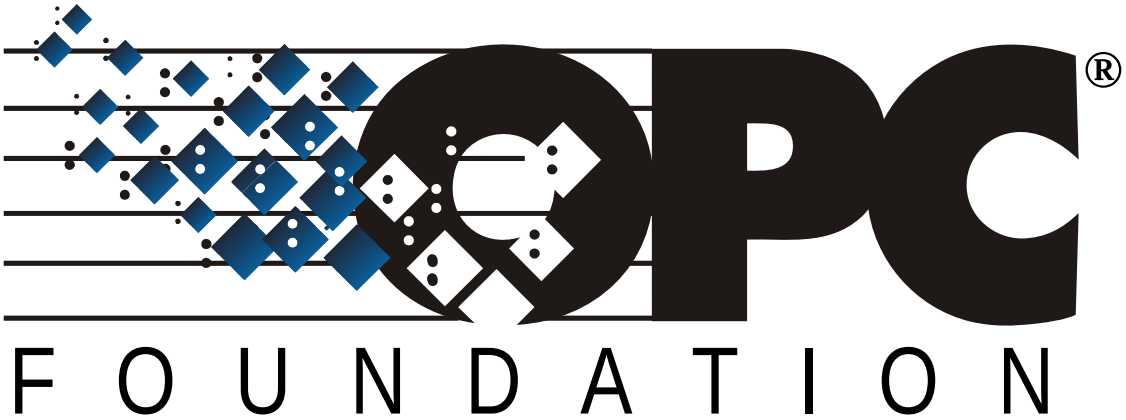


# OPC Unified Architecture

# Specification

# Part 5: Information Model

# Release 1.01

# February 6, 2009

| | | | |
|---|---|---|---|
| Specification Type: | Industry Standard Specification | Comments: | Report or view errata: http://www.opcfoundation.org/errata |
| Title: | OPC Unified Architecture<br><br>Part 5 :Information Model | Date: | February 6, 2009 |
| Version: | Release 1.01 | Software: | MS-Word |
| | | Source: | OPC UA Part 5 - Information Model 1.01 Specification.doc |
| Author: | OPC Foundation | Status: | Release |

# CONTENTS

## FIGURES

## TABLES

# OPC FOUNDATION

# UNIFIED ARCHITECTURE –

## FOREWORD

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



## AGREEMENT OF USE

COMPLIANCE

The OPC Foundation shall at all times be the sole entity that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

TRADEMARKS

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

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

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

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

ISSUE REPORTING

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

# OPC Unified Architecture Specification

# Part 5: Information Model

## 1 Scope

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

## 2 Reference documents

Part 1: **OPC UA Specification: Part 1** – Concepts, Version 1.01 or later
**http://www.opcfoundation.org/UA/Part1/**

Part 2: **OPC UA Specification: Part 2** – Security Model, Version 1.01 or later
**http://www.opcfoundation.org/UA/Part2/**

Part 3: **OPC UA Specification: Part 3** – Address Space Model, Version 1.01 or later
**http://www.opcfoundation.org/UA/Part3/**

Part 4: **OPC UA Specification: Part 4** – Services, Version 1.01 or later
**http://www.opcfoundation.org/UA/Part4/**

Part 6: **OPC UA Specification: Part 6** – Mappings, Version 1.0 or later
**http://www.opcfoundation.org/UA/Part6/**

Part 7: **OPC UA Specification: Part 7** – Profiles, Version 1.0 or later
**http://www.opcfoundation.org/UA/Part7/**

Part 9: **OPC UA Specification: Part 9** – Alarms and Conditions, Version 1.0 or later
**http://www.opcfoundation.org/UA/Part9/**

Part 10: **OPC UA Specification: Part 10** – Programs, Version 1.01 or later
**http://www.opcfoundation.org/UA/Part10/**

Part 11: **OPC UA Specification: Part 11** – Historical Access, Version 1.01 or later
**http://www.opcfoundation.org/UA/Part11/**

## 3 Terms, definitions, and conventions

### 3.1 OPC UA Part 1 terms

The following terms defined in Part 1 apply.

1) AddressSpace
2) Attribute
3) Event
4) Information Model
5) Method
6) MonitoredItem

7) Node
8) NodeClass
9) Notification
10) Object
11) ObjectType
12) Profile
13) Reference
14) ReferenceType
15) Service
16) Service Set
17) Subscription
18) Variable
19) View

### 3.2 OPC UA Part 2 terms

There are no Part 2 terms used in this part.

### 3.3 OPC UA Part 3 terms

The following terms defined in Part 3 apply.

1) DataVariable
2) EventType
3) Hierarchical Reference
4) InstanceDeclaration
5) ModellingRule
6) Property
7) SourceNode
8) TargetNode
9) TypeDefinitionNode
10) VariableType

### 3.4 OPC UA Part 4 terms

There are no Part 4 terms used in this part.

### 3.5 OPC UA Information Model terms

#### 3.5.1 ClientUserId

A String that identifies the user of the client requesting an action.

**NOTE**: 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.

### 3.6 Abbreviations and symbols

| | |
|---|---|
| UA | Unified Architecture |
| XML | Extensible Markup Language |

### 3.7 Conventions for Node descriptions

*Node* definitions are specified using tables (See Table 1)

**Table 1 – Type Definition Table**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| Attribute name | Attribute value. If it is an optional Attribute that is not set "--" will be used. | | | | |
| | | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| *ReferenceType* name | *NodeClass* of the Target*Node*. | *BrowseName* of the target *Node*. If the *Reference* is to be instantiated by the server, then the value of the target Node's BrowseName is "--". | *Attributes* of the referenced *Node*, only applicable for *Variables* and *Objects*. | | Referenced *ModellingRule* of the referenced *Object*. |
| Notes –<br>1) Notes referencing footnotes of the table content. | | | | | |

*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. That implies that the referenced *Node* has a *HasModelParent Reference* with the *Node* defined in the Table as *TargetNode* (see Part 3 for the definition of *ModelParents*).

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

**Table 2 – 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 *NodeId* of a *TypeDefinitionNode*, i.e. the specified *Node* points with a *HasTypeDefinition Reference* to the corresponding *TypeDefinitionNode*. The symbolic name of the *NodeId* is used in the table.

- 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 of this document points to their definition.

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.

Components of *Nodes* can be complex, i.e. 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. Part 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 specification is 0. The namespace for this NamespaceIndex is specified in Part 3.

Note: This specification does not only define 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. But 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 part is specified in the tables defining the *Nodes*. The NamespaceIndex for all *BrowseNames* defined in this part is 0.

# 5 Common Attributes

## 5.1 General

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

**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 specific. |
| Description | Optionally a vendor specific description is provided |
| NodeClass | Shall reflect the *NodeClass* of the *Node* |
| NodeId | The *NodeId* is described by *BrowseNames* as defined in 4.1 and defined in Part 6 |
| WriteMask | Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all *Attributes* to not writeable that are not said to be vendor-specific. For example, the *Description Attribute* may be set to writeable since a Server may provide a server-specific description for the *Node*. The *NodeId* shall not be writeable, because it is defined for each *Node* in this specification. |
| UserWriteMask | Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply. |

## 5.2 Objects

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

**Table 4 – Common Object Attributes**

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

## 5.3 Variables

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

**Table 5 – Common Variable Attributes**

| Attribute | Value |
|---|---|
| MinimumSamplingInterval | Optionally, a vendor-specific minimum sampling interval is provided |
| AccessLevel | The access level for *Variables* used for type definitions is vendor-specific, for all other *Variables* defined in this part, the access level shall allow a current read; other settings are vendor specific. |
| UserAccessLevel | The value for the *UserAccessLevel Attribute* is vendor-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-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-specific.<br>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*. |

## 5.4 VariableTypes

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

**Table 6 – Common VariableType Attributes**

| Attributes | Value |
|---|---|
| Value | Optionally a vendor-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-specific.<br>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*. |

# 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 specification 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 Part. An example of extending the *ObjectTypes* is putting the standard *Property NodeVersion* defined in Part 3 into the *BaseObjectType*, stating that each *Object* of the *Server* will provide a *NodeVersion*.

## 6.2 BaseObjectType

The *BaseObjectType* is used as type definition whenever there is an *Object* having no more concrete type definition 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 7.

**Table 7 – 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 8.

**Table 8 – 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 |
| HasComponent | Variable | ServerStatus[1] | ServerStatusDataType | ServerStatusType | Mandatory |
| HasProperty | Variable | ServiceLevel | Byte | PropertyType | Mandatory |
| HasProperty | Variable | Auditing | Boolean | PropertyType | Mandatory |
| HasComponent | Object | ServerCapabilities[1] | - | ServerCapabilitiesType | Mandatory |
| HasComponent | Object | ServerDiagnostics[1] | - | ServerDiagnosticsType | Mandatory |
| HasComponent | Object | VendorServerInfo | - | VendorServerInfoType | Mandatory |
| HasComponent | Object | ServerRedundancy[1] | - | ServerRedundancyType | Mandatory |

Notes –
1) 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*[1]. Values above 0 are used to identify remote *Servers* and are specific to a *Server*. Part 4 describes discovery mechanism that can be used to resolve URIs into URLs.

The indexes into this 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 this table. A *Server* may add entries to the 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 this 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 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 this table. A *Server* may add entries to the table even if clients are connected to the *Server*. It is recommended that *Servers* not change the indexes of this 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 this table. Clients that cache *NamespaceIndexes* of *NodeIds* should always check when starting a *Session* to verify that the cached *NamespaceIndexes* have not changed.

[1] The URI of the *ServerArray* with Index 0 must be identical to the URI of the *NamespaceArray* with Index 1, since both represent the local server.

*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. The intent is to provide the clients an indication of availability among redundant *Servers*.

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

*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*.

### 6.3.2 ServerCapabilitiesType

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

**Table 9 – ServerCapabilitiesType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | ServerCapabilitiesType | | | | |
| IsAbstract | False | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Modelling Rule** |
| 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 | SoftwareCertificate[] | PropertyType | Mandatory |
| HasComponent | Object | ModellingRules | | FolderType | Mandatory |
| HasComponent | Object | AggregateFunctions | | FolderType | Mandatory |
| HasComponent | Variable | Vendor specific *Variables* of a subtype of the ServerVendorCapabilityType defined in 7.5 | | | -- |

*ServerProfileArray* defines the conformance profile of the *Server*. See Part 7 for the definitions of *Server* profiles.

*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 ReadHistory *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 ReadHistory 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 SoftwareCertificates containing all certificates supported by the *Server*. This shall be the same list of certificates as returned by the CreateSession Service. The certificates in this *Property* are not signed.

*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 subclass 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.

### 6.3.3 ServerDiagnosticsType

This *ObjectType* defines diagnostic information about the OPC UA *Server*. This *ObjectType* is formally defined in Table 10.

**Table 10 – ServerDiagnosticsType Definition**

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

The sampling interval diagnostics are only collected by *Server* 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 7.13. 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 information is collected, and setting the value to FALSE disables the collection without resetting the statistic 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.

### 6.3.4 SessionsDiagnosticsSummaryType

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

**Table 11 – SessionsDiagnosticsSummaryType Definition**

| **Attribute** | **Value** | | | |
|---|---|---|---|---|
| BrowseName | SessionsDiagnosticsSummaryType | | | |
| IsAbstract | False | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType / TypeDefinition** | **ModellingRule** |
| Subtype of the BaseObjectType defined in 6.2 | | | | |
| HasComponent | Variable | SessionDiagnosticsArray | SessionDiagnosticsDataType[]<br>SessionDiagnosticsArrayType | Mandatory |
| HasComponent | Variable | SessionSecurityDiagnosticsArray | SessionSecurityDiagnosticsDataType[]<br>SessionSecurityDiagnosticsArrayType | Mandatory |
| HasComponent | Object | For each Session of the Server one *Object* has to be provided[2] | --<br>SessionDiagnosticsObjectType | -- |

*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*. It has the ClientName of the *Session* as *BrowseName* and 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 12.

**Table 12 – 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<br>SessionDiagnosticsVariableType | Mandatory |
| HasComponent | Variable | SessionSecurityDiagnostics | SessionSecurityDiagnosticsDataType<br>SessionSecurityDiagnosticsType | Mandatory |
| HasComponent | Variable | SubscriptionDiagnosticsArray | SubscriptionDiagnosticsDataType[]<br>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*

[2] : This row represents no *Node* in the *AddressSpace*. It is a placeholder pointing out that instances of the *ObjectType* will have those *Objects*.

SubscriptionDiagnosticsArrayType providing a *Variable* for each entry in the array, as defined in 7.13.

### 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 13.

**Table 13 – 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 14.

**Table 14 – ServerRedundancyType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | ServerRedundancyType | | | | |
| IsAbstract | False | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **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.

### 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 15.

**Table 15 – 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 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 16.

**Table 16 – NonTransparentRedundancyType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | NonTransparentRedundancyType | | | | |
| IsAbstract | False | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the ServerRedundancyType defined in 6.3.7, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| HasProperty | Variable | ServerUriArray | String[] | PropertyType | Mandatory |

*ServerUriArray* is an array with the URI of all redundant servers of the OPC UA *Server*. See Part 1 for the definition of redundancy in this specification. Since, in a non-transparent redundancy environment, the client is responsible to subscribe to the redundant servers, it might or might not open a *Session* to one or more redundant servers of this array.

The redundancy support provided by the *Server* is defined in the *RedundancySupport* (defined in the supertype). The client is allowed to access the redundant sever only as described there, however, ”hot” switchover implies the support of “warm” switchover and “warm” switchover implies the support of “cold” switchover.

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.

## 6.4 ObjectTypes used as EventTypes

### 6.4.1 General

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

### 6.4.2 BaseEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 17.

**Table 17 – BaseEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | BaseEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| 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 6.4.30 | | |
| HasSubtype | ObjectType | SemanticChangeEventType | Defined in 6.4.32 | | |
| HasSubtype | ObjectType | EventQueueOverflowEventType | Defined in 6.4.33 | | |
| 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.

*EventType* describes the specific type of *Event*.

*SourceNode* 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 *SourceNode*.

*SourceName* provides a description of the source of the *Event*. This could be the *DisplayName* of the *Event* source – if the *Event* is specific to a *Node* – or some server-specific notation.

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

*ReceiveTime* provides the time the OPC UA *Server* received the *Event* from the underlying device of another *Server*. *ReceiveTime* is analogous to *ServerTimestamp* defined in Part 4, i.e. in the case where the OPC UA *Server* gets an *Event* from another OPC UA *Server*, each *Server* applies its own *ReceiveTime*. That implies that a *Client* may get the same *Event* – having the same *EventId* – from different *Servers* having different values of the *ReceiveTime*.

*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 1000, with 1 being the lowest severity and 1000 being the highest. Typically, a severity of 1 would indicate an *Event* which is informational in nature, while a value of 1000 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 1000 distinct severity levels. Therefore, *Server* developers are responsible for distributing their severity levels across the 1 – 1000 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 – 1000 |
| MEDIUM HIGH | 601 – 800 |
| MEDIUM | 401 – 600 |
| MEDIUM LOW | 201 – 400 |
| LOW | 1 – 200 |

In many cases a strict linear mapping of underlying source severities to the OPC Severity range is not appropriate. The *Server* developer will instead intelligently map the underlying source severities to the 1 – 1000 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 – 1000, *Events* of medium urgency into the OPC severity range of 334 – 666 and *Events* of low urgency into OPC severities of 1 – 333.

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

| OPC Range | Source Severity | OPC Severity |
|---|---|---|
| HIGH (667 – 1000) | 15 | 1000 |
| | 14 | 955 |
| | 13 | 910 |
| | 12 | 865 |
| | 11 | 820 |
| | 10 | 775 |
| | 9 | 730 |
| | 8 | 685 |
| MEDIUM (334 – 666) | 7 | 650 |
| | 6 | 575 |
| | 5 | 500 |
| | 4 | 425 |
| | 3 | 350 |
| LOW (1 – 333) | 2 | 300 |
| | 1 | 150 |
| | 0 | 1 |

Some servers might not support any *Events* which are catastrophic in nature, so they may choose to map all of their severities into a subset of the 1 – 1000 range (for example, 1 – 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 – 1000 range (for example, 334 – 1000).

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

### 6.4.3 AuditEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 18.

**Table 18 – AuditEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *BaseEventType* defined in 6.4.2, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| HasSubtype | ObjectType | AuditSecurityEventType | Defined in 6.4.4 | | |
| HasSubtype | ObjectType | AuditNodeManagementEventType | Defined in 6.4.19 | | |
| HasSubtype | ObjectType | AuditUpdateEventType | Defined in 6.4.24 | | |
| HasSubtype | ObjectType | AuditUpdateMethodEventType | Defined in 6.4.27 | | |
| HasProperty | Variable | ActionTimeStamp | UtcTime | PropertyType | Mandatory |
| HasProperty | Variable | Status | Boolean | PropertyType | Mandatory |
| HasProperty | Variable | ServerId | String | PropertyType | Mandatory |
| HasProperty | Variable | ClientAuditEntryId | String | PropertyType | Mandatory |
| HasProperty | Variable | ClientUserId | String | PropertyType | Mandatory |

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

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

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

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

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

The *ClientUserId* identifies the user of the client requesting an action. The *ClientUserId* can be obtained from the *UserIdentityToken* passed in the *ActivateSession* call. This token can contain the information in multiple formats depending on the type of User Identity that is passed to the service. If the *UserIdentityToken* that was passed was defined as a UserName, then the structure contains an explicit string that is the user. If the passed *UserIdentityToken* was defined as X509v3, then the CertificateData byte string contains an element that is the user string which can be extracted from the subject key in this structure. If the passed *UserIdentityToken* was defined as WSS, then the user string can be extracted from the WS-Security XML token. If an *AnonymousIdentityToken* was used, the value is null.

### 6.4.4 AuditSecurityEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 19.

**Table 19 – AuditSecurityEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditSecurityEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the AuditEventType defined in 6.4.3, i.e. inheriting 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 | | |

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*.

### 6.4.5 AuditChannelEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 20.

**Table 20 – AuditChannelEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditChannelEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the AuditSecurityEventType defined in 6.4.4, i.e. inheriting 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* for *Events* of this type should be assigned to the *Server Object*. The *SourceName* for *Events* of this type should 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* (AuditSessionEventType and its subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

### 6.4.6 AuditOpenSecureChannelEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 21.

**Table 21 – AuditOpenSecureChannelEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditOpenSecureChannelEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Modelling Rule** |
| Subtype of the *AuditChannelEventType* defined in 6.4.5, i.e. inheriting 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 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.*

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

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

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

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

### 6.4.7 AuditSessionEventType

This *EventType* is defined in Part 3. Its representation in the *AddressSpace* is formally defined in Table 22.

**Table 22 – AuditSessionEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditSessionEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditEventType* defined in 6.4.4, i.e. inheriting 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*. Their semantic is defined in 6.4.4. There are no additional *Properties* defined for this *EventType*.

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

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

The *SessionId* should 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 NULL.

### 6.4.8 AuditCreateSessionEventType

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

**Table 23 – AuditCreateSessionEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCreateSessionEventType | | | | |
| 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. | | | | | |
| 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 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* (AuditSessionEventType and its 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.*

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

### 6.4.9 AuditUrlMismatchEventType

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

**Table 24 – AuditUrlMismatchEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditUrlMismatchEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the AuditCreateSessionEventType defined in 6.4.8, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| HasProperty | Variable | EndpointUrl | String | PropertyType | Mandatory |

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

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

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

### 6.4.10 AuditActivateSessionEventType

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

**Table 25 – AuditActivateSessionEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditActivateSessionEventType | | | | |
| 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 | ClientSoftwareCertificates | SignedSoftwareCertificate[] | PropertyType | Mandatory |
| HasProperty | Variable | UserIdentityToken | UserIdentityToken | 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 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 NOT be included.

### 6.4.11 AuditCancelEventType

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

**Table 26 – 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 be “Session/Cancel”.

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

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

### 6.4.12 AuditCertificateEventType

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

**Table 27 – AuditCertificateEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCertificateEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditSecurityEventType* defined in 6.4.7, i.e. inheriting 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 be “Security/Certificate”.

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

### 6.4.13 AuditCertificateDataMismatchEventType

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

**Table 28 – 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 be “Security/Certificate”.

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

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

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

### 6.4.14 AuditCertificateExpiredEventType

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

**Table 29 – AuditCertificateExpiredEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCertificateExpiredEventType | | | | |
| 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. | | | | | |

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

### 6.4.15 AuditCertificateInvalidEventType

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

**Table 30 – AuditCertificateInvalidEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCertificateInvalidEventType | | | | |
| 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. | | | | | |

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

include a description of why the certificate is invalid. There are no additional *Properties* defined for this *EventType*.

### 6.4.16 AuditCertificateUntrustedEventType

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

**Table 31 – AuditCertificateUntrustedEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCertificateUntrustedEventType | | | | |
| 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. | | | | | |

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

### 6.4.17 AuditCertificateRevokedEventType

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

**Table 32 – AuditCertificateRevokedEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCertificateRevokedEventType | | | | |
| 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. | | | | | |

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

### 6.4.18 AuditCertificateMismatchEventType

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

**Table 33 – AuditCertificateMismatchEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditCertificateMismatchEventType | | | | |
| 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. | | | | | |

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

### 6.4.19 AuditNodeManagementEventType

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

**Table 34 – AuditNodeManagementEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditNodeManagementEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditEventType* defined in 6.4.3, i.e. inheriting 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* for *Events* of this type should be assigned to the *Server Object*. The *SourceName* for *Events* of this type should be "NodeManagement/" and the *Service* that generates the *Event* (e.g. *AddNodes*, *AddReferences*, *DeleteNodes*, *DeleteReferences*).

### 6.4.20 AuditAddNodesEventType

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

**Table 35 – AuditAddNodesEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditAddNodesEventType | | | | |
| 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 | 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 be "NodeManagement/AddNodes".

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

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

### 6.4.21 AuditDeleteNodesEventType

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

**Table 36 – 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 be "NodeManagement/DeleteNodes".

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

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

### 6.4.22 AuditAddReferencesEventType

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

**Table 37 – AuditAddReferencesEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditAddReferencesEventType | | | | |
| 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 | 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 be "NodeManagement/AddReferences".

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

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

### 6.4.23 AuditDeleteReferencesEventType

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

**Table 38 – AuditDeleteReferencesEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditDeleteReferencesEventType | | | | |
| 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 | 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 be "NodeManagement/DeleteReferences".

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

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

### 6.4.24 AuditUpdateEventType

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

**Table 39 – AuditUpdateEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditUpdateEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditEventType* defined in 6.4.3, i.e. inheriting 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* for *Events* of this type should be assigned to the *NodeId* that was changed. The *SourceName* for *Events* of this type should be “Attribute/” and the *Service* that generated the event (e.g. *Write*, *HistoryUpdate*). Note that one *Service* call may generate several *Events* of this type, one per changed value.

### 6.4.25 AuditWriteUpdateEventType

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

**Table 40 – AuditWriteUpdateEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditWriteUpdateEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditUpdateEventType* defined in 6.4.24 , i.e. inheriting 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 be “Attribute/Write”. Their semantic is defined in 6.4.24.

*AttributeId* identifies the *Attribute* that was written on the *SourceNode*.

*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 *AttributeIndexRange* is set to null.

*NewValue* identifies the value that was written to the *SourceNode*. If the *AttributeIndexRange* is provided, only the value of that range is shown.

*OldValue* identifies the value that the *SourceNode* contained before the write. If the *AttributeIndexRange* is provided, only the value of that range is shown. It is acceptable for a *Server* that does have this information to report a null value.

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

### 6.4.26 AuditHistoryUpdateEventType

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

**Table 41 – AuditHistoryUpdateEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditHistoryUpdateEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditUpdateEventType* defined in 6.4.24, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| HasProperty | Variable | ParameterDataTypeId | NodeId | PropertyType | New |

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

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

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

### 6.4.27 AuditUpdateMethodEventType

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

**Table 42 – AuditUpdateMethodEventType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | AuditUpdateMethodEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *AuditEventType* defined in 6.4.3, i.e. inheriting 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* for *Events* of this type should be assigned to the *NodeId* of the object that the method resides on. The *SourceName* for *Events* of this type should be “Attribute/Call”. Note that one *Service* call may generate several *Events* of this type, one per method called. This EventType should be further subtyped to better reflect the functionality of the method and to reflect changes to the address space or updated values triggered by the method.

*MethodId* identifies the method that was called.

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

### 6.4.28 SystemEventType

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

**Table 43 – SystemEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | SystemEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| HasSubtype | ObjectType | DeviceFailureEventType | Defined in 6.4.29 | | |
| Subtype of the *BaseEventType* defined in 6.4.2, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |

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

### 6.4.29 DeviceFailureEventType

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

**Table 44 – DeviceFailureEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | DeviceFailureEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *SystemEventType* defined in 6.4.28, i.e. inheriting 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*.

### 6.4.30 BaseModelChangeEventType

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

**Table 45 – BaseModelChangeEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | BaseModelChangeEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *BaseEventType* defined in 6.4.2, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| HasSubtype | ObjectType | GeneralModelChangeEventType | Defined in 6.4.31 | | |

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* for Events of this type should be the *Node* of the *View* that gives the context of the changes. If the whole *AddressSpace* is the context, the *SourceNode* is set to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type should be the *String* part of the *BrowseName* of the *View*; for the whole *AddressSpace* it should be “Server”.

### 6.4.31 GeneralModelChangeEventType

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

**Table 46 – GeneralModelChangeEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | GeneralModelChangeEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *BaseModelChangeEventType* defined in 6.4.30, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| HasProperty | Variable | Changes | ModelChangeStructureDataType[] | PropertyType | Mandatory |

This *EventType* inherits all *Properties* of the *BaseModelChangeEventType*. Their semantic is defined in 6.4.30.

The additional *Property* defined for this *EventType* reflects the changes that issued the *ModelChangeEvent*. Its structure is defined in 12.16.

### 6.4.32 SemanticChangeEventType

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

**Table 47 – SemanticChangeEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | SemanticChangeEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *BaseEventType* defined in 6.4.2, i.e. inheriting 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* for Events of this type should be the *Node* of the *View* that gives the context of the changes. If the whole *AddressSpace* is the context, the *SourceNode* is set to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type should be the *String* part of the *BrowseName* of the *View*, for the whole *AddressSpace* it should be “Server”.

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

### 6.4.33 EventQueueOverflowEventType

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

The *EventType* for *EventQueueOverflow Events* is formally defined in Table 47.

**Table 48 – EventQueueEventType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | EventQueueOverflowEventType | | | | |
| IsAbstract | True | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| Subtype of the *BaseEventType* defined in 6.4.2, i.e. inheriting the InstanceDeclarations of that Node. | | | | | |

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

### 6.5 ModellingRuleType

*ModellingRules* are defined in Part 3. This *ObjectType* is used as the type for the *ModellingRules*. It is formally defined in Table 49.

**Table 49 – ModellingRuleType Definition**

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

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

### 6.6 FolderType

Instances of this *ObjectType* are used to organise the *AddressSpace* into a hierarchy of *Nodes*. They represent the *root Node* of a subtree, and have no other semantics associated with them. However, the *DisplayName* of an instance of the *FolderType*, such as “ObjectTypes”, should imply the semantics associated with the use of it. There are no References specified for this *ObjectType*. It is formally defined in Table 50.

**Table 50 – FolderType Definition**

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

### 6.7 DataTypeEncodingType

*DataTypeEncodings* are defined in Part 3. This *ObjectType* is used as type for the *DataTypeEncodings*. There are no *References* specified for this *ObjectType*. It is formally defined in Table 51.

**Table 51 – 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 Part 3. This *ObjectType* is used as type for the *DataTypeSystems*. There are no *References* specified for this *ObjectType*. It is formally defined in Table 52.

**Table 52 – DataTypeSystemType Definition**

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

### 6.9 AggregateFunctionType

This *ObjectType* defines an *AggregateFunction* supported by a UA *Server*. It is formally defined in Table 53.

**Table 53 – 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 futher Parts of this Specification.

# 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 specification 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 part. An example of extending *VariableTypes* would be putting the standard *Property NodeVersion*, defined in Part 3, into the *BaseDataVariableType*, stating that each *DataVariable* of the *Server* will provide a *NodeVersion*.

## 7.2 BaseVariableType

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

There are no *References*, except for *HasSubtype References*, specified for this *VariableType*. It is formally defined in Table 54.

**Table 54 – 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 55.

**Table 55 – 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 56.

**Table 56 – 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 7.8 |
| HasSubtype | VariableType | BuildInfoType | Defined in 7.9 |
| HasSubtype | VariableType | ServerDiagnosticsSummaryType | Defined in 7.10 |
| HasSubtype | VariableType | SamplingIntervalDiagnosticsArrayType | Defined in 7.11 |
| HasSubtype | VariableType | SamplingIntervalDiagnosticsType | Defined in 7.12 |
| HasSubtype | VariableType | SubscriptionDiagnosticsArrayType | Defined in 7.13 |
| HasSubtype | VariableType | SubscriptionDiagnosticsType | Defined in 7.14 |
| HasSubtype | VariableType | SessionDiagnosticsArrayType | Defined in 7.15 |
| HasSubtype | VariableType | SessionDiagnosticsVariableType | Defined in 7.16 |
| HasSubtype | VariableType | SessionSecurityDiagnosticsArrayType | Defined in 7.17 |
| HasSubtype | VariableType | SessionSecurityDiagnosticsType | Defined in 7.18 |

## 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 57.

**Table 57 – 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 Part 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 58.

**Table 58 – 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 meaning of the *Property* DataTypeVersion is defined in Part 3. The NamespaceURI is the URI for the namespace described by the *Value Attribute* of the DataTypeDictionary.

### 7.7 DataTypeDescriptionType

*DataTypeDescriptions* are defined in Part 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 58.

**Table 59 – 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 meaning of the *Properties* DataTypeVersion and DictionaryFragment is defined in Part 3.

### 7.8 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 60.

**Table 60 – ServerStatusType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | ServerStatusType | | | | |
| IsAbstract | False | | | | |
| ValueRank | -1 (-1 = Scalar) | | | | |
| DataType | ServerStatusDataType | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Modelling Rule** |
| Subtype of the BaseDataVariableType defined in 7.4 | | | | | |
| HasComponent | Variable | StartTime | UtcTime | BaseDataVariableType | Mandatory |
| HasComponent | Variable | CurrentTime | UtcTime | BaseDataVariableType | Mandatory |
| HasComponent | Variable | State | ServerState | BaseDataVariableType | Mandatory |
| HasComponent | Variable | BuildInfo[1] | BuildInfo | BuildInfoType | Mandatory |
| HasComponent | Variable | SecondsTillShutdown | UInt32 | BaseDataVariableType | Mandatory |
| HasComponent | Variable | ShutdownReason | LocalizedText | BaseDataVariableType | Mandatory |

Notes –

1) 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.

### 7.9 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 61.

**Table 61 – 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 |

### 7.10 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 62.

**Table 62 – ServerDiagnosticsSummaryType Definition**

| Attribute | Value | | | | |
|---|---|---|---|---|---|
| BrowseName | ServerDiagnosticsSummaryType | | | | |
| IsAbstract | False | | | | |
| ValueRank | -1 (-1 = Scalar) | | | | |
| DataType | ServerDiagnosticsSummaryDataType | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **ModellingRule** |
| 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 | SamplingRateCount | 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 |

### 7.11 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 63.

**Table 63 – SamplingIntervalDiagnosticsArrayType Definition**

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

### 7.12 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 64.

**Table 64 – SamplingIntervalDiagnosticsType Definition**

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

### 7.13 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 65.

**Table 65 – 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 | SubscriptionDiagnostics | SubscriptionDiagnosticsDataType SubscriptionDiagnosticsType | ExposesItsArray |

### 7.14 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 66.

**Table 66 – 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 |

### 7.15 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 67.

**Table 67 – SessionDiagnosticsArrayType Definition**

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

### 7.16 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 68.

**Table 68 – 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 | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | UnauthorizedRequestsCount | UInt32 BaseDataVariableType | Mandatory |
| HasComponent | Variable | ReadCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | HistoryReadCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | WriteCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | HistoryUpdateCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | CallCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | CreateMonitoredItemsCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | ModifyMonitoredItemsCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | SetMonitoringModeCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | SetTriggeringCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | DeleteMonitoredItemsCount | ServiceCounterDataType BaseDataVariableType | Mandatory |
| HasComponent | Variable | CreateSubscriptionCount | ServiceCounterDataType | Mandatory |

| | | | | |
|---|---|---|---|---|
| | | | BaseDataVariableType | |
| HasComponent | Variable | ModifySubscriptionCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | SetPublishingModeCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | PublishCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | RepublishCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | TransferSubscriptionsCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | DeleteSubscriptionsCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | AddNodesCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | AddReferencesCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | DeleteNodesCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | DeleteReferencesCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | BrowseCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | BrowseNextCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | TranslateBrowsePathsToNodeIdsCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | QueryFirstCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | QueryNextCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | RegisterNodesCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |
| HasComponent | Variable | UnregisterNodesCount | ServiceCounterDataType<br>BaseDataVariableType | Mandatory |

### 7.17 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 69. Since this information is security related, it should not be made accessible to all users, but only to authorised users.

**Table 69 – SessionSecurityDiagnosticsArrayType Definition**

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

### 7.18 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 70. Since this information is security-related, it should not be made accessible to all users, but only to authorised users.

**Table 70 – 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 |

# 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.

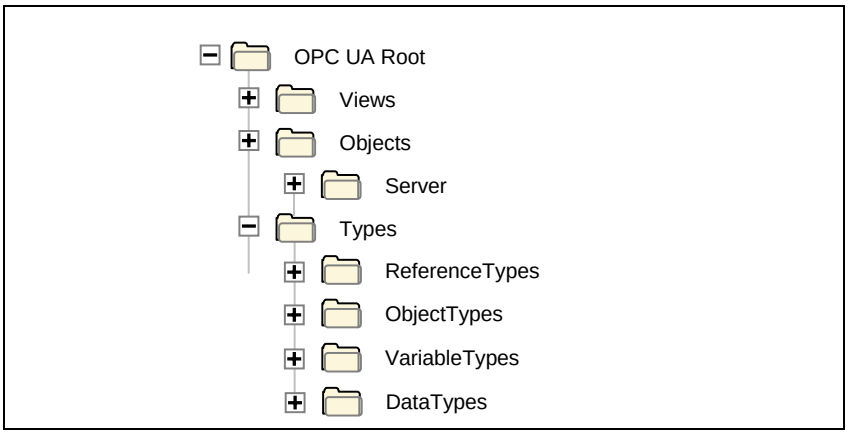


**Figure 1 – Standard AddressSpace Structure**

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 71.

**Table 71 – 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. I.e. the "*Views*" *Object* may reference other *Objects* using *Organizes References.* Those *Objects* may reference additional *Views*. Figure 2 illustrates this. The "*Views"* standard *Object* directly references the *Views* "View1" and "View2" and indirectly "View3" by referencing another *Object* called "Engineering".

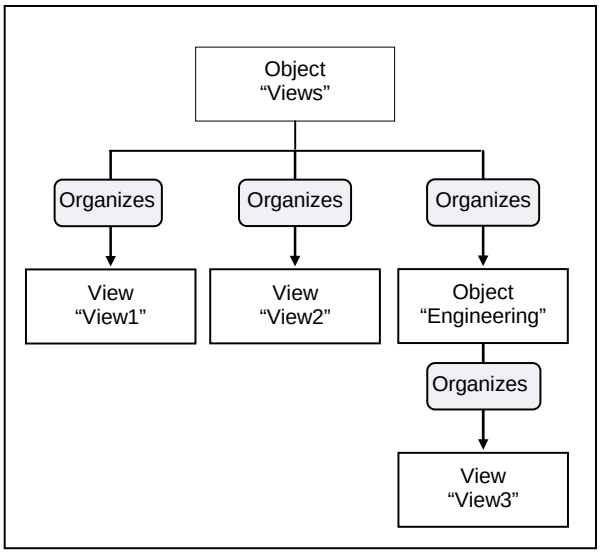


**Figure 2 – Views Organization**

The "*Views*" *Object* shall not reference any other *NodeClasses*. The "*Views*" *Object* is formally defined in Table 72.

**Table 72 – 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.

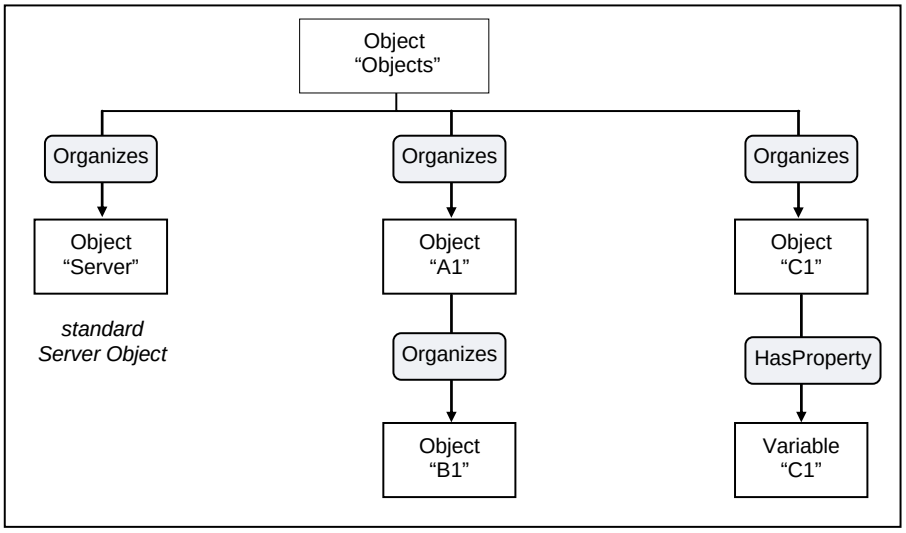


**Figure 3 – Objects Organization**

The "*Objects*" *Object* shall not reference any other *NodeClasses*. The "*Objects*" *Object* is formally defined in Table 73.

**Table 73 – 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 74.

**Table 74 – 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 8.2.12 |

### 8.2.6 ObjectTypes

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

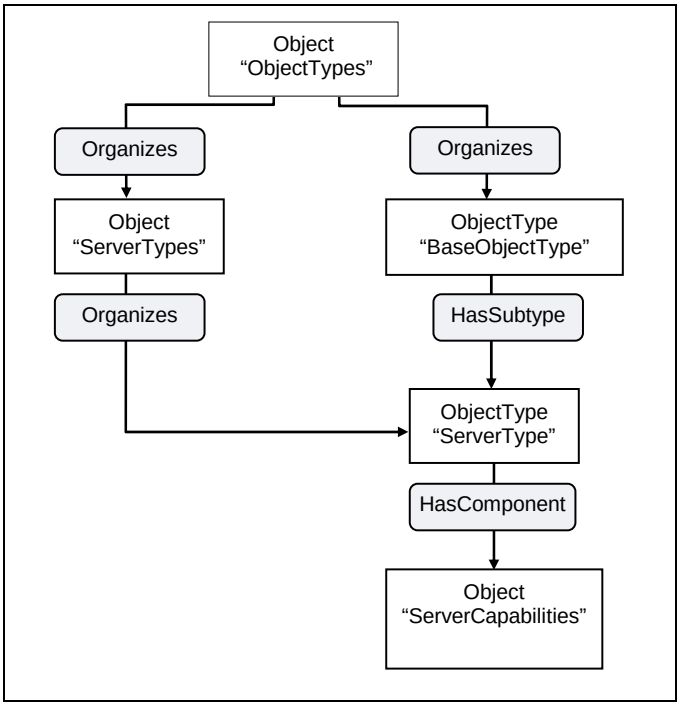


**Figure 4 – ObjectTypes Organization**

The intention of the "*ObjectTypes*" *Object* is that all *ObjectTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. However, this is not required and servers might not provide some of their *ObjectTypes* because they may be well-known in the industry, such as the *Server Type* 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 75.

**Table 75 – 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*.

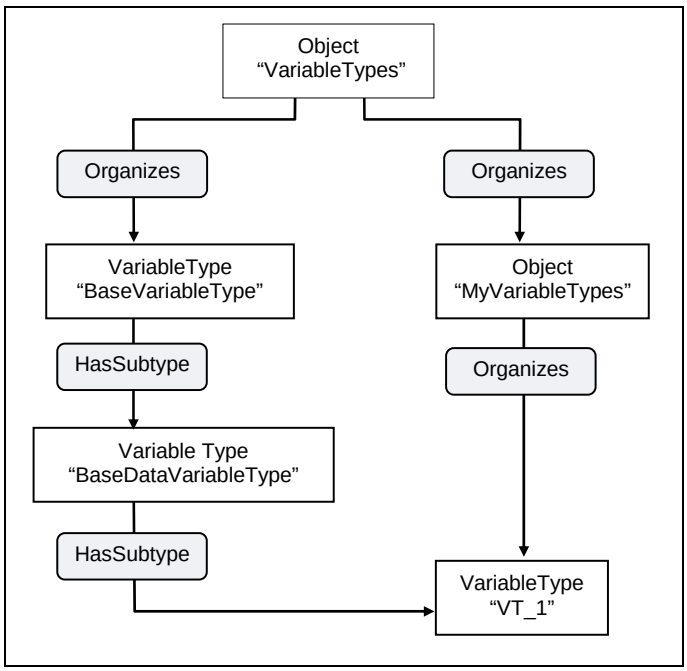


**Figure 5 – VariableTypes Organization**

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 76.

**Table 76 – 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 11 for a discussion of the standard *ReferenceTypes* that appear beneath the "*ReferenceTypes*" *Object*.

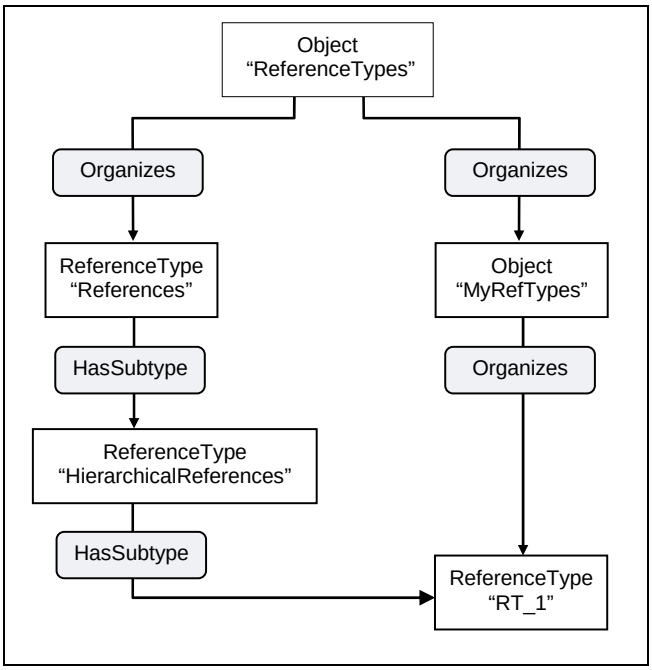


**Figure 6 – ReferenceType Definitions**

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 77.

**Table 77 – 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*.

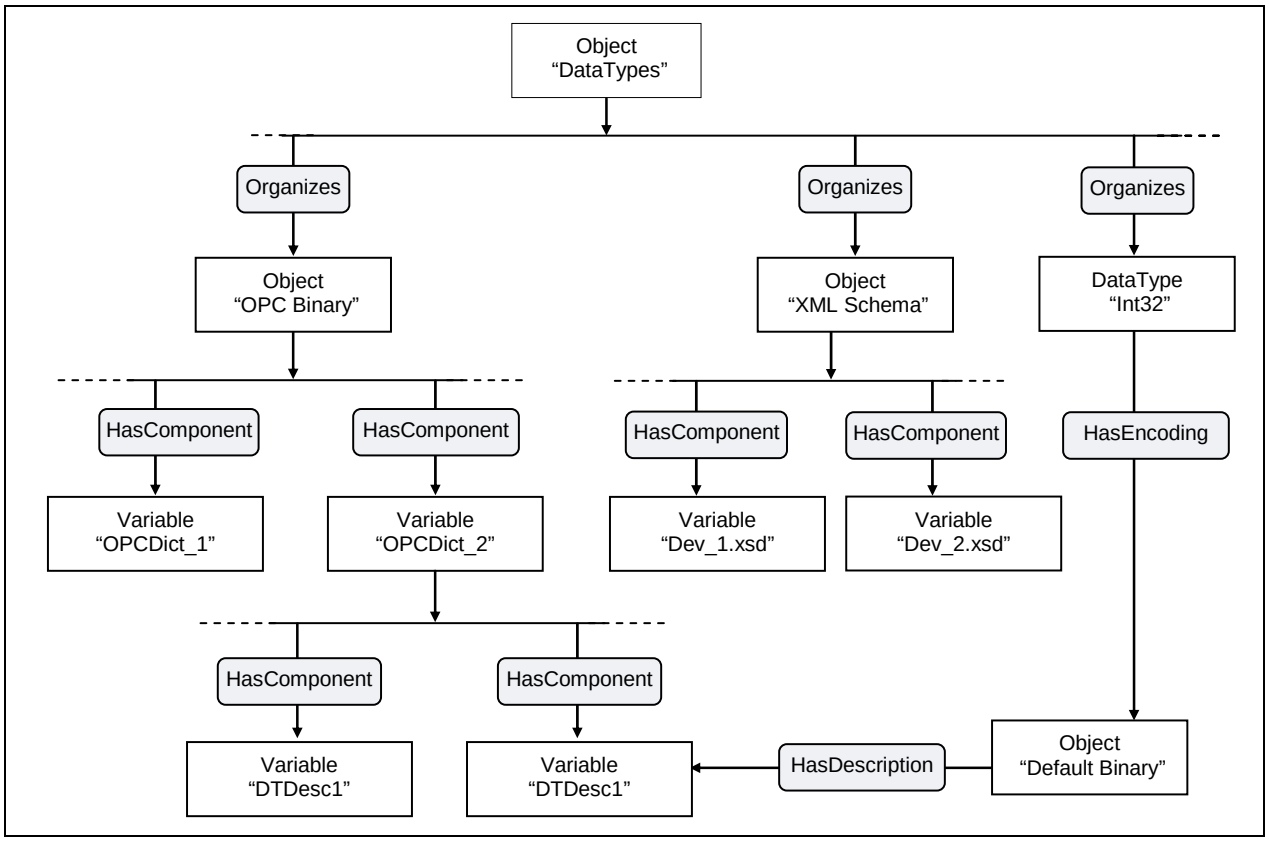


**Figure 7 – DataTypes Organization**

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*.

The “*DataTypes*” *Object* is formally defined in Table 78.

**Table 78 – 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 Part 3. OPC Binary uses XML to describe complex binary data values. The "*OPC Binary*" *Object* is formally defined in Table 79.

**Table 79 – 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 80.

**Table 80 – XML Schema Definition**

| **Attribute** | **Value** | | |
|---|---|---|---|
| BrowseName | XML Schema | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
| HasTypeDefinition | ObjectType | DataTypeSystemType | Defined in 6.8 |

### 8.2.12 EventTypes

This standard *Object Node* is the browse entry point for *EventType Nodes*. Figure 8 illustrates the structure beneath this *Node* showing some of the standard *EventTypes* defined in 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*.

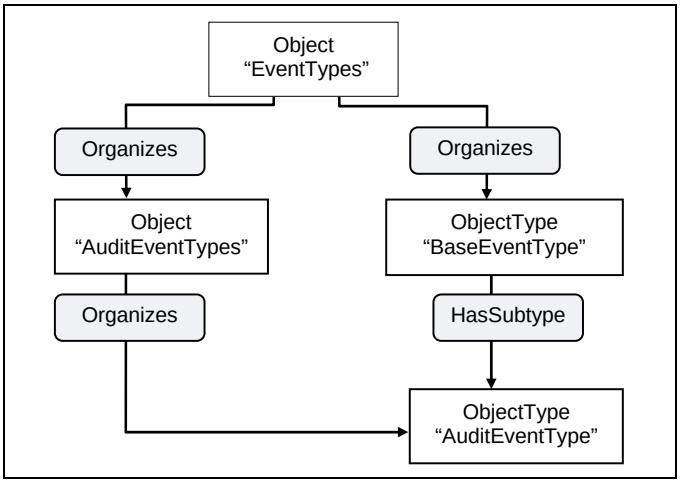


**Figure 8 – EventTypes Organization**

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 81.

**Table 81 – 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 9 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 9 is the security-related information per *Session*, which follows the same rules.

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

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

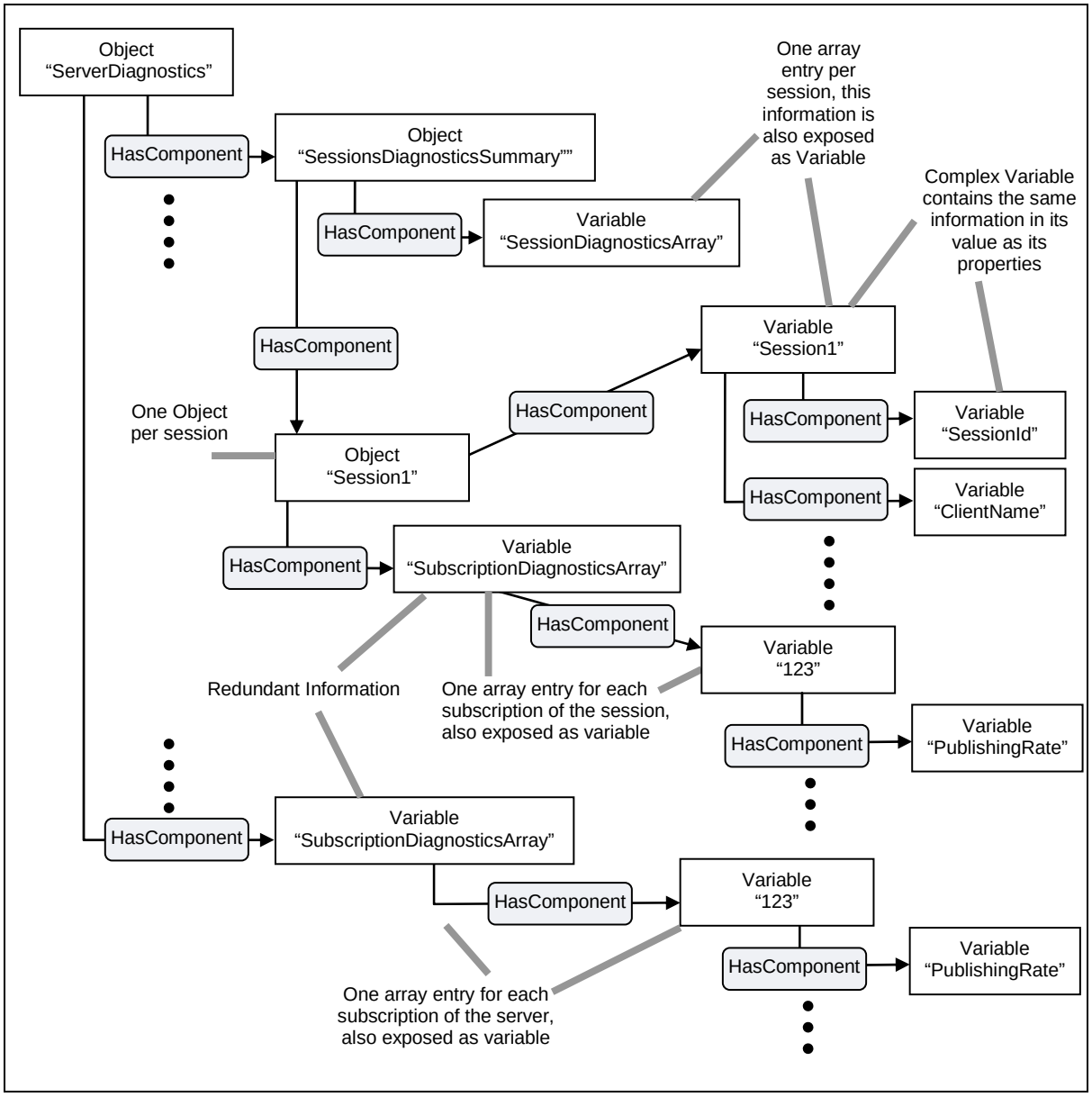


**Figure 9 – Excerpt of Diagnostic Information of the Server**

### 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 82. The *Server Object* serves as root notifier, i.e. 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 82 – Server Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | Server | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Modelling Rule** |
| HasTypeDefinition | Object Type | ServerType | Defined in 6.3.1 | | |
| HasProperty | Variable | ServerArray | String[] | PropertyType | Mandatory |
| HasProperty | Variable | NamespaceArray | String[] | PropertyType | Mandatory |
| HasComponent | Variable | ServerStatus[1] | ServerStatusDataType | ServerStatusType | Mandatory |
| HasProperty | Variable | ServiceLevel | Byte | PropertyType | Mandatory |
| HasComponent | Object | ServerCapabilities[1] | -- | ServerCapabilities | Mandatory |
| HasComponent | Object | ServerDiagnostics[1] | -- | ServerDiagnosticsType | Mandatory |
| HasComponent | Object | VendorServerInfo | -- | vendor-specific[2] | Mandatory |
| HasComponent | Object | ServerRedundancy[1] | -- | depends on supported redundancy[3] | Mandatory |

Notes –
1) 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.
2) Shall be the *VendorServerInfo ObjectType* or one of its subtypes
3) Shall be the *ServerRedundancyType* or one of its subtypes

## 8.4 ModellingRule Objects

### 8.4.1 ExposesItsArray

The *ModellingRule ExposesItsArray* is defined in Part 3. Its representation in the *AddressSpace* – the "*ExposesItsArray*" *Object* – is formally defined in Table 83.

**Table 83 – ExposesItsArray Definition**

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

### 8.4.2 Mandatory

The *ModellingRule Mandatory* is defined in Part 3. Its representation in the *AddressSpace* – the "*Mandatory*" *Object* – is formally defined in Table 84.

**Table 84 – Mandatory Definition**

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

### 8.4.3 Optional

The *ModellingRule Optional* is defined in Part 3. Its representation in the *AddressSpace* – the "*Optional*" *Object* – is formally defined in Table 85.

**Table 85 – Optional Definition**

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

## 9 Standard Methods

There are no core OPC UA *Methods* defined.

## 10 Standard Views

There are no core OPC UA *Views* defined.

## 11 Standard ReferenceTypes

### 11.1 References

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 86.

**Table 86 – References ReferenceType**

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

### 11.2 HierarchicalReferences

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 87.

**Table 87 – 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 11.15 |

### 11.3 NonHierarchicalReferences

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 88.

**Table 88 – 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 11.17 |
| HasSubtype | ReferenceType | HasModelParent | Defined in 11.19 |

### 11.4 HasChild

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 90.

**Table 89 – HasChild ReferenceType**

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

### 11.5 Aggregates

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 90.

**Table 90 – Aggregates ReferenceType**

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

### 11.6 Organizes

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 91.

**Table 91 – Organizes ReferenceType**

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

### 11.7 HasComponent

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 92.

**Table 92 – HasComponent ReferenceType**

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

### 11.8 HasOrderedComponent

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 93.

**Table 93 – HasOrderedComponent ReferenceType**

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

### 11.9 HasProperty

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 94.

**Table 94 – HasProperty ReferenceType**

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

### 11.10 HasSubtype

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 95.

**Table 95 – HasSubtype ReferenceType**

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

### 11.11 HasModellingRule

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 96.

**Table 96 – HasModellingRule ReferenceType**

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

### 11.12 HasTypeDefinition

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 97.

**Table 97 – HasTypeDefinition ReferenceType**

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

### 11.13 HasEncoding

This standard *ReferenceType* is defined in Part 3. Its representation in the *AddressSpace* is specified in Table 97.

**Table 98 – 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 Part 3. Its representation in the *AddressSpace* is specified in Table 97.

**Table 99 – HasDescription ReferenceType**

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

### 11.15 HasEventSource

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

**Table 100 – HasEventSource ReferenceType**

| **Attributes** | **Value** | | |
|---|---|---|---|
| BrowseName | HasEventSource | | |
| InverseName | EventSourceOf | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
| HasSubtype | ReferenceType | HasNotifier | Defined in 11.16 |

### 11.16 HasNotifier

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

**Table 101 – HasNotifier ReferenceType**

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

### 11.17 GeneratesEvent

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

**Table 102 – GeneratesEvent ReferenceType**

| **Attributes** | **Value** | | |
|---|---|---|---|
| BrowseName | GeneratesEvent | | |
| InverseName | GeneratedBy | | |
| Symmetric | False | | |
| IsAbstract | False | | |
| **References** | **NodeClass** | **BrowseName** | **Comment** |
| HasSubtype | ReferenceType | AlwaysGeneratesEvent | Defined in 11.18 |

### 11.18 AlwaysGeneratesEvent

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

**Table 103 – AlwaysGeneratesEvent ReferenceType**

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

### 11.19 HasModelParent

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

**Table 104 – HasModelParent ReferenceType**

| **Attributes** | **Value** | | |
|---|---|---|---|
| BrowseName | HasModelParent | | |
| InverseName | IsModelParentOf | | |
| 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. The *DataTypeEncodings*, the *DataTypeDescriptions* and the *DataTypeDictionaries* of the structured *DataTypes* and the *References* to them are specified in Part 6 as well as the *EnumStrings Properties* for enumerated *DataTypes*.

### 12.2 DataTypes defined in Part 3

Part 3 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 105.

**Table 105 – Part 3 DataType Definitions**

| BrowseName |
|---|
| BaseDataType |
| Argument |
| Boolean |
| Byte |
| ByteString |
| DateTime |
| Double |
| Duration |
| Enumeration |
| Float |
| Guid |
| IdType |
| SByte |
| Integer |
| Int16 |
| Int32 |
| Int64 |
| Image |
| ImageBMP |
| ImageGIF |
| ImageJPG |
| ImagePNG |
| LocaleId |
| LocalizedText |
| NamingRuleType |
| NodeClass |
| NodeId |
| Number |
| QualifiedName |
| String |
| Structure |
| Time |
| UInteger |
| UInt16 |
| UInt32 |
| UInt64 |
| UtcTime |
| XmlElement |
| TimeZoneDataType |

Of the *DataTypes* defined in Table 105 only some are the sources of *References* as defined in the following tables.

The *References* of the *BaseDataType* are defined in Table 106.

**Table 106 – 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 107.

**Table 107 – 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 |

The *References* of *Enumeration* are defined in Table 108.

**Table 108 – 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 109.

**Table 109 – ByteString Definition**

| Attributes | Value | | |
|---|---|---|---|
| BrowseName | ByteString | | |
| IsAbstract | TRUE | | |
| **References** | **NodeClass** | **BrowseName** | **IsAbstract** |
| HasSubtype | DataType | Image | TRUE |

The *References* of *Number* are defined in Table 110.

**Table 110 – 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 |

The *References* of *Double* are defined in Table 111.

**Table 111 – Double Definition**

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

The *References* of *Integer* are defined in Table 112.

**Table 112 – 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 113.

**Table 113 – DateTime Definition**

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

The *References* of *String* are defined in Table 112.

**Table 114 – String Definition**

| **Attributes** | **Value** | | |
|---|---|---|---|
| BrowseName | String | | |
| IsAbstract | FALSE | | |
| **References** | **NodeClass** | **BrowseName** | **IsAbstract** |
| HasSubtype | DataType | LocaleId | FALSE |
| HasSubtype | DataType | NumericRange | FALSE |

The *References* of UInteger are defined in Table 115.

**Table 115 – 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 116.

**Table 116 – 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 |

## 12.3 DataTypes defined in Part 4

Part 4 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 117.

**Table 117 – Part 4 DataType Definitions**

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

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

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

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

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

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

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

**Table 118 – 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 119.

**Table 119 – 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 120.

**Table 120 – 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 121.

**Table 121 – RedundancySupport Values**

| Value | Description |
|---|---|
| NONE_0 | None means that there is no redundancy support. |
| COLD_1 | Cold means that the redundant servers are operational, but do not have any subscriptions defined and do not accept requests to create one. |
| WARM_2 | Warm means that the redundant servers have redundant subscriptions, but with sampling disabled. |
| HOT_3 | Hot means that the redundant servers have redundant subscriptions with sampling enabled, but not reporting. |
| TRANSPARENT_4 | Transparent means that the server supports transparent redundancy as defined in Part 1. |

See Part 1 for a more detailed description of the different values.

Its representation in the *AddressSpace* is defined in Table 122.

**Table 122 – 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 123.

**Table 123 – 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 or is in the process of shutting down. Depending on the implementation, this might or might not be visible to clients. |
| 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. |

Its representation in the *AddressSpace* is defined in Table 124.

**Table 124 – 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 125.

**Table 125 – 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 126.

**Table 126 – 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 127.

**Table 127 – SamplingIntervalDiagnosticsDataType Structure**

| Name | Type | Description |
|---|---|---|
| SamplingIntervalDiagnosticsDataType | structure | |
| samplingRate | Duration | The sampling rate 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 128.

**Table 128 – SamplingIntervalDiagnosticsDataType Definition**

| Attributes | Value |
|---|---|
| BrowseName | SamplingIntervalDiagnosticsDataType |

### 12.9 ServerDiagnosticsSummaryDataType

This structure contains diagnostic summary information for the *Server*. Its elements are defined in Table 129.

**Table 129 – 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 that were rejected due to security constraints since the server was started (or restarted). |
| rejectedSessionCount | UInt32 | The number of client session establishment requests 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). |
| samplingRateCount | UInt32 | The number of sampling rates currently used in the server. |
| publishingIntervalCount | UInt32 | The number of publishing intervals currently supported in the server. |
| currentSubscriptionCount | UInt32 | The number of subscriptions currently established in the server. |
| cumulatedSubscriptionCount | UInt32 | The cumulative number of subscriptions that have been established in the server since the server was started (or restarted). This includes the *currentSubscriptionCount*. |
| securityRejectedRequestsCount | UInt32 | The number of requests that were rejected due to security constraints since the server was started (or restarted). The requests include all *Services* defined in Part 4, also requests to create sessions. |
| rejectedRequestsCount | UInt32 | The number of requests that were rejected since the server was started (or restarted). The requests include all *Services* defined in Part 4, also requests to create sessions. This number includes the *securityRejectedRequestsCount*. |

Its representation in the *AddressSpace* is defined in Table 130.

**Table 130 – 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 131.

**Table 131 – 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. |
| shutdownReason | LocalizedText | An optional localized text indicating the reason for the shutdown. The value is only relevant once the state changes into SHUTDOWN. |

Its representation in the *AddressSpace* is defined in Table 132.

**Table 132 – ServerStatusDataType Definition**

| Attributes | Value |
|---|---|
| BrowseName | ServerStatusDataType |

## 12.11 SessionDiagnosticsDataType

This structure contains diagnostic information about client *Sessions*. Its elements are defined in Table 133. 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 133 – 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. |
| clientDesription | 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. |

| | | |
|---|---|---|
| totalRequestsCount | ServiceCounter DataType | Counter of all*Services*, identifying the number of received requests of any *Services* on the session. |
| unauthorizedRequestsCount | UInt32 | Counter of all Serices, 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 CreateMonitoredItem *Service*, identifying the number of received requests of this *Service* on the session. |
| modifyMonitoredItemsCount | ServiceCounter DataType | Counter of the ModifyMonitoredItem *Service*, identifying the number of received requests of this *Service* on the session. |
| setMonitoringModeCount | ServiceCounter DataType | Counter of the SetMonitoringMode *Service*, identifying the number of received requests of this *Service* on the session. |
| setTriggeringCount | ServiceCounter DataType | Counter of the SetTriggering *Service*, identifying the number of received requests of this *Service* on the session. |
| deleteMonitoredItemsCount | ServiceCounter DataType | Counter of the DeleteMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session. |
| createSubscriptionCount | ServiceCounter DataType | Counter of the CreateSubscription *Service*, identifying the number of received requests of this *Service* on the session. |
| modifySubscriptionCount | ServiceCounter DataType | Counter of the ModifySubscription *Service*, identifying the number of received requests of this *Service* on the session. |
| setPublishingModeCount | ServiceCounter DataType | Counter of the SetPublishingMode *Service*, identifying the number of received requests of this *Service* on the session. |
| publishCount | ServiceCounter DataType | Counter of the Publish *Service*, identifying the number of received requests of this *Service* on the session. |
| republishCount | ServiceCounter DataType | Counter of the Republish *Service*, identifying the number of received requests of this *Service* on the session. |
| transferSubscriptionsCount | ServiceCounter DataType | Counter of the TransferSubscriptions *Service*, identifying the number of received requests of this *Service* on the session. |
| deleteSubscriptionsCount | ServiceCounter DataType | Counter of the DeleteSubscriptions *Service*, identifying the number of received requests of this *Service* on the session. |
| addNodesCount | ServiceCounter DataType | Counter of the AddNodes *Service*, identifying the number of received requests of this *Service* on the session. |
| addReferencesCount | ServiceCounter DataType | Counter of the AddReferences *Service*, identifying the number of received requests of this *Service* on the session. |
| deleteNodesCount | ServiceCounter DataType | Counter of the DeleteNodes *Service*, identifying the number of received requests of this *Service* on the session. |
| deleteReferencesCount | ServiceCounter DataType | Counter of the DeleteReferences *Service*, identifying the number of received requests of this *Service* on the session. |
| browseCount | ServiceCounter DataType | Counter of the Browse *Service*, identifying the number of received requests of this *Service* on the session. |
| browseNextCount | ServiceCounter DataType | Counter of the BrowseNext *Service*, identifying the number of received requests of this *Service* on the session. |
| translateBrowsePathsToNodeIdsCount | ServiceCounter DataType | Counter of the TranslateBrowsePathsToNodeIds *Service*, identifying the number of received requests of this *Service* on the session. |
| queryFirstCount | ServiceCounter DataType | Counter of the QueryFirst *Service*, identifying the number of received requests of this *Service* on the session. |
| queryNextCount | ServiceCounter DataType | Counter of the QueryNext *Service*, identifying the number of received requests of this *Service* on the session. |
| registerNodesCount | ServiceCounter DataType | Counter of the RegisterNodesCount *Service*, identifying the number of received requests of this *Service* on the session. |
| unregisterNodesCount | ServiceCounter DataType | Counter of the UnregisterNodesCount*Service*, identifying the number of received requests of this *Service* on the session. |

Its representation in the *AddressSpace* is defined in Table 134.

**Table 134 – 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 135. Because this information is security-related, it should not be made accessible to all users, but only to authorised users.

**Table 135 – 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 by 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). |
| encoding | String | Which encoding is used on the wire, e.g. XML or UA Binary. |
| transportProtocol | String | Which transport protocol is used, e.g. TCP or HTTP. |
| 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 136.

**Table 136 – SessionSecurityDiagnosticsDataType Definition**

| Attributes | Value |
|---|---|
| BrowseName | SessionSecurityDiagnosticsDataType |

### 12.13 ServiceCounterDataType

This structure contains diagnostic information about subscriptions. Its elements are defined in Table 137.

**Table 137 – 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 138.

**Table 138 – 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 139. Whether the DiagnosticInformation is returned depends on the setting of the Service calls.

**Table 139 – 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 140.

**Table 140 – StatusResult Definition**

| Attributes | Value |
|---|---|
| BrowseName | StatusResult |

### 12.15 SubscriptionDiagnosticsDataType

This structure contains diagnostic information about subscriptions. Its elements are defined in Table 141.

**Table 141 – 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 |
| 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 142.

**Table 142 – 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 143.

**Table 143 – 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>NodeId of the Node that was changed. The client should assume that the affected Node has been created or deleted, had a Reference added or deleted, or the DataType has changed as described by the verb.</td></tr>
<tr><td>affectedType</td><td>NodeId</td><td>If the affected Node was an Object or Variable, affectedType contains the NodeId of the TypeDefinitionNode of the affected Node. Otherwise it is set to null.</td></tr>
<tr><td>verb</td><td>Byte</td><td>Describes the changes happening to the affected Node.<br>The verb is an 8-bit unsigned integer used as bit mask with the structure defined in the following table:
<table>
<tr><th>Field</th><th>Bit</th><th>Description</th></tr>
<tr><td>NodeAdded</td><td>0</td><td>Indicates the affected Node has been added.</td></tr>
<tr><td>NodeDeleted</td><td>1</td><td>Indicates the affected Node has been deleted.</td></tr>
<tr><td>ReferenceAdded</td><td>2</td><td>Indicates a Reference has been added. The affected Node may be either a SourceNode or TargetNode. Note that an added bidirectional Reference is reflected by two ChangeStructures.</td></tr>
<tr><td>ReferenceDeleted</td><td>3</td><td>Indicates a Reference has been deleted. The affected Node may be either a SourceNode or TargetNode. Note that a deleted bidirectional Reference is reflected by two ChangeStructures.</td></tr>
<tr><td>DataTypeChanged</td><td>4</td><td>This verb may be used only for affected Nodes that are Variables or VariableTypes. It indicates that the DataType Attribute has changed.</td></tr>
<tr><td>Reserved</td><td>5:7</td><td>Reserved for future use. Shall always be zero.</td></tr>
</table>
A verb may identify several changes on the affected Node at once. This feature should be used if event compression is used (see Part 3 for details).<br>Note that all verbs shall always be considered in the context where the ModelChangeStructureDataType is used. A NodeDeleted may indicate that a Node was removed from a view but still exists in other Views.</td></tr>
</table>

Its representation in the *AddressSpace* is defined in Table 132.

**Table 144 – 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 145.

**Table 145 – 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 132.

**Table 146 – SemanticChangeStructureDataType Definition**

| Attributes | Value |
|---|---|
| BrowseName | SemanticChangeStructureDataType |

# Annex A (informative): Design decisions when modelling the server information

## A.1 Overview

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

This Appendix 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 Appendix A of Part 3.

This Appendix is informative, that is each *Server* vendor can model its data in the appropriate way that fits its needs.

The following subclauses describe the design decisions made while modelling the *Server Object*. General *DataTypes*, *VariableTypes* and *ObjectTypes* such as the *EventTypes* described in this Part 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 part and the *NodeId* is based on the symbolic name in Part 6. Nevertheless, aggregating *Servers* may want to expose the *Server Objects* of the OPC UA *Servers* they are aggregating in their *AddressSpace*. Therefore, it is very helpful to have a type definition for the *Server Object*. The *Server Object* is an *Object*, because it groups a set of *Variables* and *Objects* containing information about the *Server*. The *ServerType* is a complex *ObjectType*, because the basic structure of the *Server Object* should be well-defined. However, the *Server Object* can be extended by adding *Variables* and *Objects* in an appropriate structure of the *Server Object* or its containing *Objects*.

## A.3 Typed complex Objects beneath the Server Object

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

## A.4 Properties vs. DataVariables

Since the general description in Part 3 about the semantic difference between *Properties* and *DataVariables* are not applicable for the information provided about the *Server* the rules described in A.4.2 of Part 3 are used.

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

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

## A.5 Complex Variables using complex DataTypes

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

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

## A.6 Complex Variables having an array

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

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

## A.7 Redundant information

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

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

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

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

## A.8 Usage of the BaseDataVariableType

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

## A.9 Subtyping

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

Subtyping is also used as an extensibility mechanism (see next Clause).

## A.10 Extensibility mechanism

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

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

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

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

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

# Annex B (normative): StateMachines

## B.1 Scope

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

This Appendix is normative, i.e. the types defined in the Appendix 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.

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

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

## B.2 Examples of finite state machines

### B.2.1 Simple state machine

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

Figure 10 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, e.g. the call of “Method1” for the transition from “State1” to “State2”. The effect (or action) of the transition can also be specified, e.g. 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”.

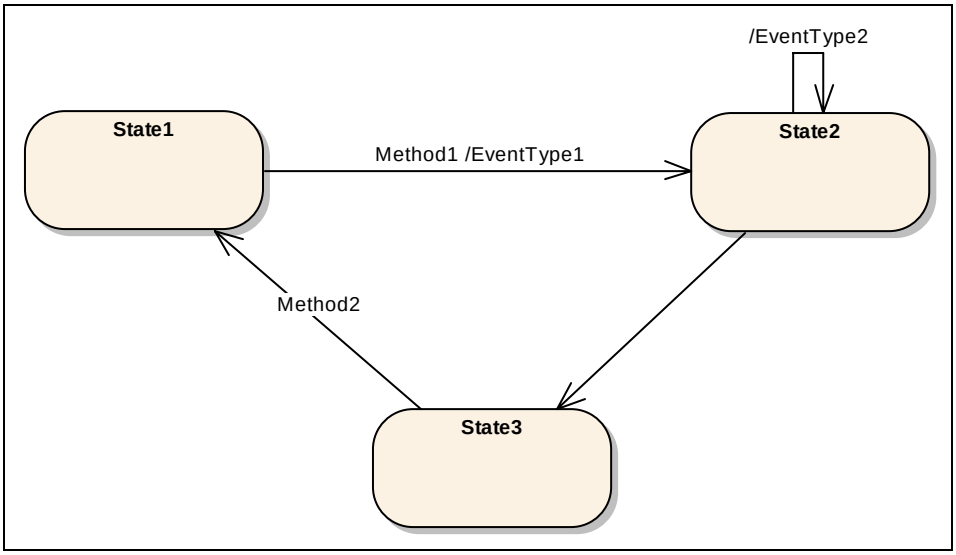


**Figure 10 – Example of a simple state machine**

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

### B.2.2 State machine containing substates

Figure 11 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 distinguish to be in the sub-states “State7” or “State8”. Sub-state-machines can be nested, i.e. “State7” could be another sub-state-machine.

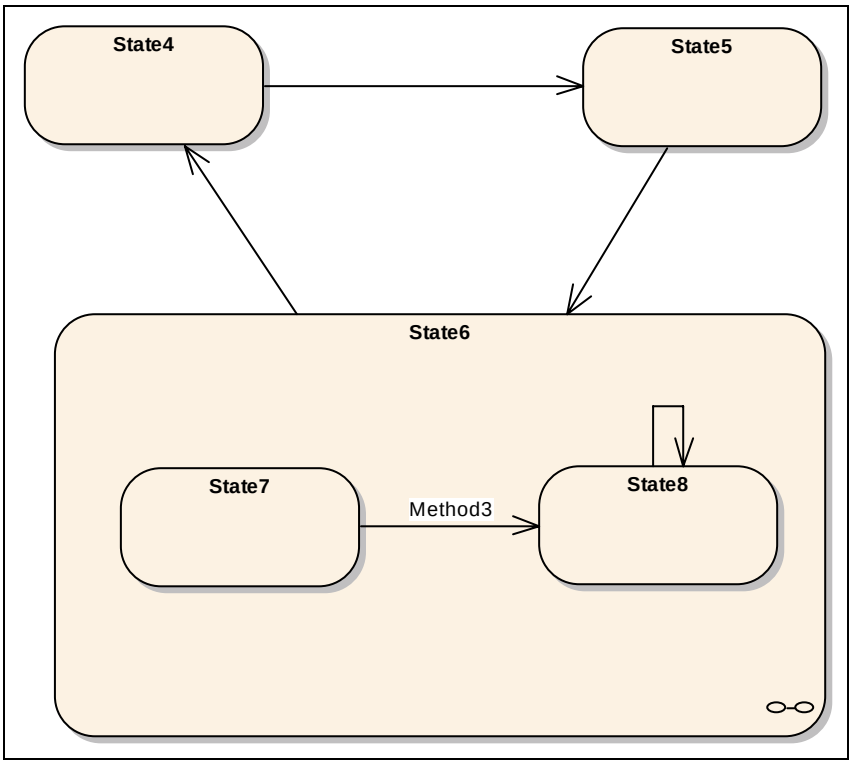


**Figure 11 – Example of a state machine having a sub-machine**

## B.3 Definition of state machine

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

For the state machines defined in this Appendix 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*. The *StateMachineType* specifies a description of the state machine – its states, transitions, etc. – whereas the *StateMachine* is an instance of the *StateMachineType* and only contains the current state.

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

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

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

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

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

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

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

## B.4 Representation of state machines in the AddressSpace

### B.4.1 Overview

The types defined in this Appendix are illustrated in Figure 12. The *MyFiniteStateMachineType* is a minimal example which illustrates how these *Types* can be used to describe a *StateMachine*. See Part 9 and Part 10 for additional examples of *StateMachines*.

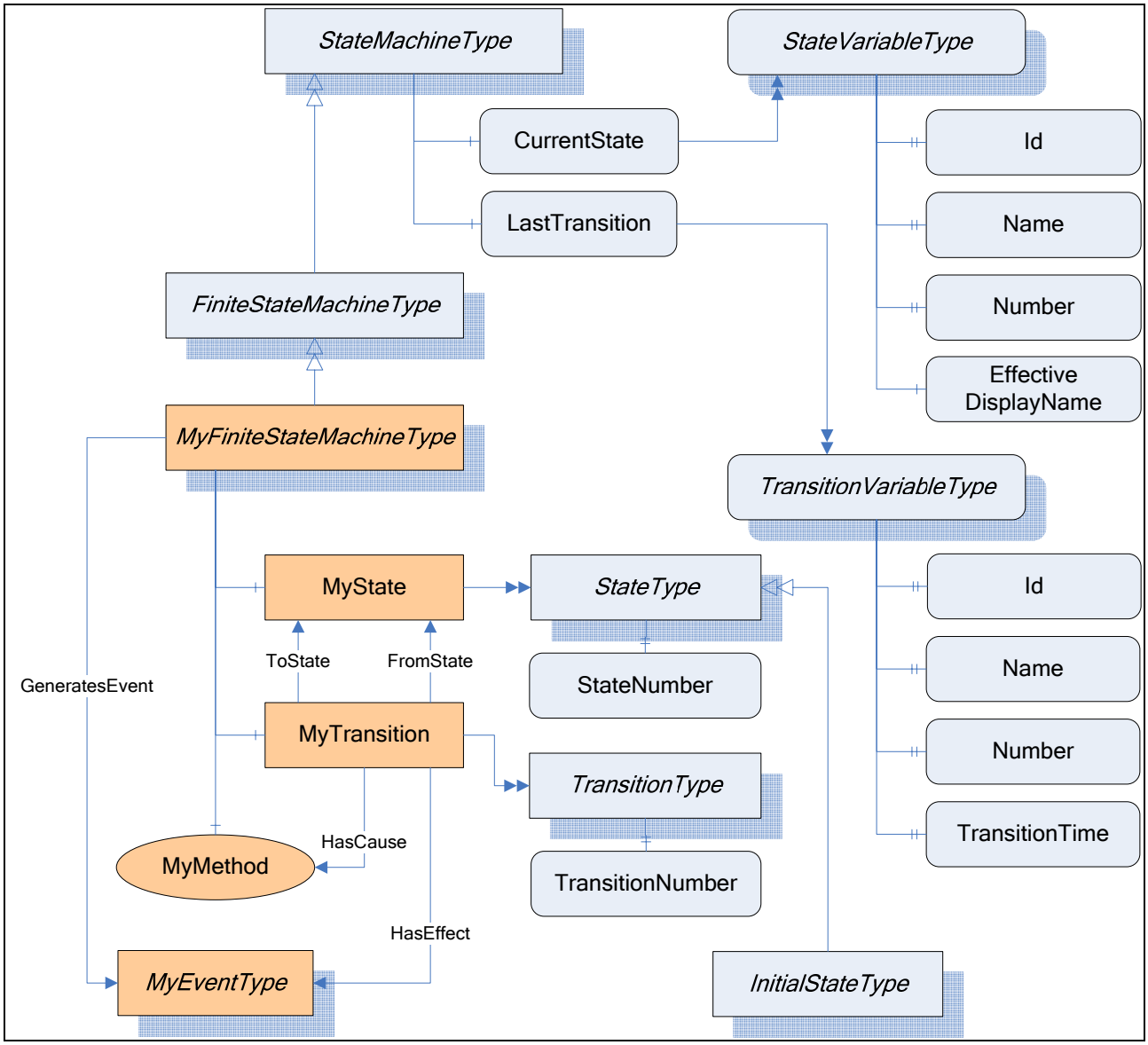


**Figure 12 – The StateMachine Information Model**

### B.4.2 StateMachineType

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

Subtypes may add *Methods* which affect the state of the machine. The *Executable Attribute* is used to indicate whether the *Method* is valid given the current state of the machine. The generation of *AuditEvents* for *Methods* is defined in Part 4. A *StateMachine* may not be active. In this case, the *CurrentState* and *LastTransition Variables* shall have a status equal to *Bad_StateNotActive* (see Table 163).

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 147.

**Table 147 – StateMachineType Definition**

| **Attribute** | **Value** | | | | |
|---|---|---|---|---|---|
| BrowseName | StateMachineType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Modelling Rule** |
| Subtype of the BaseObjectType defined in 6.2.<br>Note that a *Reference* to this sub type 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 147.

**Table 148 – 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.<br>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 149.

**Table 149 – 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.<br>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 |

*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.

### B.4.5 FiniteStateMachineType

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

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.

The *FiniteStateMachineType* is formally defined in Table 150.

**Table 150 – 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 |

### 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 151.

**Table 151 – 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 |
| HasProperty | Variable | Name | QualifiedName | PropertyType | Optional |
| HasProperty | Variable | Number | UInt32 | PropertyType | Optional |
| HasProperty | Variable | EffectiveDisplayName | LocalizedText | PropertyType | Optional |

*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.*

*Name* inherited from *StateVariableType* shall be the *BrowseName* of one of the *State Objects* of the *FiniteStateMachineType*.

*Number* inherited from *StateVariableType* 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 *TransitionVariableType* is formally defined in Table 152.

**Table 152 – 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 *BaseDataVariableType* defined in 7.4.<br>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*.

*Name* inherited from the *TransitionVariableType* shall be the *BrowseName* of one of the *Transition Objects* of the *FiniteStateMachineType*.

*Number* inherited from the *TransitionVariableType* 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 153.

**Table 153 – 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 153. An *Object* of the *InitialStateType* represents the *State* that a *FiniteStateMachine* enters when it 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 13. 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.

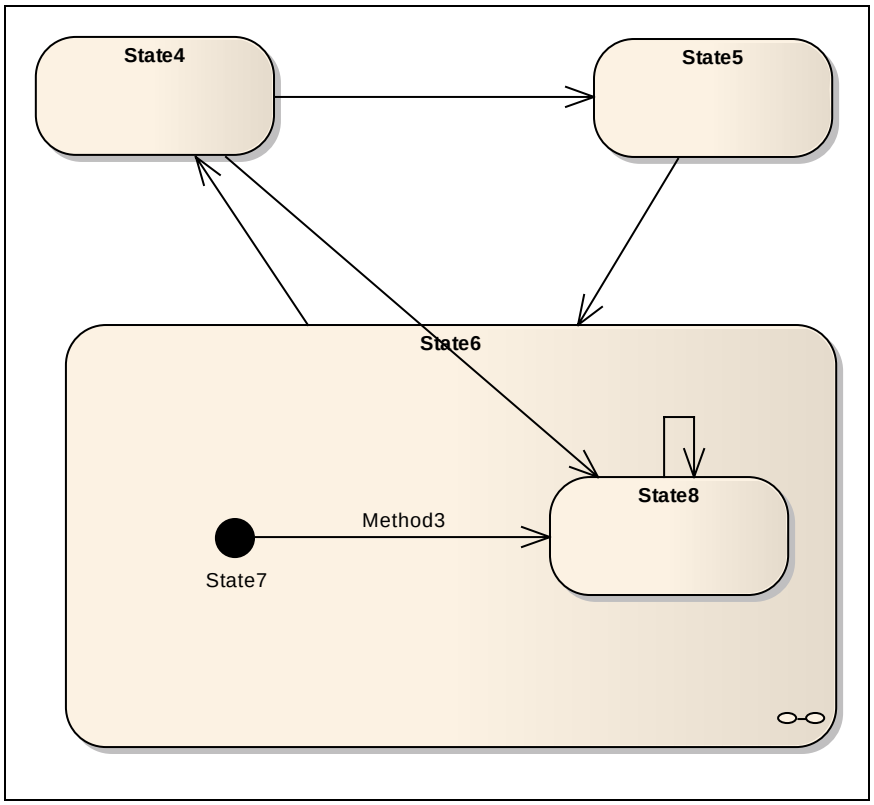


**Figure 13 – Example of an initial State in a sub-machine**

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 154 – 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 155.

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 155 – 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 156.

**Table 156 – 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 157.

**Table 157 – 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 form a *Transition* to something that causes the *Transition*. In this Appendix we only define *Methods* as *Causes*. However, the *ReferenceType* is not restricted to point to *Methods*.

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 158.

**Table 158 – HasCause ReferenceType**

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

### B.4.14 HasEffect

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

The semantic of this *ReferenceType* is to point form a *Transition* to something that will be effected when the *Transition* is triggered. In this Appendix 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 159.

**Table 159 – 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*, i.e. 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 160.

**Table 160 – 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 161.

**Table 161 - 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 were the *Transition* occurs. If the *Transition* occurs in a *SubStateMachine*, then the *NodeId* of the *SubStateMachine* has to be used. If the Transition occurs between a *StateMachine* and a *SubStateMachine*, then the *NodeId* of the *StateMachine* has to be used, independent of the direction of the *Transition*.

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

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

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

### B.4.17 AuditUpdateStateEventType

The *AuditUpdateStateEventType* is a subtype of the *AuditUpdateMethodEventType*. It can be used to generate an *Event* identifying that a *Transition* of a *StateMachine* was triggered. It is formally defined in Table 162.

**Table 162 - 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 were the *State* changed. If the *State* changed in a *SubStateMachine*, then the *NodeId* of the *SubStateMachine* has to be used.

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

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

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

### B.4.18 Special Restrictions on subtyping StateMachines

In general, all rules on subtyping apply for *StateMachine* types as well. Some additional rules apply for *StateMachine* types. If a StateMachine type is not abstract, subtypes of it shall not change the behaviour of it. That means, that in that 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 163 specific *StatusCodes* used for *StateMachines* are defined.

**Table 163 – 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

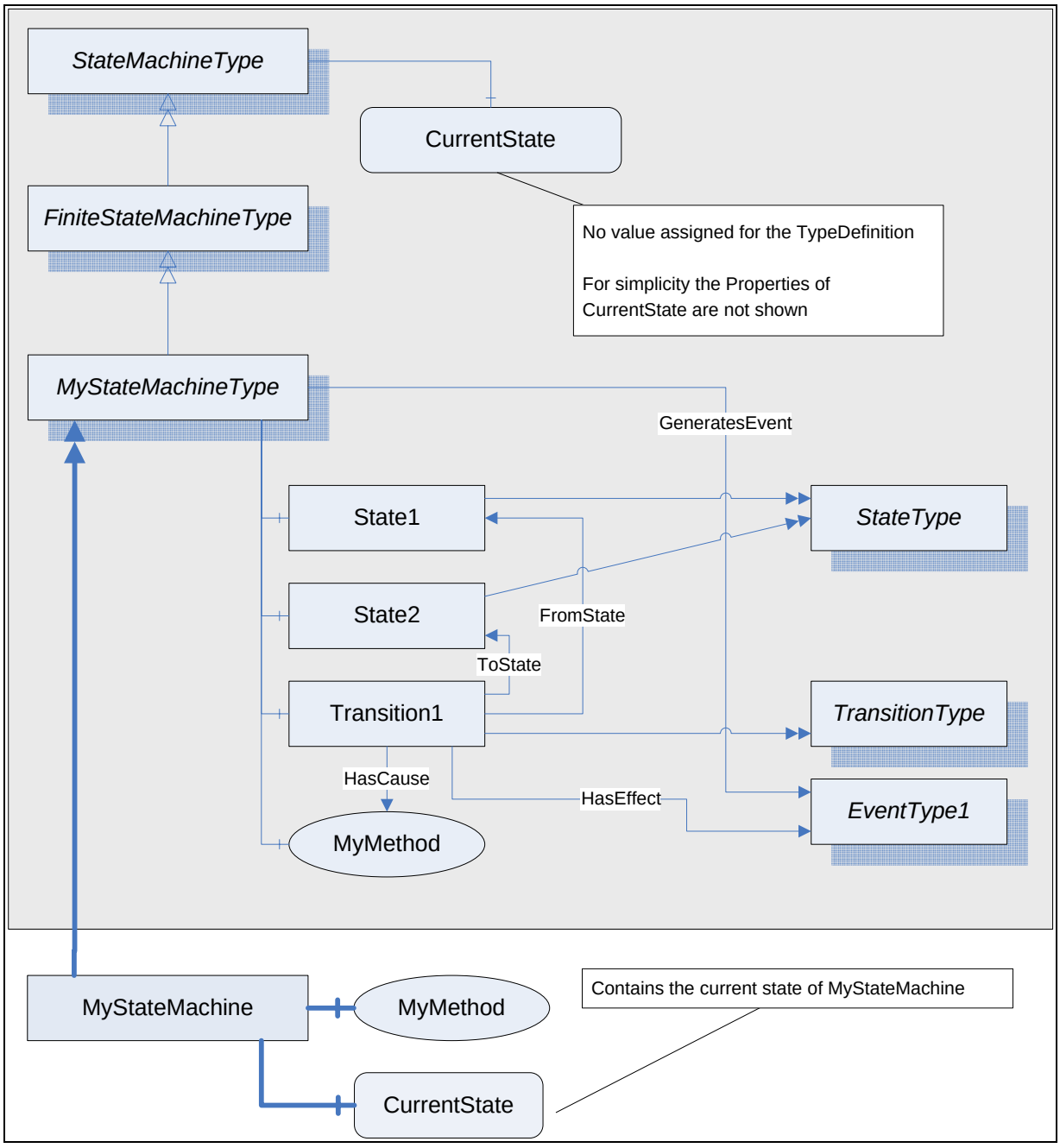


**Figure 14 – Example of a StateMachineType using inheritance**

In Figure 14 an example of a *StateMachine* is given using the Notation defined in the Appendix of Part 3. First, a new *StateMachineType* is defined, called “MyStateMachineType”, inheriting from the base *FiniteStateMachineType*. It contains two *States*, “State1” and “State2” and a *Transition* “Transition1” between them. The *Transition* points to a *Method* “MyMethod” as the *Cause* of the *Transition* and an *EventType* “EventType1” as the *Effect* of the *Transition*.

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

### B.5.2 StateMachineType with a sub-machine using inheritance

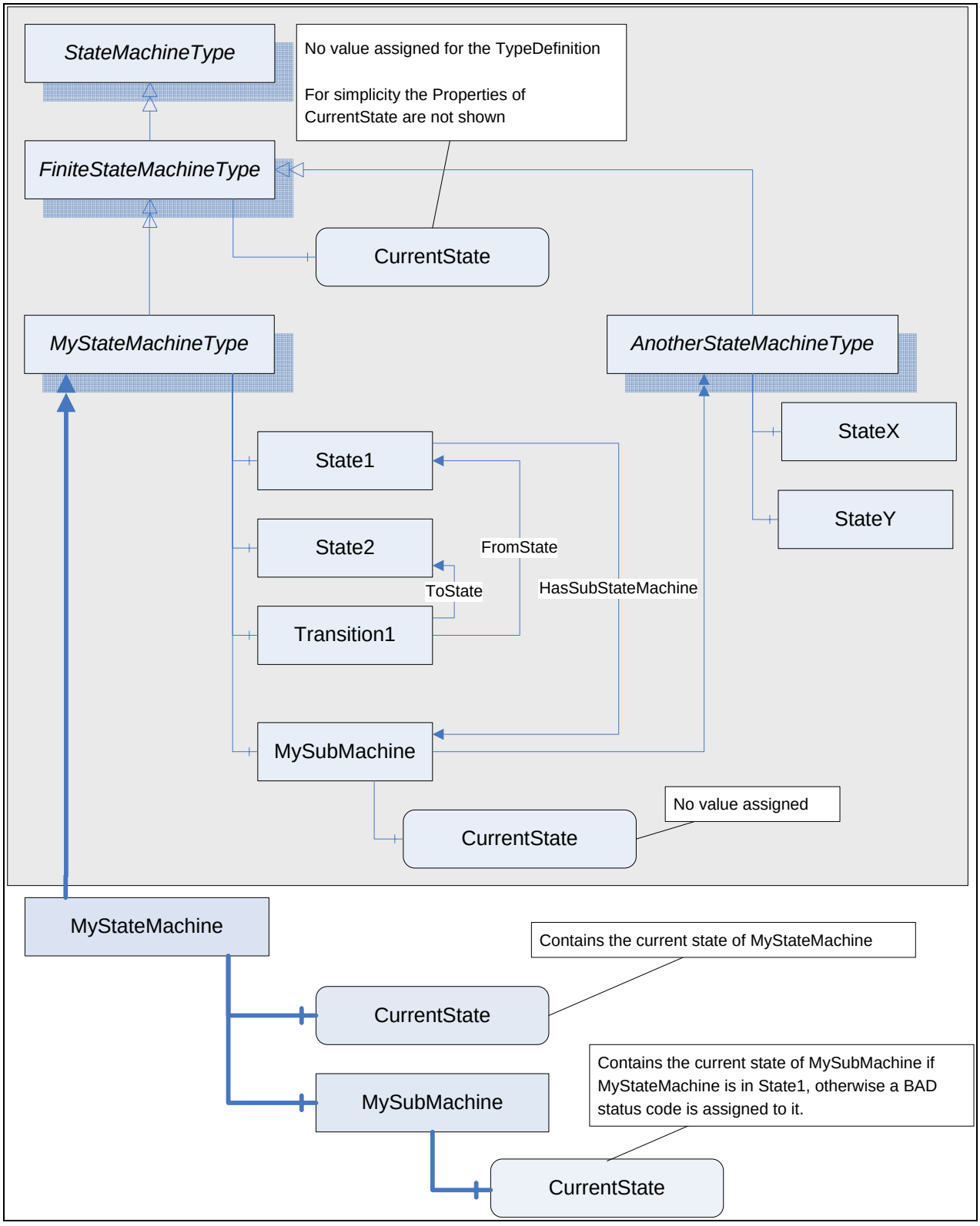


**Figure 15 – Example of a StateMachineType with a SubStateMachine using inheritance**

Figure 15 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

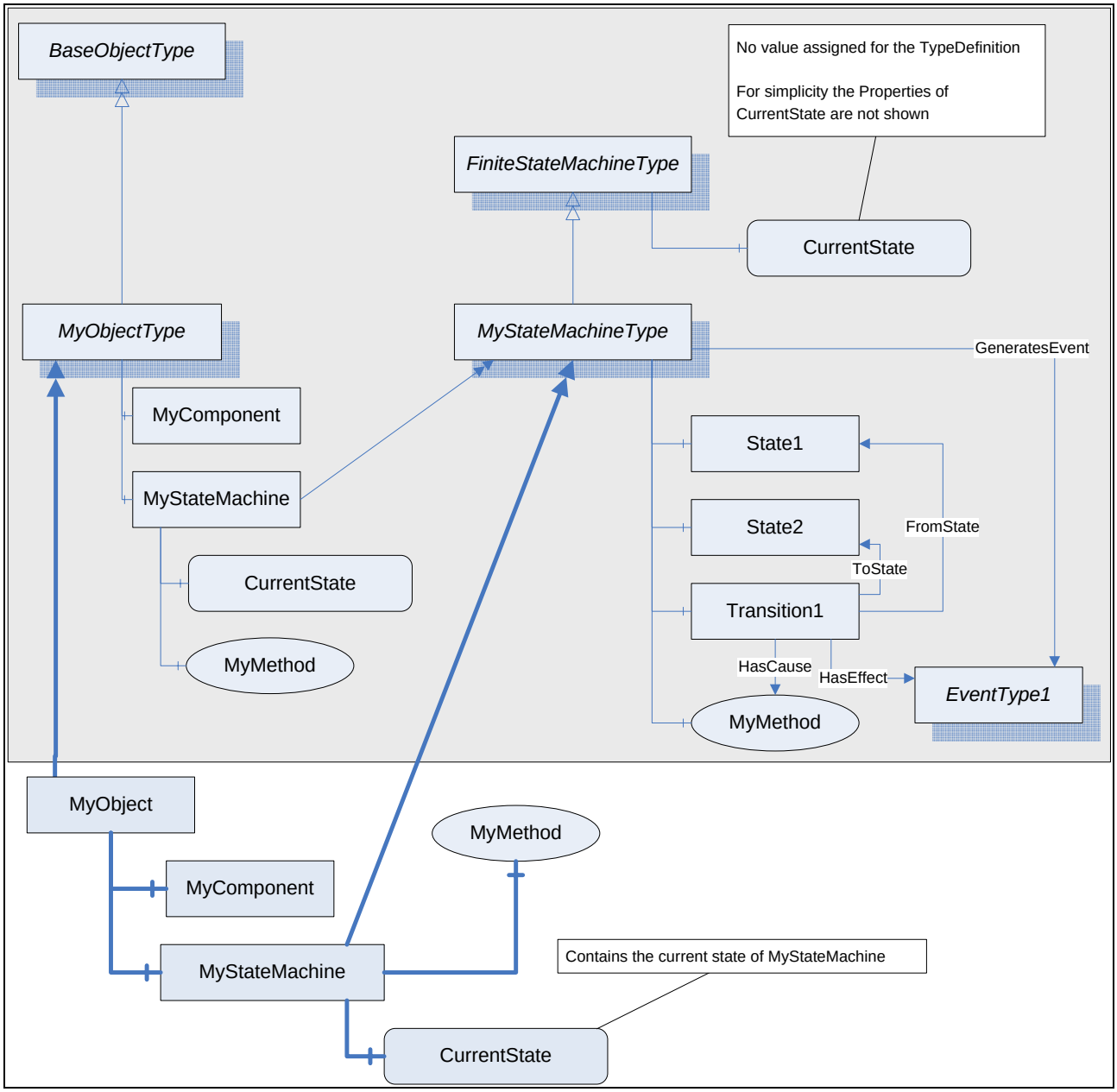


**Figure 16 – Example of a StateMachineType using containment**

Figure 16 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 is 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 Transitions to SubStateMachines

The *StateMachines* shown so far only had *Transitions* between *States* on the same level, i.e. 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, i.e. only pointing from the *Transition* to the *State* and not have the other direction browsable. 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 17 gives an example of a state machine having a transition from a sub-state to a state.

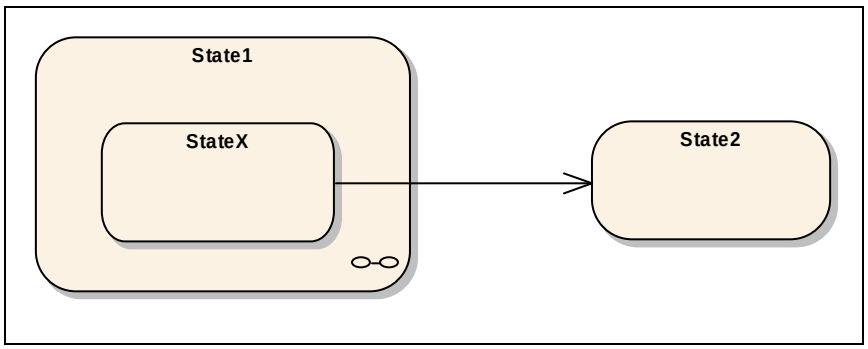


**Figure 17 – Example of a state machine with transitions from sub-states**

In Figure 18, the representation of this example as *StateMachineType* in the *AddressSpace* is given. The “Transition1”, part of the definition of “MyStateMachineType”, points to the “StateX” of the *StateMachineType* “AnotherStateMachineType”. The *Reference* is only exposed as forward *Reference* and the inverse *Reference* is omitted. Thus, there is no *Reference* from the “StateX” of “AnotherStateMachineType” to any part of “MyStateMachineType” and “AnotherStateMachineType” can be used in other places as well.

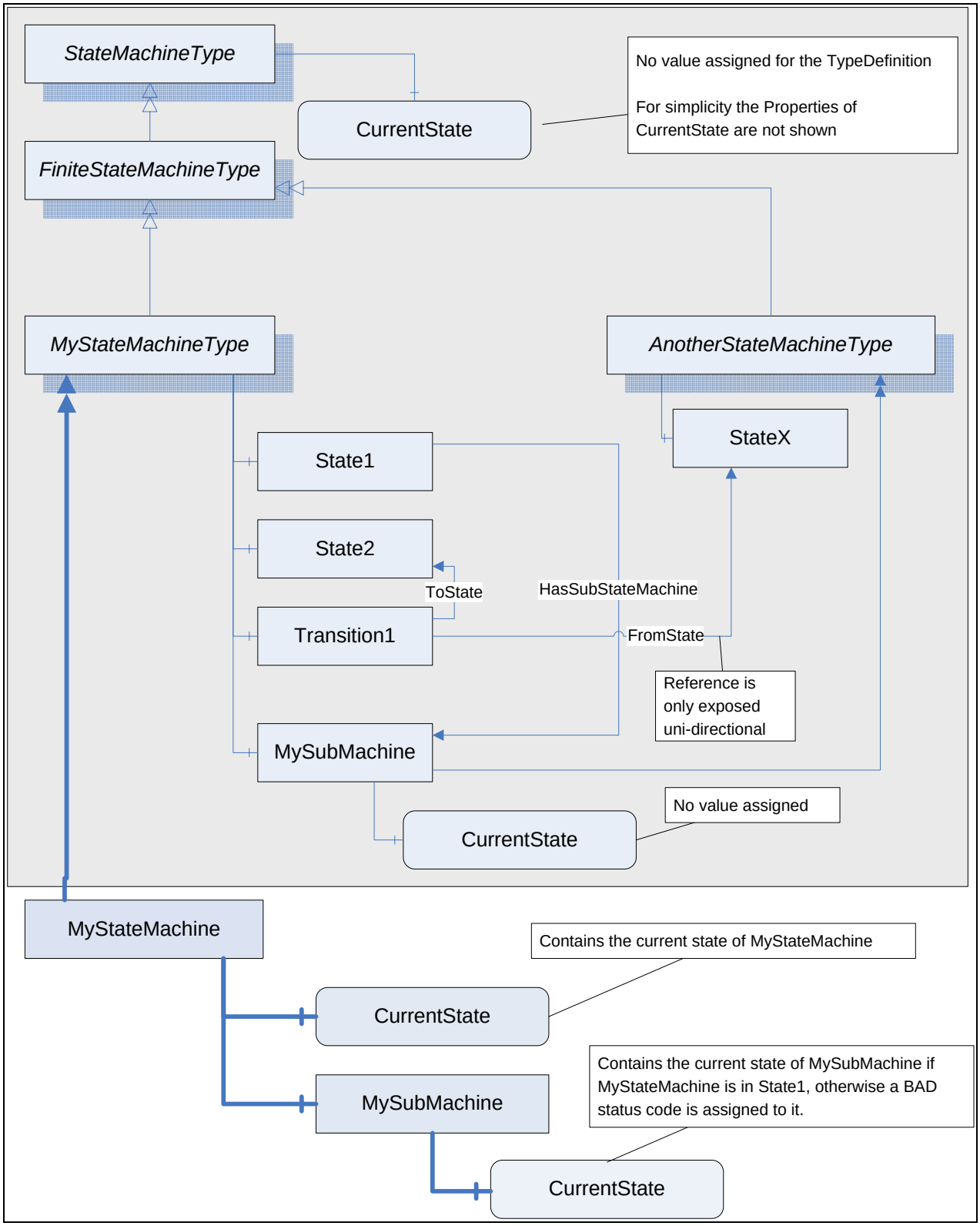


**Figure 18 – Example of a StateMachineType having Transitions to SubStateMachines**