nœuds* concrets, mais exige aussi que certains *Nœuds* doivent être créés, par exemple une fois pour chaque Session exécutée sur le *Serveur*. Les *NodeIds* de ces *Nœuds* sont spécifiques au serveur, y compris le Namespace. Toutefois, le NamespaceIndex de ces *Nœuds* ne peut pas être le NamespaceIndex 0, car ils ne sont pas définis par la Fondation OPC, mais créés par le *Serveur*.

4.2 BrowseNames

La partie texte des *BrowseNames* pour tous les *Nœuds* définis dans le présent document est spécifiée dans les tableaux définissant les *Nœuds*. Le NamespaceIndex pour tous les *BrowseNames* définis dans le présent document est 0.

5 Attributs communs

5.1 Généralités

Pour tous les *Nœuds* spécifiés dans le présent document, les *Attributs* énumérés dans le Tableau 3 doivent être définis comme spécifié dans le Tableau 3

Tableau 3 – Attributs de Nœud communs

Attribut	Valeur
DisplayName	Le *DisplayName* est un *LocalizedText*. Chaque serveur doit fournir le *DisplayName* identique au *BrowseName* du *Nœud* pour le LocaleId "en". Indique si le serveur fournit ou non des noms traduits pour d'autres LocaleIds est spécifique au serveur.
Description	Description spécifique au serveur (facultatif).
NodeClass	Doit refléter la *NodeClass* du *Nœud.*
NodeId	Le *NodeId* est décrit par des *BrowseNames* comme défini en 4.1 et défini dans l'IEC 62541-6.
WriteMask	L'*Attribut WriteMask* peut être donné (facultatif). Si l'*Attribut WriteMask* est fourni, il doit définir tous les *Attributs* non spécifiques au serveur comme non inscriptibles. Par exemple, l'*Attribut Description* peut être défini comme inscriptible, car un *Serveur* peut fournir une description spécifique au serveur pour le *Nœud*. Le *NodeId* ne doit pas être inscriptible, car il est défini pour chaque *Nœud* dans le présent document.
UserWriteMask	L'*Attribut UserWriteMask* peut être donné (facultatif). Les mêmes règles que dans le cas de l'*Attribut WriteMask* s'appliquent.
RolePermissions	Des autorisations de rôle spécifiques à un serveur peuvent être données (facultatif).
UserRolePermissions	Des autorisations de rôle pour la Session en cours peuvent être données (facultatif). La valeur est spécifique au serveur et dépend de l'*Attribut RolePermissions* (s'il est fourni) et de la *Session* en cours.
AccessRestrictions	Des restrictions d'accès spécifiques à un serveur peuvent être données (facultatif).

5.2 Objets

Pour tous les *Objets* spécifiés dans le présent document, les *Attributs* énumérés dans le Tableau 4 doivent être définis comme spécifié dans le Tableau 4

Tableau 4 – Attributs d'Objet communs

Attribut	Valeur
EventNotifier	Indique si le *Nœud* peut être utilisé ou non pour s'abonner à des *Evénements* est spécifique au serveur.

5.3 Variables

Pour toutes les *Variables* spécifiées dans le présent document, les *Attributs* énumérés dans le Tableau 5 doivent être définis comme spécifié dans le Tableau 5.

Tableau 5 – Attributs de Variable communs

Attribut	Valeur
MinimumSamplingInterval	Un intervalle d'échantillonnage minimal spécifique au serveur est donné (facultatif).
AccessLevel	Le niveau d'accès pour les *Variables* utilisées dans les définitions de types est spécifique au serveur; pour toutes les autres *Variables* définies dans le présent document, le niveau d'accès doit permettre la lecture; d'autres réglages sont spécifiques au serveur.
UserAccessLevel	La valeur pour l'*Attribut UserAccessLevel* est spécifique au serveur. Il est admis par hypothèse que toutes les *Variables* puissent être accessibles par au moins un utilisateur.
Value	Pour les *Variables* utilisées comme *InstanceDeclarations*, la valeur est spécifique au serveur; autrement, elle doit représenter la valeur décrite dans le texte.
ArrayDimensions	Si le *ValueRank* n'identifie pas une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* ≤ 0), *ArrayDimensions* peut être défini sur nul ou bien l'*Attribut* est absent. Ce comportement est spécifique au serveur. Si le *ValueRank* spécifie une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* > 0), l'*Attribut ArrayDimensions* doit être spécifié dans le tableau définissant la *Variable*.
Historizing	La valeur pour l'*Attribut Historizing* est spécifique au serveur.
AccessLevelEx	Si l'*Attribut AccessLevelEx* est donné, il doit avoir les bits 8, 9 et 10 définis sur 0, ce qui signifie que les opérations de lecture et d'écriture sur une *Variable* individuelle sont atomiques et que les matrices peuvent être en partie écrites.

5.4 VariableTypes

Pour tous les *VariableTypes* spécifiés dans le présent document, les *Attributs* énumérés dans le Tableau 6 doivent être définis comme spécifié dans le Tableau 6.

Tableau 6 – Attributs de VariableType communs

Attributs	Valeur
Value	Une valeur par défaut spécifique au serveur peut être donnée (facultatif).
ArrayDimensions	Si le *ValueRank* n'identifie pas une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* ≤ 0), *ArrayDimensions* peut être défini sur nul ou bien l'*Attribut* est absent. Ce comportement est spécifique au serveur. Si le *ValueRank* spécifie une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* > 0), l'*Attribut ArrayDimensions* doit être spécifié dans le tableau définissant le *VariableType*.

5.5 Méthodes

Pour toutes les *Méthodes* spécifiées dans le présent document, les *Attributs* énumérés dans le Tableau 7 doivent être définis comme spécifié dans le Tableau 7.

Tableau 7 – Attributs de Méthode communs

Attributs	Valeur
Executable	Toutes les *Méthodes* définies dans le présent document doivent être exécutables (*Attribut Executable* défini sur "True"), à moins d'être définies différemment dans la définition de la *Méthode*.
UserExecutable	La valeur de l'*Attribut UserExecutable* est spécifique au serveur. Il est admis par hypothèse que toutes les *Méthodes* puissent être exécutées par au moins un utilisateur.

6 ObjectTypes normalisés

6.1 Généralités

Généralement, les composantes d'un *ObjectType* sont fixes et peuvent être étendues par sous-typage. Cependant, étant donné que chaque *Objet* d'un *ObjectType* peut être étendu avec des composantes supplémentaires, le présent document permet d'étendre les *ObjectTypes* normalisés définis dans ce document avec des composantes supplémentaires. Il est ainsi possible d'exprimer les informations supplémentaires dans la définition de type qui est déjà contenue dans chaque *Objet*. Certains *ObjectTypes* fournissent déjà des points d'entrée pour les extensions spécifiques au serveur. Cependant, il n'est pas admis de limiter les composantes des *ObjectTypes* normalisés définis dans le présent document. Un exemple d'extension des *ObjectTypes* consiste à placer la *Propriété* normalisée *NodeVersion* définie dans l'IEC 62541-3 dans le *BaseObjectType*, énonçant que chaque *Objet* du *Serveur* fournit une *NodeVersion*.

Outre les *ObjectTypes* à l'Article 6, l'Annexe B fournit des *ObjectTypes* pour *StateMachines*, l'Annexe C fournit des *ObjectTypes* pour le transfert de fichiers et l'Annexe F définit des *ObjectTypes* pour les autorisations d'utilisateur.

6.2 BaseObjectType

Le *BaseObjectType* est utilisé comme définition de type chaque fois qu'il existe un *Objet* n'ayant plus de définitions disponibles pour les types concrets. Il convient que les *Serveurs* évitent d'utiliser cet *ObjectType* et utilisent si possible un type plus spécifique. Cet *ObjectType* est l'*ObjectType* de base et tous les autres *ObjectTypes* doivent hériter de lui, directement ou indirectement. Cependant, des *Serveurs* peuvent ne pas fournir toutes les *Références HasSubtype* de cet ObjectType à ses sous-types et il n'est donc pas exigé de fournir cette information.

Pour cet *ObjectType*, il n'est pas spécifié d'autres *Références* que les *Références HasSubtype*. Il est défini de manière formelle dans le Tableau 8.

Tableau 8 – Définition de BaseObjectType

Attribut	Valeur				
BrowseName	BaseObjectType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	ObjectType	ServerType	Défini en 6.3.1		
HasSubtype	ObjectType	ServerCapabilitiesType	Défini en 6.3.2		
HasSubtype	ObjectType	ServerDiagnosticsType	Défini en 6.3.3		
HasSubtype	ObjectType	SessionsDiagnosticsSummaryType	Défini en 6.3.4		
HasSubtype	ObjectType	SessionDiagnosticsObjectType	Défini en 6.3.5		
HasSubtype	ObjectType	VendorServerInfoType	Défini en 6.3.6		
HasSubtype	ObjectType	ServerRedundancyType	Défini en 6.3.7		
HasSubtype	ObjectType	BaseEventType	Défini en 6.4.2		
HasSubtype	ObjectType	ModellingRuleType	Défini en 6.5		
HasSubtype	ObjectType	FolderType	Défini en 6.6		
HasSubtype	ObjectType	DataTypeEncodingType	Défini en 6.7		

6.3 ObjectTypes pour l'Objet Serveur

6.3.1 ServerType

Cet *ObjectType* définit les fonctions prises en charge par le *Serveur* OPC UA. Il est défini de manière formelle dans le Tableau 9.

Tableau 9 – Définition de ServerType

Attribut	**Valeur**			
BrowseName	ServerType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType/TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2				
HasProperty	Variable	ServerArray	String[] PropertyType	Mandatory
HasProperty	Variable	NamespaceArray	String[] PropertyType	Mandatory
HasProperty	Variable	UrisVersion	VersionTime PropertyType	Optional
HasComponent	Variable	ServerStatus[a]	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	Objet	ServerCapabilities[a]	-- ServerCapabilitiesType	Mandatory
HasComponent	Objet	ServerDiagnostics[a]	-- ServerDiagnosticsType	Mandatory
HasComponent	Objet	VendorServerInfo	-- VendorServerInfoType	Mandatory
HasComponent	Objet	ServerRedundancy[a]	-- ServerRedundancyType	Mandatory
HasComponent	Objet	Namespaces	-- NamespacesType	Optional
HasComponent	Méthode	GetMonitoredItems	Défini en 9.1	Optional
HasComponent	Méthode	ResendData	Défini en 9.2	Optional
HasComponent	Méthode	SetSubscriptionDurable	Défini en 9.3	Optional
HasComponent	Méthode	RequestServerStateChange	Défini en 9.4	Optional

[a] Les *Objets* et *Variables* conteneurs de ces *Objets* et *Variables* sont définis par leur *BrowseName* défini dans le *TypeDefinitionNode* correspondant. Le *NodeId* est défini par le nom symbolique composé décrit en 4.1.

ServerArray définit une matrice des URI (Uniform Resource Identifier, identificateur uniforme de ressource) du *Serveur*. Cette *Variable* est également appelée *tableau de serveur*. Chaque URI dans cette matrice représente un nom logique unique au niveau global pour un *Serveur* dans le domaine d'application du réseau dans lequel il est installé. Chaque instance de *Serveur* OPC UA a un seul URI qui est utilisé dans le *tableau de serveur* d'autres *Serveurs* OPC UA. L'indice 0 est réservé à l'URI du *Serveur* local. Les valeurs au-dessus de 0 sont utilisées pour identifier des *Serveurs* distants et sont spécifiques à un *Serveur*. L'IEC 62541-4 décrit un mécanisme de découverte qui peut être utilisé pour résoudre les URI en des URL (Uniform Resource Locator, localisateur uniforme de ressource). L'URI du *Serveur* est sensible à la casse.

L'URI de la *ServerArray* avec l'indice 0 doit être identique à l'URI de la *NamespaceArray* avec l'indice 1, car ils représentent tous deux le *Serveur* local.

Les indices du *tableau de serveur* sont appelés *indices de serveur* ou *noms de serveur*. Les *Services* OPC UA les utilisent pour identifier les *TargetNodes* de *Références* qui résident dans des *Serveurs* distants. Les clients peuvent lire le tableau en entier ou lire individuellement les entrées du tableau. Le *Serveur* ne doit pas modifier ou supprimer des entrées de ce tableau lorsqu'un client a une session ouverte sur le *Serveur*, car les clients peuvent mettre le *tableau de serveur* en cache. Un *Serveur* peut ajouter des entrées au *tableau de serveur* même si des clients sont connectés au *Serveur*.

NamespaceArray définit une matrice des URI d'espace de noms. Cette *Variable* est également appelée *tableau d'espace de noms*. Les indices du *tableau d'espace de noms* sont appelés *NamespaceIndexes*. Les *NamespaceIndexes* sont utilisés dans des *NodeIds* dans des *Services* OPC UA, en lieu et place des URI d'espace de noms plus longs. L'indice 0 est réservé à l'espace de noms OPC UA, et l'indice 1 est réservé au *Serveur* local. Les clients peuvent lire le *tableau d'espace de noms* en entier ou lire individuellement les entrées du *tableau d'espace de noms*. Le *Serveur* ne doit pas modifier ou supprimer des entrées du *tableau d'espace de noms* lorsqu'un client a une session ouverte sur le *Serveur*, car les clients peuvent mettre le *tableau d'espace de noms* en cache. Un *Serveur* peut ajouter des entrées au *tableau d'espace de noms* même si des clients sont connectés au *Serveur*. Il est recommandé aux *Serveurs* de ne pas modifier les indices du *tableau d'espace de noms*, mais d'ajouter seulement des entrées, car le client peut mettre des *NodeIds* en cache en utilisant les indices. Néanmoins, il est parfois impossible aux *Serveurs* d'éviter de modifier des indices du *tableau d'espace de noms*. Il convient, pour les clients mettant en cache des *NamespaceIndexes* de *NodeIds,* de toujours vérifier lors de l'ouverture d'une session que les *NamespaceIndexes* mis en cache n'ont pas changé.

UrisVersion définit la version de la *ServerArray* et de la *NamespaceArray*. A chaque fois que la *ServerArray* ou la *NamespaceArray* est modifiée, la valeur *UrisVersion* doit être mise à jour à une valeur supérieure à la précédente. La *Propriété UrisVersion* est utilisée en combinaison avec le *Service SessionlessInvoke* défini dans l'IEC 62541-4. Si un *Serveur* prend en charge ce *Service*, le *Serveur* doit prendre en charge cette *Propriété*. Il relève de la responsabilité du *Serveur* de fournir un jeu de valeurs cohérent pour la *ServerArray*, la *NamespaceArray* et les *Propriétés UrisVersion*. Le *DataType VersionTime* est défini dans l'IEC 62541-4.

ServerStatus contient des éléments qui décrivent le statut du *Serveur*. Voir 12.10 pour une description de ces éléments.

ServiceLevel décrit la capacité du *Serveur* à fournir ses données au client. La plage de valeurs s'étend de 0 à 255, 0 indiquant le cas le plus défavorable et 255 le cas le plus favorable. L'IEC 62541-4 définit les sous-plages exigées pour différents scénarios. L'objectif est de fournir aux clients une indication de disponibilité parmi des *Serveurs* redondants.

Auditing est un booléen qui spécifie si le *Serveur* génère actuellement des événements d'audit. Il est défini sur TRUE si le *Serveur* génère des événements d'audit, sinon sur FALSE. Les *Profils* définis dans l'IEC 62541-7 spécifient le type d'événements d'audit qui sont générés par le *Serveur*.

EstimatedReturnTime indique le moment auquel le *Serveur* est supposé avoir un *ServerStatus.State* de RUNNING_0. Il convient qu'un *Client* qui observe une interruption ou un *ServiceLevel* de 0 attende avant d'essayer de se reconnecter à ce *Serveur* ou adopte une logique de relance lente. Par exemple, la plupart des *Clients* essaient de se reconnecter immédiatement après un échec et augmentent progressivement le délai entre les tentatives jusqu'à un délai maximal prédéfini. Ce temps peut être utilisé pour programmer un délai dans la logique de reconnexion du *Client*.

LocalTime est une structure contenant les fanions Offset et DaylightSavingInOffset. Le fanion Offset spécifie la différence de temps (en minutes) entre l'heure TUC du *Serveur* et l'heure locale de l'emplacement du *Serveur*. Si DaylightSavingInOffset est configuré sur TRUE, l'heure d'été/normale de l'emplacement du *Serveur* est en vigueur et le fanion Offset inclut la correction de l'heure d'été. S'il est défini sur FALSE, le fanion Offset n'inclut pas la correction de l'heure d'été et celle-ci peut être ou ne pas être en vigueur.

ServerCapabilities définit les fonctions prises en charge par le *Serveur* OPC UA. Voir 6.3.2 pour sa description.

ServerDiagnostics définit les informations de diagnostic relatives au *Serveur* OPC UA. Voir 6.3.3 pour sa description.

VendorServerInfo représente le point d'entrée de navigation pour les informations de *Serveur* définies par le fournisseur. Cet *Objet* doit être présent même s'il n'existe pas d'*Objets* définis par le fournisseur au-dessous. Voir 6.3.6 pour sa description.

ServerRedundancy décrit les fonctions de redondance fournies par le *Serveur*. Cet *Objet* est exigé même si le *Serveur* ne fournit aucun support de redondance. Si le *Serveur* prend en charge la redondance, un sous-type de *ServerRedundancyType* est alors utilisé pour décrire ses fonctions. Autrement, il fournit un *Objet* de type *ServerRedundancyType* avec la *Propriété* RedundancySupport définie sur zéro. Voir 6.3.7 pour la description de *ServerRedundancyType*.

Namespaces fournit une liste d'*Objets* *NamespaceMetadataType* comprenant des informations supplémentaires concernant les espaces de noms utilisés dans le Serveur. Voir 6.3.13 pour la description de *NamespaceMetadataType.*

La *Méthode GetMonitoredItems* sert à identifier les *MonitoredItems* d'un *Abonnement*. Elle est définie en 9.1; l'utilisation prévue est définie dans l'IEC 62541-4.

La *Méthode ResendData* est utilisée pour obtenir les dernières valeurs des éléments surveillés de données d'un *Abonnement*. Elle est définie en 9.2; l'utilisation prévue est définie dans l'IEC 62541-4.

La *Méthode SetSubscriptionDurable* est utilisée pour définir un *Abonnement* dans un mode où les données *MonitoredItem* et les files d'attente d'événements sont stockées et livrées même si un *Client* OPC UA a été déconnecté plus longtemps ou si le *Serveur* OPC UA a été redémarré. Elle est définie en 9.3; l'utilisation prévue est définie dans l'IEC 62541-4.

La *Méthode RequestServerStateChange* permet à un *Client* de demande un changement d'état dans le *Serveur*. Elle est définie en 9.4; l'utilisation prévue est définie dans l'IEC 62541-4.

6.3.2 ServerCapabilitiesType

Cet *ObjectType* définit les fonctions prises en charge par le *Serveur* OPC UA. Il est défini de manière formelle dans le Tableau 10.

Tableau 10 – Définition de ServerCapabilitiesType

Attribut	Valeur			
BrowseName	ServerCapabilitiesType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType/TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini en 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	Objet	OperationLimits	-- OperationLimitsType	Optional
HasComponent	Objet	ModellingRules	-- FolderType	Mandatory
HasComponent	Objet	AggregateFunctions	-- FolderType	Mandatory
HasComponent	Objet	RoleSet	RoleSetType	Optional

ServerProfileArray énumère les *Profils* pris en charge par le *Serveur*. Voir IEC 62541-7 pour les définitions des *Profils* de *Serveur*. Il convient de limiter cette liste aux *Profils* pris en charge par le *Serveur* dans sa configuration actuelle.

LocaleIdArray est une matrice de LocaleIds connus comme étant pris en charge par le *Serveur*. Le *Serveur* peut ne pas connaître tous les LocaleIds qu'il prend en charge, car il peut fournir un accès à des serveurs, systèmes ou appareils sous-jacents qui ne communiquent pas les LocaleIds qu'ils prennent en charge.

MinSupportedSampleRate définit la fréquence minimale d'échantillonnage prise en charge par le *Serveur*, y compris 0.

MaxBrowseContinuationPoints est un entier spécifiant le nombre maximal de points de continuation parallèles du *Service* Browse que le *Serveur* peut prendre en charge par session.

La valeur spécifie le nombre maximal que le *Serveur* peut prendre en charge dans des conditions normales, il n'est donc pas garanti que le *Serveur* puisse toujours prendre en charge le nombre maximal. Il convient que le client n'ouvre pas plus d'appels Browse avec points de continuation ouverts que ne présente cette *Variable*. La valeur 0 indique que le *Serveur* ne limite pas le nombre de points de continuation parallèles qu'il convient que le client utilise.

MaxQueryContinuationPoints est un entier spécifiant le nombre maximal de points de continuation parallèles des *Services* QueryFirst que le *Serveur* peut prendre en charge par session. La valeur spécifie le nombre maximal que le *Serveur* peut prendre en charge dans des conditions normales, il n'est donc pas garanti que le *Serveur* puisse toujours prendre en charge le nombre maximal. Il convient que le client n'ouvre pas plus d'appels QueryFirst avec points de continuation ouverts que ne présente cette *Variable*. La valeur 0 indique que le *Serveur* ne limite pas le nombre de points de continuation parallèles qu'il convient que le client utilise.

MaxHistoryContinuationPoints est un entier spécifiant le nombre maximal de points de continuation parallèles des *Services* HistoryRead que le *Serveur* peut prendre en charge par session. La valeur spécifie le nombre maximal que le *Serveur* peut prendre en charge dans des conditions normales, il n'est donc pas garanti que le *Serveur* puisse toujours prendre en charge le nombre maximal. Il convient que le client n'ouvre pas plus d'appels HistoryRead avec points de continuation ouverts que ne présente cette *Variable*. La valeur 0 indique que le *Serveur* ne limite pas le nombre de points de continuation parallèles qu'il convient que le client utilise.

SoftwareCertificates est une matrice de *SignedSoftwareCertificates* qui contient tous les *SoftwareCertificates* pris en charge par le *Serveur*. Un *SoftwareCertificate* identifie les fonctions du *Serveur.* Il contient la liste des *Profils* pris en charge par le *Serveur*. Les *Profils* sont décrits dans l'IEC 62541-7.

La *Propriété MaxArrayLength* indique la longueur maximale d'une matrice unidimensionnelle ou multidimensionnelle prise en charge par les Variables du *Serveur*. Dans une matrice multidimensionnelle, elle indique la longueur totale. Par exemple, une matrice tridimensionnelle de 2 x 3 x 10 a une longueur de 60. Le *Serveur* peut limiter davantage la longueur pour les Variables individuelles sans en informer le client. Les *Serveurs* peuvent utiliser la *Propriété MaxArrayLength* définie dans l'IEC 62541-3 concernant les *DataVariables* individuelles afin de spécifier la taille propre aux valeurs individuelles. La *Propriété* individuelle peut avoir une valeur supérieure ou inférieure à la *MaxArrayLength*.

La *Propriété MaxStringLength* indique le nombre maximal d'octets dans les *Strings* prises en charge par les *Variables* du *Serveur*. Les *Serveurs* peuvent prendre le pas sur cette configuration en ajoutant la *Propriété MaxStringLength* définie dans l'IEC 62541-3 pour une *DataVariable* individuelle. Si un *Serveur* n'impose pas un nombre maximal d'octets ou n'est pas en mesure de déterminer le nombre maximal d'octets, cette *Propriété* ne doit pas être fournie.

La *Propriété MaxByteStringLength* indique le nombre maximal d'octets dans une *ByteString* prise en charge par les *Variables* du *Serveur*. Elle spécifie également la taille maximale par défaut des tampons de lecture et d'écriture d'un *Objet FileType*. Les *Serveurs* peuvent prendre le pas sur cette configuration en ajoutant la *Propriété MaxByteStringLength* définie dans l'IEC 62541-3 à une *DataVariable* individuelle ou un *Objet FileType*. Si un *Serveur* n'impose pas un nombre maximal d'octets ou n'est pas en mesure de déterminer le nombre maximal d'octets, cette *Propriété* ne doit pas être fournie.

OperationLimits est un point d'entrée pour accéder aux informations sur les limites de fonctionnement du *Serveur*, par exemple la longueur maximale d'une matrice dans un appel de *Service* de lecture.

ModellingRules est un point d'entrée pour parcourir toutes les *ModellingRules* prises en charge par le *Serveur*. Il convient que toutes les *ModellingRules* prises en charge par le *Serveur* puissent être parcourues à partir de cet *Objet*.

AggregateFunctions est un point d'entrée pour parcourir toutes les *AggregateFunctions* prises en charge par le *Serveur*. Il convient que toutes les *AggregateFunctions* prises en charge par le *Serveur* puissent être parcourues à partir de cet *Objet*. Les AggregateFunctions sont des Objets d'*AggregateFunctionType*.

L'*Objet RoleSet* est utilisé pour publier tous les *Rôles* pris en charge par le *Serveur*. Le *RoleSetType* est spécifié en F.2.

Lorsque les fournisseurs présentent leurs propres capacités, il convient qu'ils ajoutent des *Nœuds* supplémentaires à l'instance normalisée de l'*Objet ServerCapabilities*.

6.3.3 ServerDiagnosticsType

Cet *ObjectType* définit les informations de diagnostic relatives au *Serveur* OPC UA. Cet *ObjectType* est défini de manière formelle dans le Tableau 11.

Tableau 11 – Définition de ServerDiagnosticsType

Attribut	**Valeur**			
BrowseName	ServerDiagnosticsType			
IsAbstract	False			
Références	**Node Class**	**BrowseName**	**DataType/TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2				
HasComponent	Variable	ServerDiagnosticsSummary	ServerDiagnosticsSummaryDataType ServerDiagnosticsSummaryType	Mandatory
HasComponent	Variable	SamplingIntervalDiagnosticsArray	SamplingIntervalDiagnosticsDataType[] SamplingIntervalDiagnosticsArrayType	Optional
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Mandatory
HasComponent	Objet	SessionsDiagnosticsSummary	-- SessionsDiagnosticsSummaryType	Mandatory
HasProperty	Variable	EnabledFlag	Boolean PropertyType	Mandatory

ServerDiagnosticsSummary contient des informations récapitulatives de diagnostic pour le *Serveur*, comme défini en 12.9.

SamplingIntervalDiagnosticsArray est une matrice d'information de diagnostic par fréquence d'échantillonnage comme défini en 12.8. Il existe une entrée pour chaque fréquence d'échantillonnage actuellement utilisée par le *Serveur*. Son *TypeDefinitionNode* est le *VariableType SamplingIntervalDiagnosticsArrayType* qui fournit une *Variable* pour chaque entrée de la matrice, comme défini en 7.9.

Les diagnostics d'intervalles d'échantillonnage sont uniquement collectés par les *Serveurs* qui utilisent un jeu fixe d'intervalles d'échantillonnage. Dans ces cas, la longueur de la matrice et le jeu de *Variables* contenues sont déterminés par la configuration du *Serveur*. Le *NodeId* affecté à une variable donnée de diagnostic d'intervalles d'échantillonnage ne doit pas varier tant que la configuration du *Serveur* ne varie pas. Un *Serveur* peut ne pas présenter la SamplingIntervalDiagnosticsArray s'il n'utilise pas de fréquences d'échantillonnage fixes.

SubscriptionDiagnosticsArray est une matrice d'information de diagnostic d'abonnement pour chaque abonnement, comme défini en 12.15. Il existe une entrée pour chaque canal de Notification effectivement établi dans le *Serveur*. Son *TypeDefinitionNode* est le *VariableType* SubscriptionDiagnosticsArrayType qui fournit une *Variable* pour chaque entrée de la matrice, comme défini en 7.11. Ces *Variables* sont aussi utilisées comme *Variables* référencées par d'autres *Variables*.

SessionsDiagnosticsSummary contient des informations de diagnostic par session, comme défini en 6.3.4.

EnabledFlag identifie si les informations de diagnostic sont recueillies ou non par le *Serveur*. Il peut également être utilisé par un client pour activer ou désactiver la collecte d'informations de diagnostic du *Serveur*. Les réglages suivants de la valeur booléenne s'appliquent: TRUE indique que le *Serveur* recueille des informations de diagnostic; le fait de définir la valeur sur TRUE conduit à la réinitialisation et à l'activation de la collecte. FALSE indique qu'aucune information de diagnostic n'est recueillie; le fait de définir la valeur sur FALSE désactive la collecte sans réinitialiser les valeurs diagnostiques.

Lorsque les diagnostics sont désactivés, le *Serveur* peut renvoyer Bad_NodeIdUnknown pour tous les *Nœuds* de diagnostic statique, à l'exception de la *Propriété EnabledFlag*. Les *Nœuds* de diagnostic dynamique (tels que les *Nœuds Session*) n'apparaissent pas dans l'*AddressSpace*.

Si la collecte d'informations de diagnostic n'est pas du tout prise en charge, la Propriété EnabledFlag est en lecture seule.

6.3.4 SessionsDiagnosticsSummaryType

Cet *ObjectType* définit les informations de diagnostic relatives aux sessions du *Serveur* OPC UA. Cet *ObjectType* est défini de manière formelle dans le Tableau 12.

Tableau 12 – Définition de SessionsDiagnosticsSummaryType

Attribut	**Valeur**			
BrowseName	SessionsDiagnosticsSummaryType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType/TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2				
HasComponent	Variable	SessionDiagnosticsArray	SessionDiagnosticsDataType[] SessionDiagnosticsArrayType	Mandatory
HasComponent	Variable	SessionSecurityDiagnosticsArray	SessionSecurityDiagnosticsDataType[] SessionSecurityDiagnosticsArrayType	Mandatory
HasComponent	Objet	<ClientName>	-- SessionDiagnosticsObjectType	Optional Placeholder
NOTE Cette rangée indique l'absence de *Nœud* dans l'*AddressSpace*. Il s'agit d'un paramètre fictif signalant que les instances d'*ObjectType* ont ces *Objets*.				

SessionDiagnosticsArray fournit une matrice avec une entrée pour chaque session dans le *Serveur* ayant des informations générales de diagnostic relatives à une session.

SessionSecurityDiagnosticsArray fournit une matrice avec une entrée pour chaque session active dans le *Serveur* ayant des informations de diagnostic relatives à la sécurité d'une session. Ces informations étant relatives à la sécurité, il convient de ne pas les rendre accessibles à tous les utilisateurs, mais seulement aux utilisateurs autorisés.

Pour chaque session du *Serveur*, cet *Objet* fournit également un *Objet* représentant la session, indiqué par <*ClientName*>. Le BrowseName peut résulter du *sessionName* défini dans le *Service CreateSession* (IEC 62541-4) ou d'autres mécanismes spécifiques au serveur. Il est issu de l'*ObjectType* SessionDiagnosticsObjectType, comme défini en 6.3.5.

6.3.5 SessionDiagnosticsObjectType

Cet *ObjectType* définit les informations de diagnostic relatives à une session du *Serveur* OPC UA. Cet *ObjectType* est défini de manière formelle dans le Tableau 13.

Tableau 13 – Définition de SessionDiagnosticsObjectType

Attribut	**Valeur**			
BrowseName	SessionDiagnosticsObjectType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType/TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	Mandatory
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	Mandatory
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Mandatory

SessionDiagnostics contient les informations générales de diagnostic relatives à la session; la *Variable SessionSecurityDiagnostics* contient les informations de diagnostic relatives à la sécurité. Etant donné que les informations de la seconde *Variable* concernent la sécurité, il convient qu'elles ne soient pas accessibles à tous les utilisateurs, mais seulement aux utilisateurs autorisés.

SubscriptionDiagnosticsArray est une matrice d'information de diagnostic d'Abonnement pour chaque abonnement ouvert, comme défini en 12.15. Son *TypeDefinitionNode* est le *VariableType* SubscriptionDiagnosticsArrayType qui fournit une *Variable* pour chaque entrée de la matrice, comme défini en 7.11.

6.3.6 VendorServerInfoType

Cet *ObjectType* définit un paramètre fictif *Objet* pour les informations spécifiques au fournisseur concernant le *Serveur* OPC UA. Cet *ObjectType* définit un *ObjectType* vide qui ne comporte pas de composantes. Il doit être sous-typé par les fournisseurs pour définir les informations qui leur sont spécifiques. Cet *ObjectType* est défini de manière formelle dans le Tableau 14.

Tableau 14 – Définition de VendorServerInfoType

Attribut	**Valeur**				
BrowseName	VendorServerInfoType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini en 6.2					

6.3.7 ServerRedundancyType

Cet *ObjectType* définit les fonctions de redondance prises en charge par le *Serveur* OPC UA. Il est défini de manière formelle dans le Tableau 15.

Tableau 15 – Définition de ServerRedundancyType

Attribut	**Valeur**				
BrowseName	ServerRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**Type Definition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2					
HasProperty	Variable	RedundancySupport	RedundancySupport	PropertyType	Mandatory
HasSubtype	ObjectType	TransparentRedundancyType	Défini en 6.3.8		
HasSubtype	ObjectType	NonTransparentRedundancyType	Défini en 6.3.9		

RedundancySupport indique la redondance qui est prise en charge par le *Serveur*. Ses valeurs sont définies en 12.5. Elle doit être définie sur NONE_0 pour toutes les instances du *ServerRedundancyType* utilisant directement l'*ObjectType* (pas de sous-type).

6.3.8 TransparentRedundancyType

Cet *ObjectType* est un sous-type de *ServerRedundancyType* et sert à identifier les fonctions du *Serveur* OPC UA pour la redondance commandée par serveur avec une commutation transparente pour le client. Il est défini de manière formelle dans le Tableau 16.

Tableau 16 – Définition de TransparentRedundancyType

Attribut	**Valeur**				
BrowseName	TransparentRedundancyType				
IsAbstract	False				
Références	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *ServerRedundancyType* défini en 6.3.7, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	CurrentServerId	String	PropertyType	Mandatory
HasProperty	Variable	RedundantServerArray	RedundantServerDataType []	PropertyType	Mandatory

RedundancySupport est hérité du *ServerRedundancyType*. Il doit être défini sur TRANSPARENT_4 pour toutes les instances du *TransparentRedundancyType.*

Bien que, dans un scénario de commutation transparente, tous les *Serveurs* redondants soient vus sous le même URI par le *Client*, il peut être exigé de suivre la source exacte de données sur le *Client*. Par conséquent, *CurrentServerId* contient un identificateur du *Serveur* actuellement utilisé dans l'*Ensemble Redondant*. Ce *Serveur* est valide uniquement au sein d'une *Session*; si un *Client* ouvre plusieurs *Sessions*, des *Serveurs* différents parmi les *Serveurs* de l'ensemble redondant peuvent le desservir dans des *Sessions* différentes. La valeur du *CurrentServerId* peut varier en raison d'un *Basculement* ou d'un équilibrage de charge et, par conséquent, un *Client* pour lequel il est nécessaire de suivre la source de données doit s'abonner à cette *Variable*.

En tant qu'information de diagnostic, *RedundantServerArray* contient une matrice de *Serveurs* disponibles dans l'*Ensemble Redondant*, y compris leurs niveaux de service (voir 12.7). Cette matrice peut varier au cours d'une *Session*.

6.3.9 NonTransparentRedundancyType

Cet *ObjectType* est un sous-type de *ServerRedundancyType* et sert à identifier les fonctions du *Serveur* OPC UA pour la redondance non transparente. Il est défini de manière formelle dans le Tableau 17.

Tableau 17 – Définition de NonTransparentRedundancyType

Attribut	**Valeur**				
BrowseName	NonTransparentRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du ServerRedundancyType défini en 6.3.7, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	ServerUriArray	String[]	PropertyType	Mandatory
HasSubtype	ObjectType	NonTransparentNetworkRedundancyType	Défini en 6.3.10		

ServerUriArray est une matrice avec l'URI de tous les *Serveurs* redondants du *Serveur* OPC UA. Voir IEC 62541-4 pour la définition de la redondance dans le présent document. Dans un environnement de redondance non transparente, le *Client* a la responsabilité de s'abonner aux *Serveurs* redondants. Par conséquent, il peut ouvrir une session sur un ou plusieurs *Serveurs* redondants de cette matrice. La *ServerUriArray* doit contenir le *Serveur* local.

RedundancySupport est hérité du *ServerRedundancyType*. Elle doit être définie sur COLD_1, WARM_2, HOT_3 ou HOT_AND_MIRRORED_5 pour toutes les instances du *NonTransparentRedundancyType*. Elle définit la prise en charge d'une redondance fournie par le *Serveur*. Son utilisation prévue est définie dans l'IEC 62541-4.

6.3.10 NonTransparentNetworkRedundancyType

Cet *ObjectType* est un sous-type du *NonTransparentRedundancyType* et sert à identifier les fonctions du *Serveur* OPC UA pour la redondance de réseaux non transparente. Il est défini de manière formelle dans le Tableau 18.

Tableau 18 – Définition de NonTransparentNetworkRedundancyType

Attribut	Valeur				
BrowseName	NonTransparentNetworkRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *NonTransparentRedundancyType* défini en 6.3.9, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	ServerNetworkGroups	NetworkGroupDataType[]	PropertyType	Mandatory

Les *Clients* qui effectuent des commutations entre des chemins de réseaux reliés au même *Serveur* ont le même comportement que la redondance HotAndMirrored. Les redondances du *Serveur* et du réseau peuvent être combinées. Dans l'approche combinée, il est important que le *Client* sache quels ServerUris appartiennent au même *Serveur* représentant différents chemins de réseaux et quels ServerUris représentent différents Serveurs. Par conséquent, un *Serveur* qui met en œuvre une redondance de réseaux non transparente doit utiliser le *NonTransparentNetworkRedundancyType* pour identifier sa prise en charge d'une redondance.

RedundancySupport est hérité du *ServerRedundancyType*. Elle doit être définie sur COLD_1, WARM_2, HOT_3 ou HOT_AND_MIRRORED_5 pour toutes les instances du *NonTransparentNetworkRedundancyType*. Lorsqu'aucune redondance du *Serveur* n'est prise en charge (la *ServerUriArray* ne contient qu'une entrée), la *RedundancySupport* doit être définie sur HOT_AND_MIRRORED_5.

ServerNetworkGroups contient une matrice de *NetworkGroupDataType*. Les URI des *Serveurs* dans cette matrice (dans le *ServerUri* de la structure) doivent être exactement identiques à ceux fournis dans la *ServerUriArray*. L'ordre peut toutefois être différent. Ainsi, la matrice représente une liste des *Serveurs* redondants HotAndMirrored. Si un *Serveur* prend uniquement en charge la redondance de réseaux, il ne comporte qu'une seule entrée dans les *ServerNetworkGroups*. Les *networkPaths* dans la structure représentent les chemins de réseaux redondants pour chacun des *Serveurs*. Les *networkPaths* décrivent les différents chemins (une entrée pour chaque chemin) ordonnés par priorité. Chaque chemin de réseau contient une *endpointUrlList* comportant une matrice de Strings, chacune contenant une URL d'un *Point d'extrémité.* Cela permet d'utiliser différentes options de protocole pour le même chemin de réseau.

Les *Points d'extrémité* fournis doivent correspondre aux *Points d'extrémité* fournis par le Service *GetEndpoints* du *Serveur* correspondant.

6.3.11 OperationLimitsType

Cet *ObjectType* est un sous-type de *FolderType* et sert à identifier les limites de fonctionnement du *Serveur* OPC UA. Il est défini de manière formelle dans le Tableau 19.

Tableau 19 – Définition de OperationLimitsType

Attribut	Valeur				
BrowseName	OperationLimitsType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *FolderType* défini en 6.6, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
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	MaxNodesPerTranslateBrowsePathsToNodeIds	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerNodeManagement	UInt32	PropertyType	Optional
HasProperty	Variable	MaxMonitoredItemsPerCall	UInt32	PropertyType	Optional

Toute *Propriété* des limites opérationnelles fournie doit avoir une valeur autre que zéro.

La *Propriété MaxNodesPerRead* indique la taille maximale de la matrice nodesToRead lorsqu'un *Client* appelle le *Service* Read.

La *Propriété MaxNodesPerHistoryReadData* indique la taille maximale de la matrice nodesToRead lorsqu'un *Client* appelle le *Service* HistoryRead en utilisant les historyReadDetails RAW, PROCESSED, MODIFIED ou ATTIME.

La *Propriété MaxNodesPerHistoryReadEvents* indique la taille maximale de la matrice nodesToRead lorsqu'un *Client* appelle le *Service* HistoryRead en utilisant les historyReadDetails EVENTS.

La *Propriété MaxNodesPerWrite* indique la taille maximale de la matrice nodesToWrite lorsqu'un *Client* appelle le *Service* Write.

La *Propriété MaxNodesPerHistoryUpdateData* indique la taille maximale de la matrice historyUpdateDetails prise en charge par le *Serveur* lorsqu'un *Client* appelle le *Service* HistoryUpdate.

La *Propriété MaxNodesPerHistoryUpdateEvents* indique la taille maximale de la matrice historyUpdateDetails lorsqu'un *Client* appelle le *Service* HistoryUpdate.

La *Propriété MaxNodesPerMethodCall* indique la taille maximale de la matrice methodsToCall lorsqu'un *Client* appelle le *Service* Call.

La *Propriété MaxNodesPerBrowse* indique la taille maximale de la matrice nodesToBrowse lors de l'appel du *Service* Browse ou de la matrice continuationPoints lorsqu'un *Client* appelle le *Service* BrowseNext.

La *Propriété MaxNodesPerRegisterNodes* indique la taille maximale de la matrice nodesToRegister lorsqu'un *Client* appelle le *Service* RegisterNodes et la taille maximale de la matrice nodesToUnregister lors de l'appel du *Service* UnregisterNodes.

La *Propriété MaxNodesPerTranslateBrowsePathsToNodeIds* indique la taille maximale de la matrice browsePaths lorsqu'un *Client* appelle le *Service* TranslateBrowsePathsToNodeIds.

La *Propriété MaxNodesPerNodeManagement* indique la taille maximale de la matrice nodesToAdd lorsqu'un *Client* appelle le *Service* AddNodes, la taille maximale de la matrice referencesToAdd lorsqu'un *Client* appelle le *Service* AddReferences, la taille maximale de la matrice nodesToDelete lorsqu'un *Client* appelle le *Service* DeleteNodes, et la taille maximale de la matrice referencesToDelete lorsqu'un *Client* appelle le *Service* DeleteReferences.

La *Propriété MaxMonitoredItemsPerCall* indique:

- la taille maximale de la matrice itemsToCreate lorsqu'un *Client* appelle le *Service* CreateMonitoredItems;
- la taille maximale de la matrice itemsToModify lorsqu'un *Client* appelle le *Service* ModifyMonitoredItems;
- la taille maximale de la matrice monitoredItemIds lorsqu'un *Client* appelle le *Service* SetMonitoringMode ou le *Service* DeleteMonitoredItems;
- la taille maximale de la somme des matrices linksToAdd et linksToRemove lorsqu'un *Client* appelle le *Service* SetTriggering.

6.3.12 AddressSpaceFileType

Cet *ObjectType* définit le fichier propre à un espace de noms fourni par le *Serveur* OPC UA. Il est défini de manière formelle dans le Tableau 20. Il représente un fichier d'espace d'adressage XML utilisant le schéma XML défini dans l'IEC 62541-6.

Tableau 20 – Définition d'AddressSpaceFileType

Attribut	**Valeur**				
BrowseName	AddressSpaceFileType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du FileType défini en C.2					
HasComponent	Méthode	ExportNamespace	La méthode ne comporte pas de paramètres.		Optional

La *Méthode ExportNamespace* fournit un moyen d'exportation de l'espace de noms de l'*AddressSpace* du *Serveur* au fichier XML représenté par l'*AddressSpaceFileType*. Les *Attributs Value* sont exportés uniquement s'ils représentent des informations de configuration statiques. Le client est supposé appeler en premier lieu la *Méthode ExportNamespace* afin de mettre à jour le fichier XML, puis accéder au fichier avec les *Méthodes* définies dans le *FileType*.

Les *Serveurs* peuvent fournir certains mécanismes spécifiques au fournisseur, qui importent des parties d'un espace d'adressage comme sous-type de cet *ObjectType* (en définissant par exemple des *Méthodes* appropriées).

6.3.13 NamespaceMetadataType

Cet *ObjectType* définit les métadonnées applicables à un espace de noms fourni par le *Serveur*. Il est défini de manière formelle dans le Tableau 21.

Les instances de cet *Objet* permettent que les *Serveurs* fournissent davantage d'informations telles que des informations concernant la version outre l'URI de l'espace de noms. Des informations importantes pour les *Serveurs* d'agrégation sont fournies par les *Propriétés StaticNodeIdTypes, StaticNumericNodeIdRange et StaticStringNodeIdPattern*.

Tableau 21 – Définition de NamespaceMetadataType

Attribut	**Valeur**				
BrowseName	NamespaceMetadataType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 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	Objet	NamespaceFile	-	AddressSpaceFileType	Optional
HasProperty	Variable	DefaultRolePermissions	RolePermissionType[]	PropertyType	Optional
HasProperty	Variable	DefaultUserRolePermissions	RolePermissionType[]	PropertyType	Optional
HasProperty	Variable	DefaultAccessRestrictions	Uint16	PropertyType	Optional

Le *BrowseName* des instances de ce type doit résulter de l'espace de noms représenté. Cela peut, par exemple, être réalisé au moyen de l'indice de l'espace de noms dans la *NamespaceArray* en tant que *namespaceIndex* du *QualifiedName* et de l'URI de l'espace de noms en tant que *name* du QualifiedName.

La *Propriété NamespaceUri* contient l'espace de noms représenté par une instance du MetaDataType.

La *Propriété NamespaceVersion* fournit les informations de version applicables à l'espace de noms. Elle est destinée à des fins d'affichage et ne doit pas être utilisée pour identifier de manière programmatique la dernière version. S'il n'existe pas de version formelle définie pour l'espace de noms, la *String* de cette *Propriété* doit être définie sur nulle.

La *Propriété NamespacePublicationDate* fournit la date de publication de la version de l'espace de noms. Cette valeur de *Propriété* peut être utilisée par les *Clients* pour déterminer la dernière version lorsque différentes versions sont fournies par différents *Serveurs*. S'il n'existe pas de date de publication formelle définie pour l'espace de noms, la *DateTime* de cette *Propriété* doit être définie sur nulle.

La *Propriété IsNamespaceSubset* définit si tous les *Nœuds* de l'espace de noms sont accessibles dans le *Serveur* ou uniquement un sous-ensemble. Elle est définie sur FALSE si l'espace de noms complet est fourni et sur TRUE dans le cas contraire. Si l'exhaustivité est inconnue, cette *Propriété* doit alors être définie sur TRUE.

Les *Nœuds* statiques sont identiques pour tous les *Attributs* dans tous les *Serveurs*, y compris l'*Attribut Value*. Pour les *TypeDefinitionNodes*, les *InstanceDeclarations* doivent également être identiques. Cela signifie que la sémantique est toujours la même pour les

Nœuds statiques. Les espaces de noms avec des *Nœuds* statiques sont, par exemple, des espaces de noms définis par les organismes de normalisation tels que la Fondation OPC. Cela constitue des informations importantes pour les *Serveurs* d'agrégation. Lorsque l'espace de noms est dynamique et utilisé dans plusieurs *Serveurs*, il est nécessaire que le *Serveur* d'agrégation différencie l'espace de noms pour chaque *Serveur* agrégé. Il est nécessaire de traiter les *Nœuds* statiques d'un espace de noms une fois uniquement, même s'ils sont utilisés par plusieurs *Serveurs* agrégés.

La *Propriété StaticNodeIdTypes* fournit une liste des *IdTypes* utilisés pour les *Nœuds* statiques. Tous les *Nœuds* dans l'*AddressSpace* de l'espace de noms qui utilise l'un des *IdTypes* dans la matrice doivent être des *Nœuds* statiques.

La *Propriété StaticNumericNodeIdRange* fournit une liste des *NumericRanges* utilisés pour les *NodeIds* des *Nœuds* statiques. Si la *Propriété StaticNodeIdTypes* contient une entrée propre aux NodeIds numériques, cette Propriété est alors ignorée.

La *Propriété StaticStringNodeIdPattern* fournit une expression régulière telle que spécifiée pour l'*Opérateur Like* défini dans l'IEC 62541-4 pour le filtrage des *NodeIds* de chaîne des *Nœuds* statiques. Lorsque la *Propriété StaticNodeIdTypes* contient une entrée propre aux *NodeIds* de chaîne, cette *Propriété* est alors ignorée.

L'*Objet NamespaceFile* contient tous les *Nœuds* et *Références* de l'espace de noms dans un fichier XML lorsque le schéma XML du Modèle d'information est défini dans l'IEC 62541-6. Le fichier XML est fourni par un *Objet AddressSpaceFileType*.

La *Propriété DefaultRolePermissions* fournit les autorisations par défaut si un *Serveur* prend en charge les *RolePermissions* pour le *Namespace*. Un *Nœud* dans le *Namespace* prend le pas sur cette valeur par défaut en ajoutant au *Nœud* un *Attribut RolePermissions*. Si un *Serveur* met en œuvre un modèle de *RolePermissions* spécifique à un fournisseur pour un *Namespace*, il n'ajoute pas la *Propriété DefaultRolePermissions* à l'*Objet NamespaceMetadata*.

La *Propriété DefaultUserRolePermissions* fournit les autorisations d'utilisateur par défaut si un *Serveur* prend en charge les *UserRolePermissions* pour le *Namespace*. Un *Nœud* dans le *Namespace* prend le pas sur cette valeur par défaut en ajoutant au *Nœud* un *Attribut UserRolePermissions*. Si un *Serveur* met en œuvre un modèle de *UserRolePermissions* spécifique à un fournisseur pour un *Namespace*, il n'ajoute pas la *Propriété DefaultUserRolePermissions* à l'*Objet NamespaceMetadata*.

La *Propriété DefaultAccessRestrictions* est présente si un *Serveur* prend en charge les *AccessRestrictions* pour le *Namespace* et fournit les valeurs par défaut. *Un Nœud* dans le *Namespace* prend le pas sur cette valeur par défaut en ajoutant au *Nœud* un *Attribut AccessRestrictions*. Si un *Serveur* met en œuvre un modèle d'*AccessRestriction* spécifique à un fournisseur pour un *Namespace*, il n'ajoute pas la *Propriété DefaultAccessRestrictions* à l'*Objet NamespaceMetadata*.

6.3.14 NamespacesType

Cet *ObjectType* définit une liste des *Objets NamespaceMetadataType* fournis par le *Serveur*. Il est défini de manière formelle dans le Tableau 22.

Tableau 22 – Définition de NamespacesType

Attribut	Valeur				
BrowseName	NamespacesType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2					
HasComponent	Objet	<NamespaceIdentifier>	-	NamespaceMetadataType	OptionalPlaceholder

L'*ObjectType* contient une liste des *Objets NamespaceMetadataType* qui représentent les espaces de noms dans le *Serveur*. Le *BrowseName* d'un *Objet* doit résulter de l'espace de noms représenté par l'*Objet*. Cela peut, par exemple, être réalisé au moyen de l'indice de l'espace de noms dans la *NamespaceArray* en tant que *namespaceIndex* du *QualifiedName* et de l'URI de l'espace de noms en tant que *name* du QualifiedName. Il convient que les *Clients* ne prennent pas pour hypothèse que tous les espaces de noms fournis par un *Serveur* sont présents dans cette liste dans la mesure où un espace de noms peut ne pas fournir les informations nécessaires pour remplir toutes les *Propriétés* obligatoires du *NamespaceMetadataType*.

6.4 ObjectTypes utilisés comme EventTypes

6.4.1 Généralités

Le présent document définit les *EventTypes* normalisés. Ils sont représentés dans l'*AddressSpace* comme des *ObjectTypes*. Les *EventTypes* sont déjà définis dans l'IEC 62541-3. Les paragraphes ci-après spécifient leur représentation dans l'*AddressSpace*.

6.4.2 BaseEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 23.

Tableau 23 – Définition de BaseEventType

Attribut	**Valeur**				
BrowseName	BaseEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini en 6.2					
HasSubtype	ObjectType	AuditEventType	Défini en 6.4.3		
HasSubtype	ObjectType	SystemEventType	Défini en 6.4.28		
HasSubtype	ObjectType	BaseModelChangeEventType	Défini en 6.4.31		
HasSubtype	ObjectType	SemanticChangeEventType	Défini en 6.4.33		
HasSubtype	ObjectType	EventQueueOverflowEventType	Défini en 6.4.34		
HasSubtype	ObjectType	ProgressEventType	Défini en 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 est généré par le *Serveur* pour identifier de façon unique une *Notification* d'*Evénement* particulière. Le *Serveur* est chargé d'assurer que chaque *Evénement* a un *EventId* unique. Il peut le faire, par exemple, en plaçant des GUID dans la ByteString. Les Clients peuvent utiliser l'*EventId* pour contribuer à réduire le plus possible, voire éliminer les trous et recouvrements qui peuvent se produire au cours d'un basculement de redondance. L'*EventId* doit toujours être retourné comme valeur et le *Serveur* n'est pas autorisé à retourner un *StatusCode* pour l'*EventId* qui indique une erreur.

EventType décrit le type spécifique de l'*Evénement*. *EventType* doit toujours être retourné comme valeur et le *Serveur* n'est pas autorisé à retourner un *StatusCode* pour l'*EventType* qui indique une erreur.

La *Propriété SourceNode* identifie le *Nœud* qui est à l'origine de l'*Evénement*. Si l'*Evénement* n'est pas spécifique à un *Nœud*, le *NodeId* est défini sur nul. Certains sous-types de ce *BaseEventType* peuvent définir des règles complémentaires pour la *Propriété SourceNode*.

SourceName fournit une description de la source de l'*Evénement*. Il peut constituer la partie chaîne du *DisplayName* de la source de l'*Evénement* qui utilise le paramètre de lieu par défaut du serveur, si l'*Evénement* est spécifique à un *Nœud,* ou quelque autre notation spécifique au serveur.

Time fournit l'heure à laquelle l'*Evénement* a eu lieu. Cette valeur est mise à disposition du générateur d'événement dès que possible. Elle provient souvent de l'appareil ou du système sous-jacent. La valeur étant établie, les *Serveurs* OPC UA intermédiaires ne doivent pas modifier la valeur.

ReceiveTime fournit l'heure à laquelle le *Serveur* OPC UA a reçu l'*Evénement* de la part de l'appareil sous-jacent d'un autre *Serveur*. *ReceiveTime* est analogue à *ServerTimestamp* défini dans l'IEC 62541-4, ce qui signifie que dans le cas où le *Serveur* OPC UA reçoit un *Evénement* d'un autre *Serveur* OPC UA, chaque *Serveur* applique son propre *ReceiveTime*. Cela implique qu'un *Client* peut obtenir le même *Evénement,* ayant le même *EventId*, de la part de *Serveurs* différents dont *ReceiveTime* affiche des valeurs différentes. *ReceiveTime* doit toujours être retourné comme valeur et le *Serveur* n'est pas autorisé à retourner un *StatusCode* pour le *ReceiveTime* qui indique une erreur.

LocalTime est une structure contenant les fanions Offset et DaylightSavingInOffset. Le fanion Offset spécifie la différence de temps (en minutes) entre la *Propriété Time* et l'heure de l'emplacement où l'événement a été émis. Si DaylightSavingInOffset est configuré sur TRUE, l'heure d'été/normale du lieu d'origine est en vigueur et Offset inclut la correction de l'heure d'été. S'il est FAUX, le Décalage n'inclut pas la correction de l'heure d'été et celle-ci peut avoir été ou ne pas avoir été en vigueur.

Message fournit une description textuelle en clair et localisable de l'*Evénement*. Le *Serveur* peut retourner n'importe quel texte approprié pour décrire l'*Evénement*. Une chaîne nulle n'est pas une valeur valide; si le *Serveur* n'a pas de description, il doit retourner la partie chaîne du *BrowseName* du *Nœud* associé à l'*Evénement*.

Severity est une indication de l'urgence de l'*Evénement*. Cela s'appelle également communément "priorité". Les valeurs s'étendent entre 1 (la sévérité la plus faible) et 1 000 (la sévérité la plus élevée). Généralement, une sévérité de 1 indique un *Evénement* de nature informative alors qu'une valeur de 1 000 indique un *Evénement* de nature catastrophique, qui peut se traduire par de graves pertes financières ou humaines.

Le nombre de mises en œuvre de *Serveur* supposées prendre en charge 1 000 niveaux de sévérité distincts est très faible. Par conséquent, les développeurs de *Serveur* ont la responsabilité de répartir leurs niveaux de sévérité dans la plage de 1 à 1 000 de manière à ce que les clients puissent mettre en place une distribution linéaire. Par exemple, si un client souhaite présenter cinq niveaux de sévérité à un utilisateur, il convient qu'il soit capable d'effectuer le mapping suivant:

Sévérité client	Sévérité OPC
HIGH	801 à 1 000
MEDIUM HIGH	601 à 800
MEDIUM	401 à 600
MEDIUM LOW	201 à 400
LOW	1 à 200

Dans de nombreux cas, un mapping strictement linéaire vers la plage de Sévérités OPC n'est pas responsable des sévérités des sources sous-jacentes. Au contraire, le développeur de *Serveur* mappe de façon intelligente les sévérités des sources sous-jacentes vers la plage de Sévérités OPC de 1 à 1 000. En particulier, il est recommandé que les développeurs de *Serveur* mappent les *Evénements* de haute urgence vers la plage de sévérités OPC de 667 à 1 000, les *Evénements* de moyenne urgence vers la plage de sévérités OPC de 334 à 666 et les *Evénements* de faible urgence vers la plage de sévérités OPC de 1 à 333.

Par exemple, si une source prend en charge 16 niveaux de sévérité qui sont regroupés afin que les sévérités 0 à 2 soient prises en compte comme LOW, 3 à 7 comme MEDIUM et 8 à 15 comme HIGH, un mapping approprié peut alors être comme suit:

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

Certains *Serveurs* peuvent ne prendre en charge aucun *Evénement* de nature catastrophique et ils peuvent choisir de mapper toutes leurs sévérités vers un sous-ensemble de la plage de 1 à 1 000 (par exemple, 1 à 666). D'autres *Serveurs* peuvent ne prendre en charge aucun *Evénement* de nature purement informationnelle et ils peuvent ainsi choisir de mapper toutes leurs sévérités vers un autre sous-ensemble de la plage de 1 à 1 000 (par exemple, 334 à 1 000).

Cette approche a pour but de permettre à des clients d'utiliser en toute cohérence des valeurs de sévérité issues de plusieurs *Serveurs* provenant de fournisseurs différents. Des informations complémentaires sur la sévérité peuvent être consultées dans l'IEC 62541-9.

6.4.3 AuditEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 24.

Tableau 24 – Définition d'AuditEventType

Attribut	**Valeur**				
BrowseName	AuditEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditSecurityEventType	Défini en 6.4.4		
HasSubtype	ObjectType	AuditNodeManagementEventType	Défini en 6.4.19		
HasSubtype	ObjectType	AuditUpdateEventType	Défini en 6.4.24		
HasSubtype	ObjectType	AuditUpdateMethodEventType	Défini en 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

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2.

ActionTimeStamp identifie l'heure à laquelle l'utilisateur a lancé l'action qui a abouti à la génération de l'*AuditEvent*. Il diffère de la *Propriété Time,* car il s'agit de l'heure à laquelle le serveur a généré l'*AuditEvent* qui documente l'action.

Status identifie si l'action demandée peut être exécutée (définir *Status* sur TRUE) ou non (définir *Status* sur FALSE).

ServerId identifie de façon unique le *Serveur* qui génère l'*Evénement*. Il identifie le *Serveur* de façon unique même dans un scénario de redondance transparente commandée par serveur lorsque plusieurs *Serveurs* peuvent utiliser le même URI.

ClientAuditEntryId contient l'*AuditEntryId* en clair qui est défini dans l'IEC 62541-3.

Le *ClientUserId* identifie l'utilisateur du client demandant une action. Le *ClientUserId* peut être obtenu à partir du *UserIdentityToken* passé dans l'appel d'*ActivateSession*. Si l'*UserIdentityToken* est un *UserNameIdentityToken*, le *ClientUserId* est alors l'*UserName.* Si l'*UserIdentityToken* est un *X509IdentityToken,* le *ClientUserId* est alors le Nom de sujet X509 du *Certificat*. Si l'*UserIdentityToken* est un *IssuedIdentityToken*, le *ClientUserId* doit alors être une chaîne qui représente le propriétaire du jeton. Le meilleur choix de la chaîne dépend du type d'*IssuedIdentityToken.* Si un *AnonymousIdentityToken* a été utilisé, la valeur est nulle.

6.4.4 AuditSecurityEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 25.

Tableau 25 – Définition d'AuditSecurityEventType

Attribut	**Valeur**				
BrowseName	AuditSecurityEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.3, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditChannelEventType	Défini en 6.4.5		
HasSubtype	ObjectType	AuditSessionEventType	Défini en 6.4.7		
HasSubtype	ObjectType	AuditCertificateEventType	Défini en 6.4.12		
HasProperty	Variable	StatusCodeId	StatusCode	PropertyType	Optional

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

La *Propriété* facultative StatusCodeId fournit l'erreur de sécurité exacte responsable de la production de l'*Evénement.*

6.4.5 AuditChannelEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 26.

Tableau 26 – Définition d'AuditChannelEventType

Attribut	**Valeur**				
BrowseName	AuditChannelEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type de l'*AuditSecurityEventType* défini en 6.4.4, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditOpenSecureChannelEventType	Défini en 6.4.6		
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSecurityEventType*. Leur sémantique est définie en 6.4.4. La *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée à l'*Objet Serveur*. Le *SourceName* pour les *Evénements* de ce type doit être "SecureChannel/" et le *Service* qui génère l'*Evénement* (par exemple SecureChannel/*OpenSecureChannel* ou SecureChannel/*CloseSecureChannel*). Si le *ClientUserId* n'est pas disponible pour un appel *CloseSecureChannel*, ce paramètre doit être défini sur "System/CloseSecureChannel".

Le *SecureChannelId* doit identifier de façon unique le SecureChannel. L'application doit utiliser le même identificateur dans tous les *AuditEvents* relatifs au Jeu de Services Session (AuditCreateSessionEventType, AuditActivateSessionEventType et leurs sous-types) et au Jeu de Services SecureChannel (AuditChannelEventType et ses sous-types).

6.4.6 AuditOpenSecureChannelEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 27.

Tableau 27 – Définition d'AuditOpenSecureChannelEventType

Attribut	Valeur				
BrowseName	AuditOpenSecureChannelEventType				
IsAbstract	True				
Références	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type de l'*AuditChannelEventType* défini en 6.4.5, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
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

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditChannelEventType*. Leur sémantique est définie en 6.4.5. Le *SourceName* pour les *Evénements* de ce type doit être "SecureChannel/OpenSecureChannel". Le *ClientUserId* n'est pas disponible pour cet appel; ce paramètre doit donc être défini sur "System/OpenSecureChannel".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

ClientCertificate est le paramètre clientCertificate d'appel du *Service* OpenSecureChannel.

ClientCertificateThumbprint est une empreinte caractéristique du *ClientCertificate.* Voir IEC 62541-6 pour plus d'informations concernant les empreintes.

RequestType est le paramètre requestType d'appel du *Service* OpenSecureChannel.

SecurityPolicyUri est le paramètre securityPolicyUri d'appel du *Service* OpenSecureChannel.

SecurityMode est le paramètre securityMode d'appel du *Service* OpenSecureChannel.

RequestedLifetime est le paramètre requestedLifetime d'appel du *Service* OpenSecureChannel.

6.4.7 AuditSessionEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 28.

Tableau 28 – Définition d'AuditSessionEventType

Attribut	Valeur				
BrowseName	AuditSessionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSecurityEventType* défini en 6.4.4, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditCreateSessionEventType	Défini en 6.4.8		
HasSubtype	ObjectType	AuditActivateSessionEventType	Défini en 6.4.10		
HasSubtype	ObjectType	AuditCancelEventType	Défini en 6.4.11		
HasProperty	Variable	SessionId	NodeId	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSecurityEventType*. Leur sémantique est définie en 6.4.4.

Si l'*Evénement* est généré par un appel de *Service TransferSubscriptions*, la *Propriété SourceNode* doit être affectée à l'*Objet SessionDiagnostics* qui représente la session. Le *SourceName* pour les *Evénements* de ce type doit être "Session/TransferSubscriptions".

Autrement, la *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée à l'*Objet Serveur*. Le *SourceName* pour les *Evénements* de ce type doit être "Session/" et le *Service* ou la cause qui génère l'*Evénement* (par exemple *CreateSession*, *ActivateSession* ou *CloseSession*).

Le *SessionId* doit contenir le *SessionId* de la session sur laquelle l'appel de *Service* a été émis. Dans le *Service CreateSession*, il doit être défini sur le *SessionId* nouvellement créé. En l'absence de contexte de session (par exemple pour un appel de *Service CreateSession* non abouti), le *SessionId* doit être défini sur nul.

6.4.8 AuditCreateSessionEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 29.

Tableau 29 – Définition d'AuditCreateSessionEventType

Attribut	Valeur				
BrowseName	AuditCreateSessionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSessionEventType* défini en 6.4.7, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditUrlMismatchEventType	Défini en 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

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.7. Le *SourceName* pour les *Evénements* de ce type doit être

"Session/CreateSession". Le *ClientUserId* n'est pas disponible pour cet appel; ce paramètre doit donc être défini sur "System/CreateSession".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

Le *SecureChannelId* doit identifier de façon unique le SecureChannel. L'application doit utiliser le même identificateur dans tous les *AuditEvents* relatifs au Jeu de Services Session (AuditCreateSessionEventType, AuditActivateSessionEventType et leurs sous-types) et au Jeu de Services SecureChannel (AuditChannelEventType et ses sous-types).

ClientCertificate est le paramètre clientCertificate de l'appel de *Service* CreateSession.

ClientCertificateThumbprint est une empreinte caractéristique du *ClientCertificate.* Voir IEC 62541-6 pour plus d'informations concernant les empreintes.

RevisedSessionTimeout est le paramètre revisedSessionTimeout qui est retourné lors de l'appel de *Service* CreateSession.

6.4.9 AuditUrlMismatchEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 30.

Tableau 30 – Définition d'AuditUrlMismatchEventType

Attribut	**Valeur**				
BrowseName	AuditUrlMismatchEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCreateSessionEventType* défini en 6.4.8 ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	EndpointUrl	String	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.8.

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

EndpointUrl est le paramètre endpointUrl de l'appel de *Service* CreateSession.

6.4.10 AuditActivateSessionEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 31.

Tableau 31 – Définition d'AuditActivateSessionEventType

Attribut	Valeur				
BrowseName	AuditActivateSessionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type de l'*AuditSessionEventType* défini en 6.4.7, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	ClientSoftwareCertificates	SignedSoftwareCertificate[]	PropertyType	Mandatory
HasProperty	Variable	UserIdentityToken	UserIdentityToken	PropertyType	Mandatory
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.7. Le *SourceName* pour les *Evénements* de ce type doit être "Session/ActivateSession".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

ClientSoftwareCertificates est le paramètre clientSoftwareCertificates de l'appel de *Service* ActivateSession.

UserIdentityToken reflète le paramètre userIdentityToken de l'appel de *Service* ActivateSession. Pour les jetons Username/Password, le mot de passe ne doit pas être inclus.

Le *SecureChannelId* doit identifier de façon unique le SecureChannel. L'application doit utiliser le même identificateur dans tous les *AuditEvents* relatifs au Jeu de Services Session (AuditCreateSessionEventType, AuditActivateSessionEventType et leurs sous-types) et au Jeu de Services SecureChannel (AuditChannelEventType et ses sous-types).

6.4.11 AuditCancelEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 32.

Tableau 32 – Définition d'AuditCancelEventType

Attribut	Valeur				
BrowseName	AuditCancelEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSessionEventType* défini en 6.4.7, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	RequestHandle	UInt32	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.7. Le *SourceName* pour les *Evénements* de ce type doit être "Session/Cancel".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

RequestHandle est le paramètre requestHandle de l'appel de *Service* Cancel.

6.4.12 AuditCertificateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 33.

Tableau 33 – Définition d'AuditCertificateEventType

Attribut	**Valeur**				
BrowseName	AuditCertificateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type de l'*AuditSecurityEventType* défini en 6.4.7, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditCertificateDataMismatchEventType	Défini en 6.4.13		
HasSubtype	ObjectType	AuditCertificateExpiredEventType	Défini en 6.4.14		
HasSubtype	ObjectType	AuditCertificateInvalidEventType	Défini en 6.4.15		
HasSubtype	ObjectType	AuditCertificateUntrustedEventType	Défini en 6.4.16		
HasSubtype	ObjectType	AuditCertificateRevokedEventType	Défini en 6.4.17		
HasSubtype	ObjectType	AuditCertificateMismatchEventType	Défini en 6.4.18		
HasProperty	Variable	Certificat	ByteString	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSecurityEventType*. Leur sémantique est définie en 6.4.4. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate".

Certificat est le certificat qui a rencontré un problème de validation. Des sous-types complémentaires de cet *EventType* sont définis et représentent les erreurs individuelles de validation. Ce certificat peut être mappé vers le *Service* qui l'a passé (Jeu de Services Session ou SecureChannel), car les *AuditEvents* de ces *Services* incluent aussi le certificat.

6.4.13 AuditCertificateDataMismatchEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 34.

Tableau 34 – Définition d'AuditCertificateDataMismatchEventType

Attribut	**Valeur**				
BrowseName	AuditCertificateDataMismatchEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	InvalidHostname	String	PropertyType	Mandatory
HasProperty	Variable	InvalidUri	String	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate".

InvalidHostname est la chaîne qui représente le nom d'hôte transmis comme partie de l'URL qui a été détecté non valide. Si le nom d'hôte n'est pas non valide, il peut être nul.

InvalidUri est l'URI qui a été transmis et qui ne correspond pas au contenu du certificat. Si l'URI n'est pas non valide, il peut être nul.

L'*InvalidHostname* ou bien l'*InvalidUri* doit être fourni.

6.4.14 AuditCertificateExpiredEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 35.

Tableau 35 – Définition d'AuditCertificateExpiredEventType

Attribut	Valeur				
BrowseName	AuditCertificateExpiredEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate". La *Variable Message* doit inclure une description des raisons pour lesquelles le certificat a expiré (c'est-à-dire le temps avant le début ou le temps après la fin). Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.15 AuditCertificateInvalidEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 36.

Tableau 36 – Définition d'AuditCertificateInvalidEventType

Attribut	Valeur				
BrowseName	AuditCertificateInvalidEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate". Le *Message* doit inclure une description de la non-validité du certificat. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.16 AuditCertificateUntrustedEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 37.

Tableau 37 – Définition d'AuditCertificateUntrustedEventType

<table>
<tr><th colspan="2">Attribut</th><th colspan="4">Valeur</th></tr>
<tr><td colspan="2">BrowseName</td><td colspan="4">AuditCertificateUntrustedEventType</td></tr>
<tr><td colspan="2">IsAbstract</td><td colspan="4">True</td></tr>
<tr><td><b>Références</b></td><td><b>NodeClass</b></td><td><b>BrowseName</b></td><td><b>DataType</b></td><td><b>TypeDefinition</b></td><td><b>ModellingRule</b></td></tr>
<tr><td colspan="6">Sous-type de l'<i>AuditCertificateEventType</i> défini en 6.4.12, ce qui signifie qu'il hérite des <i>InstanceDeclarations</i> de ce Nœud.</td></tr>
</table>

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate". La *Variable Message* doit inclure une description des raisons pour lesquelles il n'est pas accordé de confiance au certificat. Si une chaîne de confiance est impliquée, il convient de décrire le certificat qui a échoué dans la chaîne de confiance. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.17 AuditCertificateRevokedEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 38.

Tableau 38 – Définition d'AuditCertificateRevokedEventType

<table>
<tr><th colspan="2">Attribut</th><th colspan="4">Valeur</th></tr>
<tr><td colspan="2">BrowseName</td><td colspan="4">AuditCertificateRevokedEventType</td></tr>
<tr><td colspan="2">IsAbstract</td><td colspan="4">True</td></tr>
<tr><td><b>Références</b></td><td><b>NodeClass</b></td><td><b>BrowseName</b></td><td><b>DataType</b></td><td><b>TypeDefinition</b></td><td><b>ModellingRule</b></td></tr>
<tr><td colspan="6">Sous-type de l'<i>AuditCertificateEventType</i> défini en 6.4.12, ce qui signifie qu'il hérite des <i>InstanceDeclarations</i> de ce Nœud.</td></tr>
</table>

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate". La *Variable Message* doit inclure une description des raisons de la révocation du certificat (une liste de révocation est-elle disponible ou le certificat figure-t-il sur la liste). Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.18 AuditCertificateMismatchEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 39.

Tableau 39 – Définition d'AuditCertificateMismatchEventType

Attribut	Valeur				
BrowseName	AuditCertificateMismatchEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Le *SourceName* pour les *Evénements* de ce type doit être "Security/Certificate". La *Variable Message* doit inclure une description de l'utilisation impropre du certificat. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.19 AuditNodeManagementEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 40.

Tableau 40 – Définition d'AuditNodeManagementEventType

Attribut	Valeur				
BrowseName	AuditNodeManagementEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.3, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditAddNodesEventType			
HasSubtype	ObjectType	AuditDeleteNodesEventType			
HasSubtype	ObjectType	AuditAddReferencesEventType			
HasSubtype	ObjectType	AuditDeleteReferencesEventType			

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*. La *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée à l'*Objet Serveur*. Le *SourceName* pour les *Evénements* de ce type doit être "NodeManagement/" et le *Service* qui génère l'*Evénement* (par exemple *AddNodes*, *AddReferences*, *DeleteNodes*, *DeleteReferences*).

6.4.20 AuditAddNodesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 41.

Tableau 41 – Définition d'AuditAddNodesEventType

Attribut	**Valeur**				
BrowseName	AuditAddNodesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.19, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	NodesToAdd	AddNodesItem[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.19. Le *SourceName* pour les *Evénements* de ce type doit être "NodeManagement/AddNodes".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

NodesToAdd est le paramètre NodesToAdd de l'appel de *Service* AddNodes.

6.4.21 AuditDeleteNodesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 42.

Tableau 42 – Définition d'AuditDeleteNodesEventType

Attribut	**Valeur**				
BrowseName	AuditDeleteNodesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.19, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	NodesToDelete	DeleteNodesItem[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.19. Le *SourceName* pour les *Evénements* de ce type doit être "NodeManagement/DeleteNodes".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

NodesToDelete est le paramètre nodesToDelete de l'appel de *Service* DeleteNodes.

6.4.22 AuditAddReferencesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 43.

Tableau 43 – Définition d'AuditAddReferencesEventType

Attribut	**Valeur**				
BrowseName	AuditAddReferencesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.19, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	ReferencesToAdd	AddReferencesItem[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.19. Le *SourceName* pour les *Evénements* de ce type doit être "NodeManagement/AddReferences".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

ReferencesToAdd est le paramètre referencesToAdd de l'appel de *Service* AddReferences.

6.4.23 AuditDeleteReferencesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 44.

Tableau 44 – Définition d'AuditDeleteReferencesEventType

Attribut	**Valeur**				
BrowseName	AuditDeleteReferencesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.19, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	ReferencesToDelete	DeleteReferencesItem[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.19. Le *SourceName* pour les *Evénements* de ce type doit être "NodeManagement/DeleteReferences".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres d'appel du *Service* qui déclenche l'*Evénement*.

ReferencesToDelete est le paramètre referencesToDelete de l'appel de *Service* DeleteReferences.

6.4.24 AuditUpdateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 45.

Tableau 45 – Définition d'AuditUpdateEventType

Attribut	Valeur				
BrowseName	AuditUpdateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.3, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	AuditWriteUpdateEventType	Défini en 6.4.25		
HasSubtype	ObjectType	AuditHistoryUpdateEventType	Défini en 6.4.26		

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. La *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée au *NodeId* qui a été modifié. Le *SourceName* pour les *Evénements* de ce type doit être "Attribute/" et le *Service* qui a généré l'événement (par exemple, *Write*, *HistoryUpdate*). Un même appel de *Service* peut générer plusieurs *Evénements* de ce type, un par valeur modifiée.

6.4.25 AuditWriteUpdateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 46.

Tableau 46 – Définition d'AuditWriteUpdateEventType

Attribut	Valeur				
BrowseName	AuditWriteUpdateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateEventType* défini en 6.4.24, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	AttributeID	UInt32	PropertyType	Mandatory
HasProperty	Variable	IndexRange	NumericRange	PropertyType	Mandatory
HasProperty	Variable	NewValue	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	OldValue	BaseDataType	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateEventType*. Le *SourceName* pour les *Evénements* de ce type doit être "Attribute/Write". Leur sémantique est définie en 6.4.24.

AttributeId identifie l'*Attribut* qui a été écrit. La *Propriété SourceNode* identifie le *Nœud* qui a été écrit.

IndexRange identifie la plage d'indices de l'*Attribut* écrit si l'*Attribut* est une matrice. Si l'*Attribut* n'est pas une matrice ou que la matrice complète a été écrite, *IndexRange* est définie sur nulle.

NewValue identifie la valeur qui a été écrite. Si *IndexRange* est fournie, seules les valeurs de la plage fournie sont présentées.

OldValue identifie la valeur que contient l'*Attribut* avant l'écriture. Si *IndexRange* est fournie, seule la valeur de la plage considérée est présentée. Il est acceptable qu'un *Serveur* qui n'a pas cette information rapporte une valeur nulle.

La *NewValue* et l'*OldValue* contiennent toutes deux une valeur dans le *DataType* et l'encodage utilisé pour écrire la valeur.

6.4.26 AuditHistoryUpdateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 47.

Tableau 47 – Définition d'AuditHistoryUpdateEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryUpdateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateEventType* défini en 6.4.24, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	ParameterDataTypeId	NodeId	PropertyType	Nouveau

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateEventType*. Leur sémantique est définie en 6.4.24.

Le *ParameterDataTypeId* identifie le *DataTypeId* pour le paramètre extensible utilisé par l'HistoryUpdate. Ce paramètre indique le type d'HistoryUpdate qui est en cours d'exécution.

Les sous-types de cet *EventType* sont définis dans l'IEC 62541-11 qui représente les différentes possibilités de manipulation de données historiques.

6.4.27 AuditUpdateMethodEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 48.

Tableau 48 – Définition d'AuditUpdateMethodEventType

Attribut	**Valeur**				
BrowseName	AuditUpdateMethodEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.3, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	MethodId	NodeId	PropertyType	Mandatory
HasProperty	Variable	InputArguments	BaseDataType[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. La *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée au *NodeId* de l'*Objet* sur lequel la *Méthode* réside. Le *SourceName* pour les *Evénements* de ce type doit être "Attribute/Call". Un même appel de *Service* peut générer plusieurs *Evénements* de ce type, un par méthode appelée. Il convient de sous-typer cet *EventType* afin de mieux refléter la fonctionnalité de la méthode et de refléter les modifications apportées à l'espace d'adressage ou les valeurs mises à jour déclenchées par la méthode.

MethodId identifie la méthode qui a été appelée.

InputArguments identifie les arguments d'entrée pour la méthode. Ce paramètre peut être nul si aucun argument d'entrée n'a été fourni.

6.4.28 SystemEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 49.

Tableau 49 – Définition de SystemEventType

Attribut	**Valeur**				
BrowseName	SystemEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	ObjectType	DeviceFailureEventType	Défini en 6.4.29		
HasSubtype	ObjectType	SystemStatusChangeEventType	Défini en 6.4.30		
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.29 DeviceFailureEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 50.

Tableau 50 – Définition de DeviceFailureEventType

Attribut	**Valeur**				
BrowseName	DeviceFailureEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *SystemEventType* défini en 6.4.28, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					

Cet *EventType* hérite de toutes les *Propriétés* du *SystemEventType*. Leur sémantique est définie en 6.4.28. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.30 SystemStatusChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 51.

Tableau 51 – Définition de SystemStatusChangeEventType

Attribut	**Valeur**				
BrowseName	SystemStatusChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *SystemEventType* défini en 6.4.28, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	SystemState	ServerState	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* du *SystemEventType*. Leur sémantique est définie en 6.4.28. La *Propriété SourceNode* et le *SourceName* doivent identifier le système. Le système peut être le *Serveur* lui-même ou un système sous-jacent.

Le *SystemState* spécifie l'état actuel du système. Les modifications apportées au *ServerState* du système doivent déclencher un *SystemStatusChangeEvent*, lorsque l'événement est pris en charge par le système.

6.4.31 BaseModelChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 52.

Tableau 52 – Définition de BaseModelChangeEventType

Attribut	**Valeur**				
BrowseName	BaseModelChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasSubtype	ObjectType	GeneralModelChangeEventType	Défini en 6.4.32		

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*. La *Propriété SourceNode* pour les Evénements de ce type doit être le *Nœud* de la *Vue* qui donne le contexte des modifications. Si l'*AddressSpace* entier est le contexte, la *Propriété SourceNode* est définie sur le *NodeId* de l'*Objet Serveur*. Le *SourceName* pour les *Evénements* de ce type doit être la partie *Chaîne* du *BrowseName* de la *Vue*; pour l'*AddressSpace* entier, il doit être "Serveur".

6.4.32 GeneralModelChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 53.

Tableau 53 – Définition de GeneralModelChangeEventType

Attribut	**Valeur**				
BrowseName	GeneralModelChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseModelChangeEventType* défini en 6.4.31, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	Changes	ModelChangeStructureDataType[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* du *BaseModelChangeEventType*. Leur sémantique est définie en 6.4.31.

La *Propriété* supplémentaire définie pour cet *EventType* reflète les modifications qui ont émis le *ModelChangeEvent*. Elle doit contenir au moins une entrée dans sa matrice. Sa structure est définie en 12.16.

6.4.33 SemanticChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 54.

Tableau 54 – Définition de SemanticChangeEventType

Attribut	Valeur				
BrowseName	SemanticChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	Changes	SemanticChangeStructureDataType[]	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Il n'existe pas d'autres *Propriétés* définies pour cet *EventType*. La *Propriété SourceNode* pour les Evénements de ce type doit être le *Nœud* de la *Vue* qui donne le contexte des modifications. Si l'*AddressSpace* entier est le contexte, la *Propriété SourceNode* est définie sur le *NodeId* de l'*Objet Serveur*. Le *SourceName* pour les *Evénements* de ce type doit être la partie *Chaîne* du *BrowseName* de la *Vue*; pour l'*AddressSpace* entier, il doit être "Serveur".

La *Propriété* supplémentaire définie pour cet *EventType* reflète les modifications qui ont émis le *SemanticChangeEvent*. Sa structure est définie en 12.17.

6.4.34 EventQueueOverflowEventType

Les *Evénements EventQueueOverflow* sont créés lorsqu'une file d'attente interne d'un MonitoredItem s'abonnant aux *Evénements* dans le *Serveur* déborde. L'IEC 62541-4 définit le moment auquel les *Evénements EventQueueOverflow* internes doivent être générés.

L'*EventType* pour les *Evénements EventQueueOverflow* est défini de manière formelle dans le Tableau 55.

Tableau 55 – Définition d'EventQueueOverflowEventType

Attribut	Valeur				
BrowseName	EventQueueOverflowEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. La *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée au *NodeId* de l'*Objet Serveur*. Le *SourceName* pour les *Evénements* de ce type doit être "Internal/EventQueueOverflow".

6.4.35 ProgressEventType

Les ProgressEvents sont créés pour identifier l'avancement d'une opération. Une opération peut être un appel de *Service* ou toute action spécifique à l'application telle que l'exécution d'un programme.

L'*EventType* pour les *Evénements Progress* est défini de manière formelle dans le Tableau 56.

Tableau 56 – Définition de ProgressEventType

Attribut	**Valeur**				
BrowseName	ProgressEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des *InstanceDeclarations* de ce Nœud.					
HasProperty	Variable	Contexte	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Avancement	UInt16	PropertyType	Mandatory

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. La *Propriété SourceNode* pour les *Evénements* de ce type doit être affectée au *NodeId* de l'*Objet Session* où l'opération a été déclenchée. Le *SourceName* pour les *Evénements* de ce type doit être "Service/<Service Name as defined in IEC 62541-4>" lorsque l'avancement d'un appel de *Service* est présenté.

La *Propriété* supplémentaire *Context* contient les informations de contexte concernant le type d'avancement d'opération consigné. Dans le cas des appels de *Service*, ce doit être un UInt32 contenant la *requestHandle* du *RequestHeader* de l'appel de *Service*.

La *Propriété* supplémentaire *Progress* contient le pourcentage d'achèvement de l'avancement. La valeur doit être comprise entre 0 et 100, où 100 indique la fin de l'opération.

Il est recommandé que les *Serveurs* présentent uniquement les *ProgressEvents* pour les appels de *Service* à la *Session* qui a appelé le *Service*.

6.5 ModellingRuleType

Les *ModellingRules* sont définies dans l'IEC 62541-3. Cet *ObjectType* est utilisé comme étant le type pour les *ModellingRules*. Il est défini de manière formelle dans le Tableau 57.

Tableau 57 – Définition de ModellingRuleType

Attribut	**Valeur**				
BrowseName	ModellingRuleType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini en 6.2					
HasProperty	Variable	NamingRule	NamingRuleType	PropertyType	Mandatory

La *Propriété NamingRule* identifie la *NamingRule* d'une *ModellingRule* définie dans l'IEC 62541-3.

6.6 FolderType

Des Instances de cet *ObjectType* sont utilisées pour organiser l'*AddressSpace* en une hiérarchie de *Nœuds*. Elles représentent le *Nœud* racine d'un sous-arbre et n'ont aucune autre sémantique associée. Cependant, il convient que le *DisplayName* d'une instance du *FolderType*, tel que "ObjectTypes", implique la sémantique associée à son utilisation. Il n'existe pas de Références spécifiées pour cet *ObjectType*. Il est défini de manière formelle dans le Tableau 58.

Tableau 58 – Définition de FolderType

Attribut	Valeur				
BrowseName	FolderType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

6.7 DataTypeEncodingType

Les *DataTypeEncodings* sont définis dans l'IEC 62541-3. Cet *ObjetType* est utilisé comme type pour les *DataTypeEncodings*. L'utilisation de *DataTypeEncodingType* avec *DataTypeDictionaries* est définie dans l'Annexe D. Il n'existe pas de *Références* spécifiées pour cet *ObjectType*. Il est défini de manière formelle dans le Tableau 59.

Tableau 59 – Définition de DataTypeEncodingType

Attribut	Valeur				
BrowseName	DataTypeEncodingType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

6.8 AggregateFunctionType

Cet *ObjectType* définit une *AggregateFunction* prise en charge par un *Serveur* UA. Il est défini de manière formelle dans le Tableau 60.

Tableau 60 – Définition de AggregateFunctionType

Attribut	Valeur				
BrowseName	AggregateFunctionType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

Pour l'*AggregateFunctionType*, l'*Attribut Description* est obligatoire. L'*Attribut Description* fournit une description localisée de l'*AggregateFunction.* Des *AggregateFunctions* spécifiques peuvent être définies dans d'autres parties de l'IEC 62541.

7 VariableTypes normalisés

7.1 Généralités

Généralement, les composantes d'un *VariableType* complexe sont fixes et peuvent être étendues par sous-typage. Cependant, étant donné que chaque *Variable* d'un *VariableType* peut être étendue avec des composantes supplémentaires, le présent document permet d'étendre les *VariableTypes* normalisés définis dans le présent document avec des composantes supplémentaires. Cela permet l'expression d'informations complémentaires dans la TypeDefinition qui sont de toute façon contenues dans chaque *Variable*. Cependant, il n'est pas admis de limiter les composantes des *VariableTypes* normalisés définis dans le présent document. Un exemple d'extension de *VariableTypes* consiste à placer la *Propriété* normalisée NodeVersion définie dans l'IEC 62541-3 dans le *BaseDataVariableType*, énonçant que chaque *DataVariable* du *Serveur* fournit une *NodeVersion*.

7.2 BaseVariableType

Le *BaseVariableType* est le type de base abstrait pour tous les autres *VariableTypes*. Cependant, seuls le *PropertyType* et le *BaseDataVariableType* héritent directement de ce type.

Pour ce *VariableType*, il n'est pas spécifié d'autres *Références* que les *Références HasSubtype*. Il est défini de manière formelle dans le Tableau 61.

Tableau 61 – Définition de BaseVariableType

Attribut	**Valeur**				
BrowseName	BaseVariableType				
IsAbstract	True				
ValueRank	−2 (−2 = Any)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	VariableType	PropertyType	Défini en 7.3		
HasSubtype	VariableType	BaseDataVariableType	Défini en 7.4		

7.3 PropertyType

Le *PropertyType* est un sous-type du *BaseVariableType*. Il est utilisé comme la définition de type de toutes les *Propriétés*. Les *Propriétés* sont définies par leur *BrowseName*; il n'est donc pas nécessaire de leur attribuer une définition de type spécialisée. Il n'est pas admis de sous-typer ce *VariableType*.

Il n'existe pas de *Références* spécifiées pour ce *VariableType*. Il est défini de manière formelle dans le Tableau 62.

Tableau 62 – Définition de PropertyType

Attribut	**Valeur**				
BrowseName	PropertyType				
IsAbstract	False				
ValueRank	−2 (−2 = Any)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseVariableType défini en 7.2					

7.4 BaseDataVariableType

Le *BaseDataVariableType* est un sous-type du *BaseVariableType*. Il est utilisé comme définition de type chaque fois qu'il existe une *DataVariable* n'ayant plus de définition de type concret disponible. Ce *VariableType* est le *VariableType* de base pour les *VariableTypes* des *DataVariables*, et tous les autres *VariableTypes* des *DataVariables* doivent hériter, directement ou indirectement, de celui-ci. Cependant, des *Serveurs* peuvent ne pas pouvoir fournir toutes les *Références HasSubtype* de ce *VariableType* à ses sous-types; il n'est donc pas exigé de fournir cette information.

Pour ce *VariableType*, il n'est pas spécifié d'autres *Références* que les *Références HasSubtype*. Il est défini de manière formelle dans le Tableau 63.

Tableau 63 – Définition de BaseDataVariableType

Attribut	Valeur		
BrowseName	BaseDataVariableType		
IsAbstract	False		
ValueRank	−2 (−2 = Any)		
DataType	BaseDataType		
Références	**NodeClass**	**BrowseName**	**Commentaire**
Sous-type du BaseVariableType défini en 7.2			
HasSubtype	VariableType	ServerVendorCapabilityType	Défini en 7.5
HasSubtype	VariableType	ServerStatusType	Défini en 7.6
HasSubtype	VariableType	BuildInfoType	Défini en 7.7
HasSubtype	VariableType	ServerDiagnosticsSummaryType	Défini en 7.8
HasSubtype	VariableType	SamplingIntervalDiagnosticsArrayType	Défini en 7.9
HasSubtype	VariableType	SamplingIntervalDiagnosticsType	Défini en 7.10
HasSubtype	VariableType	SubscriptionDiagnosticsArrayType	Défini en 7.11
HasSubtype	VariableType	SubscriptionDiagnosticsType	Défini en 7.12
HasSubtype	VariableType	SessionDiagnosticsArrayType	Défini en 7.13
HasSubtype	VariableType	SessionDiagnosticsVariableType	Défini en 7.14
HasSubtype	VariableType	SessionSecurityDiagnosticsArrayType	Défini en 7.15
HasSubtype	VariableType	SessionSecurityDiagnosticsType	Défini en 7.16
HasSubtype	VariableType	OptionSetType	Défini en 7.17

7.5 ServerVendorCapabilityType

Ce *VariableType* est un type abstrait dont les sous-types définissent des fonctions du *Serveur*. Les fournisseurs peuvent définir des sous-types de ce type. Ce *VariableType* est défini de manière formelle dans le Tableau 64.

Tableau 64 – Définition de ServerVendorCapabilityType

Attribut	Valeur				
BrowseName	ServerVendorCapabilityType				
IsAbstract	True				
ValueRank	−1 (−1 = Scalar)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4					

7.6 ServerStatusType

Ce *VariableType* complexe est utilisé pour des informations relatives au statut du *Serveur*. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.10. Le *VariableType* est défini de manière formelle dans le Tableau 65.

Tableau 65 – Définition de ServerStatusType

Attribut	**Valeur**				
BrowseName	ServerStatusType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ServerStatusDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4					
HasComponent	Variable	StartTime	UtcTime	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentTime	UtcTime	BaseDataVariableType	Mandatory
HasComponent	Variable	State	ServerState	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildInfo[a]	BuildInfo	BuildInfoType	Mandatory
HasComponent	Variable	SecondsTillShutdown	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	ShutdownReason	LocalizedText	BaseDataVariableType	Mandatory

[a] Les *Objets* et *Variables* conteneurs de ces *Objets* et *Variables* sont définis par leur *BrowseName* défini dans le *TypeDefinitionNode* correspondant. Le *NodeId* est défini par le nom symbolique composé décrit en 4.1.

7.7 BuildInfoType

Ce *VariableType* complexe est utilisé pour des informations relatives au statut du *Serveur*. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.4. Le *VariableType* est défini de manière formelle dans le Tableau 66.

Tableau 66 – Définition de BuildInfoType

Attribut	**Valeur**				
BrowseName	BuildInfoType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	BuildInfo				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 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

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.9. Le *VariableType* est défini de manière formelle dans le Tableau 67.

Tableau 67 – Définition de ServerDiagnosticsSummaryType

Attribut	**Valeur**				
BrowseName	ServerDiagnosticsSummaryType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ServerDiagnosticsSummaryDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 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

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SamplingIntervalDiagnosticsType avec la fréquence d'échantillonnage comme *BrowseName*. Le *VariableType* est défini de manière formelle dans le Tableau 68.

Tableau 68 – Définition de SamplingIntervalDiagnosticsArrayType

Attribut		Valeur		
BrowseName		SamplingIntervalDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SamplingIntervalDiagnosticsDataType		
Références	**NodeClass**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4				
HasComponent	Variable	SamplingIntervalDiagnostics	SamplingIntervalDiagnosticsDataType SamplingIntervalDiagnosticsType	ExposesItsArray

7.10 SamplingIntervalDiagnosticsType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.8. Le *VariableType* est défini de manière formelle dans le Tableau 69.

Tableau 69 – Définition de SamplingIntervalDiagnosticsType

Attribut		Valeur			
BrowseName		SamplingIntervalDiagnosticsType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		SamplingIntervalDiagnosticsDataType			
Références	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 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

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SubscriptionDiagnosticsType avec le SubscriptionId comme *BrowseName*. Le *VariableType* est défini de manière formelle dans le Tableau 70.

Tableau 70 – Définition de SubscriptionDiagnosticsArrayType

Attribut	Valeur			
BrowseName	SubscriptionDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SubscriptionDiagnosticsDataType			
Références	**NodeClass**	**BrowseName**	**DataType** **TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4				
HasComponent	Variable	SubscriptionDiagnostics	SubscriptionDiagnosticsDataType SubscriptionDiagnosticsType	ExposesItsArray

7.12 SubscriptionDiagnosticsType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.15. Le *VariableType* est défini de manière formelle dans le Tableau 71.

Tableau 71 – Définition de SubscriptionDiagnosticsType

Attribut	Valeur				
BrowseName	SubscriptionDiagnosticsType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	SubscriptionDiagnosticsDataType				
Références	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 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

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SessionDiagnosticsVariableType avec le SessionDiagnostics comme *BrowseName*. Ces

Variables sont également référencées par les *Objets* SessionDiagnostics définis par leur type en 6.3.5. Le *VariableType* est défini de manière formelle dans le Tableau 72.

Tableau 72 – Définition de SessionDiagnosticsArrayType

Attribut	Valeur			
BrowseName	SessionDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionDiagnosticsDataType			
Références	**NodeClass**	**BrowseName**	**DataType** **TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	ExposesItsArray

7.14 SessionDiagnosticsVariableType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.11. Le *VariableType* est défini de manière formelle dans le Tableau 73.

Tableau 73 – Définition de SessionDiagnosticsVariableType

Attribut	**Valeur**			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
Références	**Node Class**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 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

Attribut	Valeur			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
Références	**Node Class**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4				
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 BaseDataVariableType	Mandatory
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

Attribut	Valeur			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
Références	**Node Class**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4				
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

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SessionSecurityDiagnosticsType avec le SessionSecurityDiagnosticsType comme *BrowseName*. Ces *Variables* sont également référencées par les *Objets* SessionDiagnostics définis par leur type en 6.3.5. Le *VariableType* est défini de manière formelle dans le Tableau 74. Ces informations étant relatives à la sécurité, il convient de ne pas les rendre accessibles à tous les utilisateurs, mais seulement aux utilisateurs autorisés.

Tableau 74 – Définition de SessionSecurityDiagnosticsArrayType

Attribut	Valeur			
BrowseName	SessionSecurityDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionSecurityDiagnosticsDataType			
Références	**Node Class**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4				
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	ExposesItsArray

7.16 SessionSecurityDiagnosticsType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* avec la même sémantique que celle définie en 12.12. Le *VariableType* est défini de manière formelle dans le Tableau 75. Ces informations étant relatives à la sécurité, il convient de ne pas les rendre accessibles à tous les utilisateurs, mais seulement aux utilisateurs autorisés.

Tableau 75 – Définition de SessionSecurityDiagnosticsType

Attribut	Valeur			
BrowseName	SessionSecurityDiagnosticsType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionSecurityDiagnosticsDataType			
Références	**Node Class**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 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

Le *VariableType OptionSetType* est utilisé pour représenter un masque de bits. Chaque élément de matrice de la *Propriété OptionSetValues* contient une représentation en clair du bit correspondant utilisé dans l'ensemble d'options ou un *LocalizedText* vide pour un bit qui n'a pas de signification spécifique. L'ordre des bits du masque de bits correspond à une position dans la matrice, ce qui signifie que le premier bit (bit de poids faible) correspond à la première entrée dans la matrice, etc.

Outre ce *VariableType*, le *DataType OptionSet* peut aussi être utilisé pour représenter un masque de bits. A titre de ligne directrice, le *DataType* est utilisé lorsque le masque de bits est fixe et s'applique à plusieurs *Variables.* Le *VariableType* est utilisé lorsque le masque de bits est spécifique à cette *Variable* uniquement.

Le *DataType* de ce *VariableType* doit être capable de représenter un masque de bits. Il doit être soit un *DataType* numérique représentant un entier signé ou non signé, soit une *ByteString*. Par exemple, il peut être le *BitFieldMaskDataType.*

La *Propriété* facultative BitMask fournit le masque de bits dans une matrice de Booléens. Cela permet de s'abonner à des entrées individuelles du masque de bits. L'ordre des bits du masque de bits pointe vers une position de la matrice, ce qui signifie que le premier bit est dirigé vers la première entrée dans la matrice, etc. Le *VariableType* est défini de manière formelle dans le Tableau 74.

Tableau 76 – Définition d'OptionSetType

Attribut	**Valeur**			
BrowseName	OptionSetType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	BaseDataType			
Références	**NodeClass**	**Browse** **Name**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4				
HasProperty	Variable	OptionSetValues	LocalizedText[] PropertyType	Mandatory
HasProperty	Variable	BitMask	Boolean[] PropertyType	Optional

7.18 SelectionListType

Le *VariableType SelectionListType* est utilisé pour une *Variable* quand les valeurs possibles sont fournies par un ensemble de valeurs.

La *Propriété Selections* contient une matrice de valeurs qui représente les valeurs valides pour la valeur de ce *VariableType*.

Le *DataType* de la matrice de la *Propriété Selections* doit être du même *DataType* que ce *VariableType.*

Chaque élément de matrice de la *Propriété* facultative *SelectionDescriptions* contient une représentation en clair de la valeur correspondante dans la *Propriété Selections* et doit être de la même taille de matrice que la *Propriété Selections*.

La valeur de ce *VariableType* peut être restreinte aux seules valeurs définies dans la *Propriété Selections* en définissant la *Propriété RestrictToList* facultative sur une valeur *True*. En l'absence de la *Propriété RestrictToList* ou si elle a une valeur *False*, la valeur n'est alors pas restreinte à l'ensemble défini par la *Propriété Selections*.

Le *VariableType* est défini de manière formelle dans le Tableau 77.

Tableau 77 – Définition de SelectionListType

Attribut	Valeur			
BrowseName	SelectionListType			
IsAbstract	False			
ValueRank	−2 (−2 = Any)			
DataType	BaseDataType			
Références	**NodeClass**	**BrowseName**	**DataType** **TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4				
HasProperty	Variable	Selections	BaseDataType[] PropertyType	Mandatory
HasProperty	Variable	SelectionDescriptions	LocalizedText[] PropertyType	Optional
HasProperty	Variable	RestrictToList	Boolean PropertyType	Optional

7.19 AudioVariableType

Le *VariableType AudioVariableType* définit un type de support au format MIME (extensions multifonctions du courrier Internet) pour la *Propriété* AudibleSound. Le code de texte défini dans l'IETF RFC 2045, l'IETF RFC 2046 et l'IETF RFC 2047 doit être utilisé pour les types de MIME. L'*AudioVariableType* référence le Type de contenu défini comme faisant partie du type MIME et communément utilisé en référence à une MIME spécifique. Le type de support de niveau supérieur est utilisé pour déclarer le type général des données, tandis que le sous-type spécifie un format spécifique pour ce type de données. Aussi, un type de support "audio /xyz" est une description suffisante pour qu'un agent utilisateur puisse déterminer que les données constituent un fichier audio, même si l'agent utilisateur n'a aucune connaissance du format audio spécifique "xyz".

Le *VariableType* est défini de manière formelle dans le Tableau 78.

Tableau 78 – Définition d'AudioVariableType

Attribut	Valeur				
BrowseName	AudioVariableType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	AuditDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4					
HasProperty	Variable	ListId	String	PropertyType	Optional
HasProperty	Variable	AgencyId	String	PropertyType	Optional
HasProperty	Variable	VersionId	String	PropertyType	Optional

8 Objets normalisés et leurs Variables

8.1 Généralités

Les *Objets* et *Variables* décrits dans les paragraphes ci-après peuvent être étendus par des *Propriétés* supplémentaires ou des *Références* à d'autres *Nœuds*, sauf lorsque le texte indique qu'ils sont limités.

8.2 Objets utilisés pour organiser la structure de l'AddressSpace

8.2.1 Vue d'ensemble

Afin de favoriser l'interopérabilité des clients et des *Serveurs*, l'*AddressSpace* OPC UA est structuré sous la forme d'une hiérarchie, les niveaux supérieurs étant normalisés pour tous les *Serveurs*. La Figure 1 représente la structure de l'*AddressSpace*. Tous les *Objets* de cette figure sont organisés à l'aide de *Références Organise* et ont l'*ObjectType FolderType* comme définition de type.

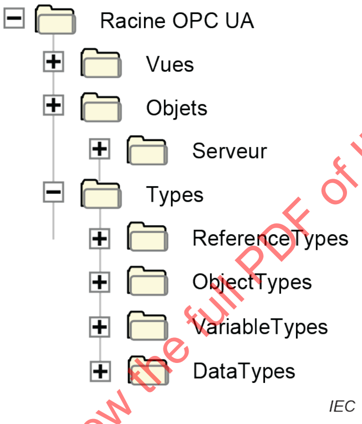


Figure 1 – Structure normalisée de l'AddressSpace

La suite fournit des descriptions de ces *Nœuds* normalisés et l'organisation des *Nœuds* au-dessous d'eux. Généralement, les *Serveurs* mettent en œuvre un sous-ensemble de ces *Nœuds* normalisés en fonction de leurs capacités.

8.2.2 Root

Cet *Objet* normalisé est le point d'entrée de navigation pour l'*AddressSpace*. Il contient un jeu de *Références Organise* qui pointent vers d'autres *Objets* normalisés. L'*Objet Root* ne doit pas référencer d'autres *NodeClasses*. Il est défini de manière formelle dans le Tableau 79.

Tableau 79 – Définition de Root

Attribut	Valeur		
BrowseName	Root		
Références	**NodeClass**	**BrowseName**	**Commentaire**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Objet	Vues	Défini en 8.2.3
Organizes	Objet	Objets	Défini en 8.2.4
Organizes	Objet	Types	Défini en 8.2.5

8.2.3 Vues

Cet *Objet* normalisé est le point d'entrée de navigation pour les *Vues*. Seules les *Références Organise* sont utilisées pour relier des *Nœuds Vue* à l'*Objet* normalisé *Vues*. Tous les *Nœuds Vue* dans l'*AddressSpace* doivent être référencés par ce *Nœud*, directement ou indirectement. C'est-à-dire que l'*Objet Vues* peut référencer d'autres *Objets* en utilisant les *Références Organizes*. Ces *Objets* peuvent référencer des *Vues* supplémentaires. La Figure 2 représente l'organisation des Vues. L'*Objet* normalisé *Vues* référence directement les *Vues* "Vue1" et "Vue2" et indirectement "Vue3" en référençant un autre *Objet* appelé "Engineering".

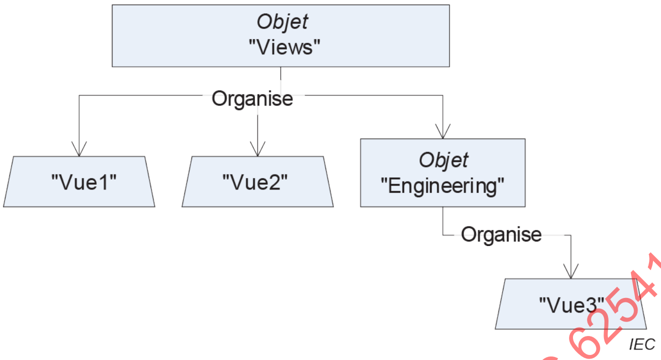


Figure 2 – Organisation des Vues

L'*Objet Vues* ne doit pas référencer d'autres *NodeClasses*. L'*Objet Vues* est défini de manière formelle dans le Tableau 80.

Tableau 80 – Définition de Vues

Attribut	Valeur		
BrowseName	Vues		
Références	**NodeClass**	**BrowseName**	**Commentaire**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6

8.2.4 Objets

Cet *Objet* normalisé est le point d'entrée de navigation pour les *Nœuds Objet*. La Figure 3 représente la structure sous ce *Nœud*. Seules les *Références Organizes* sont utilisées pour relier des *Objets* à l'*Objet* normalisé *Objets*. Un *Nœud Vues* peut être utilisé comme point d'entrée dans un sous-ensemble de l'*Espace d'Adresses* contenant des *Objets* et des *Variables* et, ainsi, l'*Objet* "*Objets*" peut aussi référencer des *Nœuds Vues* en utilisant des *Références Organizes*. L'*Objet* "*Objets*" vise à ce que tous les *Objets* et *Variables* qui ne sont pas utilisés pour des définitions de type ou à d'autres fins organisationnelles (par exemple, organisation des *Vues*) soient accessibles par le biais de *Références hiérarchiques* en commençant à partir de ce *Nœud*. Cependant, elle ne constitue pas une exigence, car tous les *Serveurs* peuvent ne pas être capables de la prendre en charge. Cet *Objet* référence l'*Objet* normalisé *Serveur* défini en 8.3.2.

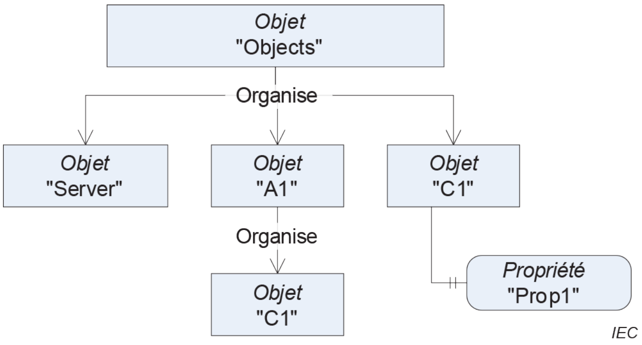


Figure 3 – Organisation des Objects

L'*Objet* "*Objets*" ne doit pas référencer d'autres *NodeClasses*. L'*Objet* "*Objets*" est défini de manière formelle dans le Tableau 81.

Tableau 81 – Définition d'Objets

Attribut	**Valeur**		
BrowseName	Objets		
Références	**NodeClass**	**BrowseName**	**Commentaire**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Objet	Serveur	Défini en 8.3.2

8.2.5 Types

Ce Nœud "*Objet*" normalisé est le point d'entrée de navigation pour le type *Nœuds.* La Figure 1 représente la structure sous ce *Nœud*. Seules les *Références Organizes* sont utilisées pour relier des *Objets* à l'*Objet* normalisé "*Types*". L'*Objet Types* ne doit pas référencer d'autres *NodeClasses*. Il est défini de manière formelle dans le Tableau 82.

Tableau 82 – Définition de Types

Attribut	**Valeur**		
BrowseName	Types		
Références	**NodeClass**	**BrowseName**	**Commentaire**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Objet	ObjectTypes	Défini en 8.2.6
Organizes	Objet	VariableTypes	Défini en 8.2.7
Organizes	Objet	ReferenceTypes	Défini en 8.2.8
Organizes	Objet	DataTypes	Défini en 8.2.9
Organizes	Objet	EventTypes	Défini en 8.2.10

8.2.6 ObjectTypes

Ce *Nœud Objet* normalisé est le point d'entrée de navigation pour les *Nœuds ObjectTypes*. La Figure 4 représente la structure sous ce *Nœud* en présentant certains des *ObjectTypes* normalisés définis à l'Article 6. Seules les *Références Organizes* sont utilisées pour relier des *Objets* et *ObjectTypes* à l'*Objet* normalisé "*ObjectTypes*". L'*Objet ObjectTypes* ne doit pas référencer d'autres *NodeClasses*.