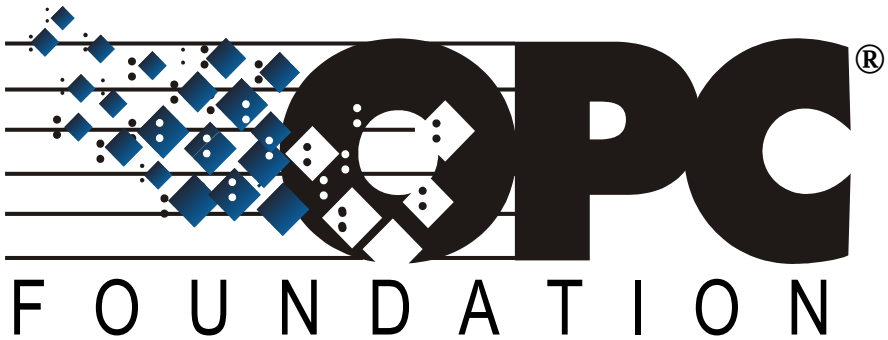

OPC 10000-110

OPC Unified Architecture

Part 110: Asset Management Basics

Release 1.00.0

2022-01-13

OPC UA Companion Specification

Specification Type:	Industry Standard Specification	Comments :	
Doc-Number	**OPC 10000-110**		
Title:	OPC Unified Architecture Part 110 :Asset Management Basics	Date:	2022-01-13
Version:	Release 1.00.0	Software:	MS-Word
		Source:	OPC 10000-100 - UA Specification Part 110 - Asset Management Basics 1.00.0.docx
Author:	OPC Foundation	Status:	Release

CONTENTS

FIGURES

TABLES

OPC FOUNDATION

UNIFIED ARCHITECTURE –

FOREWORD

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

AGREEMENT OF USE

GENERAL PROVISIONS

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

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

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

ISSUE REPORTING

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

ASSET MANAGEMENT BASICS

1 Scope

This document defines basic concepts for asset management used in an OPC UA Information Model. It considers different aspects of asset management. Applications do not have to use all or nothing of this specification, but can chose which concepts they want to support. In some cases, this specification just defines how to use concepts of the base OPC UA specifications, in other cases it does define additional OPC UA Nodes (types or standardized instances).

The intention of this specification is to define commonalities on asset management, that can be used as base from other companion specifications which might further refine those concepts for their domain specific needs. However, it is also possible to use this specification directly, without additional companion specifications.

OPC Foundation

OPC is the interoperability standard for the secure and reliable exchange of data and information in the industrial automation space and in other industries. It is platform independent and ensures the seamless flow of information among devices from multiple vendors. The OPC Foundation is responsible for the development and maintenance of this standard.

OPC UA is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework. This multi-layered approach accomplishes the original design specification goals of:

- Platform independence: from an embedded microcontroller to cloud-based infrastructure
- Secure: encryption, authentication, authorisation and auditing
- Extensible: ability to add new features including transports without affecting existing applications
- Comprehensive information modelling capabilities: for defining any model from simple to complex

2 Normative references

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

OPC 10000-1, *OPC Unified Architecture - Part 1: Overview and Concepts*

http://www.opcfoundation.org/UA/Part1/

OPC 10000-2, *OPC Unified Architecture - Part 2: Security Model*

http://www.opcfoundation.org/UA/Part2/

OPC 10000-3, *OPC Unified Architecture - Part 3: Address Space Model*

http://www.opcfoundation.org/UA/Part3/

OPC 10000-4, *OPC Unified Architecture - Part 4: Services*

http://www.opcfoundation.org/UA/Part4/

OPC 10000-5, *OPC Unified Architecture - Part 5: Information Model*
http://www.opcfoundation.org/UA/Part5/

OPC 10000-6, *OPC Unified Architecture - Part 6: Mappings*
http://www.opcfoundation.org/UA/Part6/

OPC 10000-7, *OPC Unified Architecture - Part 7: Profiles*
http://www.opcfoundation.org/UA/Part7/

OPC 10000-9, *OPC Unified Architecture - Part 9: Alarms and Conditions*
http://www.opcfoundation.org/UA/Part9/

OPC 10000-11, *OPC Unified Architecture - Part 11: Historical Access*
http://www.opcfoundation.org/UA/Part11/

OPC 10000-16, *OPC Unified Architecture - Part 16: State Machines*
http://www.opcfoundation.org/UA/Part16/

OPC 10000-17, *OPC Unified Architecture - Part 17: Alias Names*
http://www.opcfoundation.org/UA/Part17/

OPC 10000-19, *OPC Unified Architecture - Part 19: Dictionary Reference*
http://www.opcfoundation.org/UA/Part19/

OPC 10000-100, *OPC Unified Architecture - Part 100: Devices*
http://www.opcfoundation.org/UA/Part100/

3 Terms, abbreviated terms and conventions

3.1 Overview

It is assumed that basic concepts of OPC UA information modelling and asset management are understood in this document. This document will use these concepts to describe the Asset Management Basics Information Model. For the purposes of this document, the terms and definitions given in OPC 10000-1, OPC 10000-2, OPC 10000-3, OPC 10000-4, OPC 10000-5, OPC 10000-6, OPC 10000-7, OPC 10000-9, OPC 10000-11, OPC 10000-16, OPC 10000-17, OPC 10000-19 and OPC 10000-100 apply.

Note that OPC UA terms and terms defined in this document are *italicized* in the document.

3.2 Abbreviated terms

ERP	Enterprise Resource Planning
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
MES	Manufacturing Execution System
PLC	Programmable Logical Controller
PMS	Production Management System
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language

3.3 Conventions used in this document

3.3.1 Conventions for Node descriptions

3.3.1.1 Node definitions

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

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

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

- If the *TargetNode* is a component of the *Node* being defined in the table the *Attributes* of the composed *Node* are defined in the same row of the table.
- The *DataType* is only specified for *Variables*; “[<number>]” indicates a single-dimensional array, for multi-dimensional arrays the expression is repeated for each dimension (e.g. [2][3] for a two-dimensional array). For all arrays the *ArrayDimensions* is set as identified by <number> values. If no <number> is set, the corresponding dimension is set to 0, indicating an unknown size. If no number is provided at all the *ArrayDimensions* can be omitted. If no brackets are provided, it identifies a scalar *DataType* and the *ValueRank* is set to the corresponding value (see OPC 10000-3). 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 OPC 10000-3) and the *ArrayDimensions* is set to null or is omitted. Examples are given in Table 1.

Table 1 – Examples of DataTypes

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

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

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

Note that if a symbolic name of a different namespace is used, it is prefixed by the *NamespaceIndex* (see 3.3.2.2).

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

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

Table 2 – Type Definition Table

Attribute	Value				
Attribute name	Attribute value. If it is an optional Attribute that is not set "--" is used.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
ReferenceType name	*NodeClass* of the target *Node*.	*BrowseName* of the target *Node*.	*DataType* of the referenced *Node*, only applicable for *Variables*.	*TypeDefinition* of the referenced *Node*, only applicable for *Variables* and *Objects*.	Additional characteristics of the *TargetNode* such as the *ModellingRule* or *AccessLevel*.
NOTE Notes referencing footnotes of the table content.					

Components of *Nodes* can be complex that is containing components by themselves. The *TypeDefinition*, *NodeClass* and *DataType* can be derived from the type definitions, and the symbolic name can be created as defined in 3.3.3.1. Therefore, those containing components are not explicitly specified; they are implicitly specified by the type definitions.

The Other column defines additional characteristics of the Node. Examples of characteristics that can appear in this column are show in Table 3.

Table 3 – Examples of Other Characteristics

Name	Short Name	Description
0:Mandatory	M	The *Node* has the *Mandatory ModellingRule*.
0:Optional	O	The *Node* has the *Optional ModellingRule.*
0:MandatoryPlaceholder	MP	The *Node* has the *MandatoryPlaceholder ModellingRule.*
0:OptionalPlaceholder	OP	The *Node* has the *OptionalPlaceholder ModellingRule.*
ReadOnly	RO	The *Node AccessLevel* has the *CurrentRead* bit set but not the *CurrentWrite* bit.
ReadWrite	RW	The *Node AccessLevel* has the *CurrentRead* and *CurrentWrite* bits set.
WriteOnly	WO	The Node AccessLevel has the *CurrentWrite* bit set but not the *CurrentRead* bit.

If multiple characteristics are defined they are separated by commas. The name or the short name may be used.

3.3.1.2 Additional References

To provide information about additional *References*, the format as shown in Table 4 is used.

Table 4 – <some>Type Additional References

SourceBrowsePath	Reference Type	Is Forward	TargetBrowsePath
SourceBrowsePath is always relative to the *TypeDefinition*. Multiple elements are defined as separate rows of a nested table.	*ReferenceType* name	True = forward *Reference*.	TargetBrowsePath points to another *Node*, which can be a well-known instance or a *TypeDefinition*. You can use *BrowsePaths* here as well, which is either relative to the *TypeDefinition* or absolute. If absolute, the first entry needs to refer to a type or well-known instance, uniquely identified within a namespace by the *BrowseName*.

References can be to any other *Node*.

3.3.1.3 Additional sub-components

To provide information about sub-components, the format as shown in Table 5 is used.

Table 5 – <some>Type Additional Subcomponents

BrowsePath	References	NodeClass	BrowseName	DataType	TypeDefinition	Others
BrowsePath is always relative to the *TypeDefinition*. Multiple elements are defined as separate rows of a nested table	NOTE Same as for Table 2					

3.3.1.4 Additional Attribute values

The type definition table provides columns to specify the values for required *Node Attributes* for *InstanceDeclarations*. To provide information about additional *Attributes*, the format as shown in Table 6 is used.

Table 6 – <some>Type Attribute values for child Nodes

BrowsePath	<Attribute name> Attribute
BrowsePath is always relative to the *TypeDefinition*. Multiple elements are defined as separate rows of a nested table	The values of attributes are converted to text by adapting the reversible JSON encoding rules defined in OPC 10000-6. If the JSON encoding of a value is a JSON string or a JSON number then that value is entered in the value field. Double quotes are not included. If the DataType includes a NamespaceIndex (QualifiedNames, NodeIds or ExpandedNodeIds) then the notation used for BrowseNames is used. If the value is an Enumeration the name of the enumeration value is entered. If the value is a Structure then a sequence of name and value pairs is entered. Each pair is followed by a newline. The name is followed by a colon. The names are the names of the fields in the DataTypeDefinition. If the value is an array of non-structures then a sequence of values is entered where each value is followed by a newline. If the value is an array of Structures or a Structure with fields that are arrays or with nested Structures then the complete JSON array or JSON object is entered. Double quotes are not included.

There can be multiple columns to define more than one *Attribute*.

3.3.2 NodeIds and BrowseNames

3.3.2.1 NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names. Annex A 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.

The *NamespaceUri* for all *NodeIds* defined in this document is defined in Annex A. The *NamespaceIndex* for this *NamespaceUri* is vendor-specific and depends on the position of the *NamespaceUri* in the server namespace table.

Note that this document not only defines concrete *Nodes*, but also requires that some *Nodes* shall 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* used for the *Nodes* defined in this document, because they are not defined by this document but generated by the *Server*.

3.3.2.2 BrowseNames

The text part of the *BrowseNames* for all *Nodes* defined in this document is specified in the tables defining the *Nodes*. The *NamespaceUri* for all *BrowseNames* defined in this document is defined in 14.2.

For *InstanceDeclarations* of *NodeClass Object* and *Variable* that are placeholders (*OptionalPlaceholder* and *MandatoryPlaceholder ModellingRule*), the *BrowseName* and the *DisplayName* are enclosed in angle brackets (<>) as recommended in OPC 10000-3. If the *BrowseName* is not defined by this document, a namespace index prefix is added to the *BrowseName* (e.g., prefix '0' leading to '0:EngineeringUnits' or prefix '2' leading to '2:DeviceRevision'). This is typically necessary if a *Property* of another specification is overwritten or used in the OPC UA types defined in this document. Table 53 provides a list of namespaces and their indexes as used in this document.

3.3.3 Common Attributes

3.3.3.1 General

The *Attributes* of *Nodes*, their *DataTypes* and descriptions are defined in OPC 10000-3. Attributes not marked as optional are mandatory and shall be provided by a *Server*. The following tables define if the *Attribute* value is defined by this document or if it is server-specific.

For all *Nodes* specified in this document, the *Attributes* named in Table 7 shall be set as specified in the table.

Table 7 – 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* are server-specific.
Description	Optionally a server-specific description is provided.
NodeClass	Shall reflect the *NodeClass* of the *Node.*
NodeId	The *NodeId* is described by *BrowseNames* as defined in 3.3.2.1.
WriteMask	Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all non-server-specific *Attributes* to not writable. For example, the *Description Attribute* may be set to writable since a *Server* may provide a server-specific description for the *Node*. The *NodeId* shall not be writable, because it is defined for each *Node* in this document.
UserWriteMask	Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply.
RolePermissions	Optionally server-specific role permissions can be provided.
UserRolePermissions	Optionally the role permissions of the current Session can be provided. The value is server-specific and depends on the *RolePermissions Attribute* (if provided) and the current *Session*.
AccessRestrictions	Optionally server-specific access restrictions can be provided.

3.3.3.2 Objects

For all *Objects* specified in this document, the *Attributes* named in Table 8 shall be set as specified in the Table 8. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 8 – Common Object Attributes

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

3.3.3.3 Variables

For all *Variables* specified in this document, the *Attributes* named in Table 9 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 9 – Common Variable Attributes

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

3.3.3.4 VariableTypes

For all *VariableTypes* specified in this document, the *Attributes* named in Table 10 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 10 – Common VariableType Attributes

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

3.3.3.5 Methods

For all *Methods* specified in this document, the *Attributes* named in Table 11 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 11 – Common Method Attributes

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

4 General information to Asset Management and OPC UA

4.1 Introduction to Asset Management

Asset management is a huge topic with many different facets. In this specification, the topic is intentionally rather left open, and for example, this specification does not define what is considered to be an asset. The intention of this specification is to define basic information modelling constructs that can be used and refined by companion specifications to domain-specific needs.

The specification does define specific use cases like the identification and discovery of assets, the access of the health status of assets, and the classification of assets. Section 5 describes those use cases addressed by this specification in more detail.

4.2 Introduction to OPC Unified Architecture

4.2.1 What is OPC UA?

OPC UA is an open and royalty free set of standards designed as a universal communication protocol. While there are numerous communication solutions available, OPC UA has key advantages:

- A state of art security model (see OPC 10000-2).
- A fault tolerant communication protocol.
- An information modelling framework that allows application developers to represent their data in a way that makes sense to them.

OPC UA has a broad scope which delivers for economies of scale for application developers. This means that a larger number of high-quality applications at a reasonable cost are available. When combined with semantic models such as asset management basics, OPC UA makes it easier for end users to access data via generic commercial applications.

The OPC UA model is scalable from small devices to ERP systems. OPC UA *Servers* process information locally and then provide that data in a consistent format to any application requesting data - ERP, MES, PMS, Maintenance Systems, HMI, Smartphone or a standard Browser, for examples. For a more complete overview see OPC 10000-1.

4.2.2 Basics of OPC UA

As an open standard, OPC UA is based on standard internet technologies, like TCP/IP, HTTP, Web Sockets.

As an extensible standard, OPC UA provides a set of *Services* (see OPC 10000-4) and a basic information model framework. This framework provides an easy manner for creating and exposing vendor defined information in a standard way. More importantly all OPC UA *Clients* are expected to be able to discover and use vendor-defined information. This means OPC UA users can benefit from the economies of scale that come with generic visualisation and historian applications. This specification is an example of an OPC UA *Information Model* designed to meet the needs of developers and users.

OPC UA *Clients* can be any consumer of data from another device on the network to browser based thin clients and ERP systems. The full scope of OPC UA applications is shown in Figure 1.

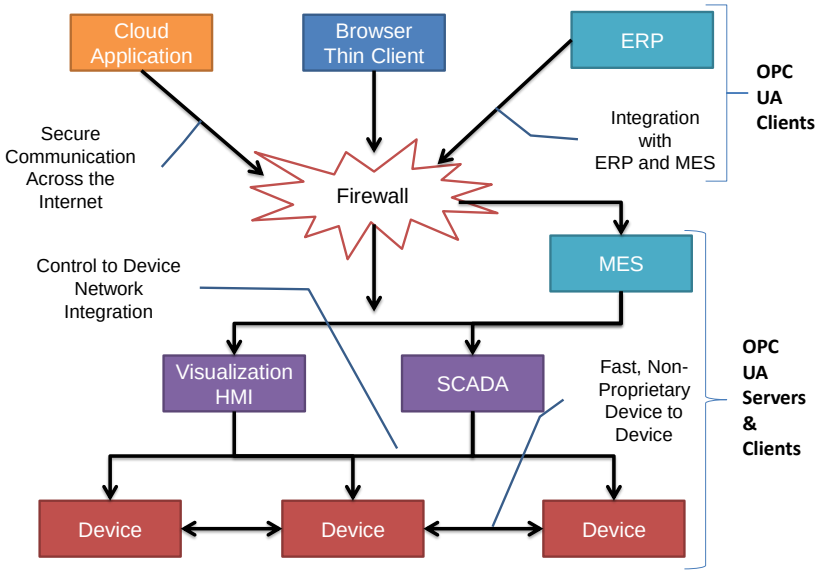

Figure 1 – The Scope of OPC UA within an Enterprise

OPC UA provides a robust and reliable communication infrastructure having mechanisms for handling lost messages, failover, heartbeat, etc. With its binary encoded data, it offers a high-performing data exchange solution. Security is built into OPC UA as security requirements become more and more important especially since environments are connected to the office network or the internet and attackers are starting to focus on automation systems.

4.2.3 Information modelling in OPC UA

4.2.3.1 Concepts

OPC UA provides a framework that can be used to represent complex information as *Objects* in an *AddressSpace* which can be accessed with standard services. These *Objects* consist of *Nodes* connected by *References*. Different classes of *Nodes* convey different semantics. For example, a *Variable Node* represents a value that can be read or written. The *Variable Node* has an associated *DataType* that can define the actual value, such as a string, float, structure etc. It can also describe the *Variable* value as a variant. A *Method Node* represents a function that can be called. Every *Node* has a number of *Attributes* including a unique identifier called a *NodeId* and non-localized name called as *BrowseName*. An *Object* representing a 'Reservation' is shown in Figure 2.

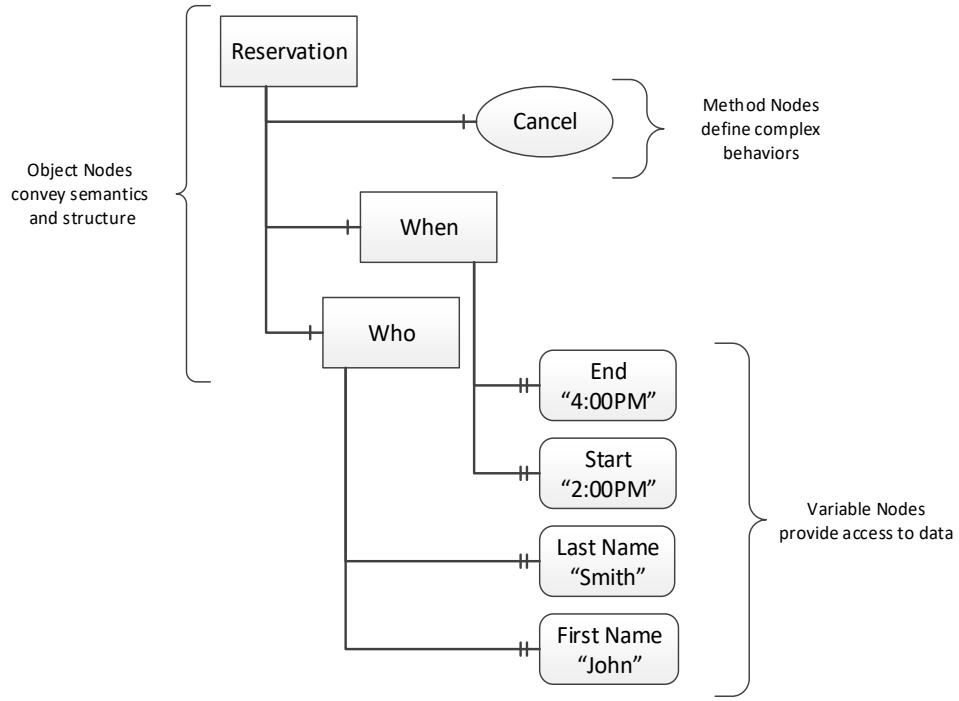

Figure 2 – A Basic Object in an OPC UA Address Space

Object and *Variable Nodes* represent instances and they always reference a *TypeDefinition* (*ObjectType* or *VariableType*) *Node* which describes their semantics and structure. Figure 3 illustrates the relationship between an instance and its *TypeDefinition*.

The type *Nodes* are templates that define all of the children that can be present in an instance of the type. In the example in Figure 3 the PersonType *ObjectType* defines two children: First Name and Last Name. All instances of PersonType are expected to have the same children with the same *BrowseNames*. Within a type the *BrowseNames* uniquely identify the children. This means *Client* applications can be designed to search for children based on the *BrowseNames* from the type instead of *NodeIds*. This eliminates the need for manual reconfiguration of systems if a *Client* uses types that multiple *Servers* implement.

OPC UA also supports the concept of sub-typing. This allows a modeller to take an existing type and extend it. There are rules regarding sub-typing defined in OPC 10000-3, but in general they allow the extension of a given type or the restriction of a *DataType*. For example, the modeller may decide that the existing *ObjectType* in some cases needs an additional *Variable*. The modeller can create a subtype of the *ObjectType* and add the *Variable*. A *Client* that is expecting the parent type can treat the new type as if it was of the parent type. Regarding *DataTypes*, subtypes can only restrict. If a *Variable* is defined to have a numeric value, a sub type could restrict it to a float.

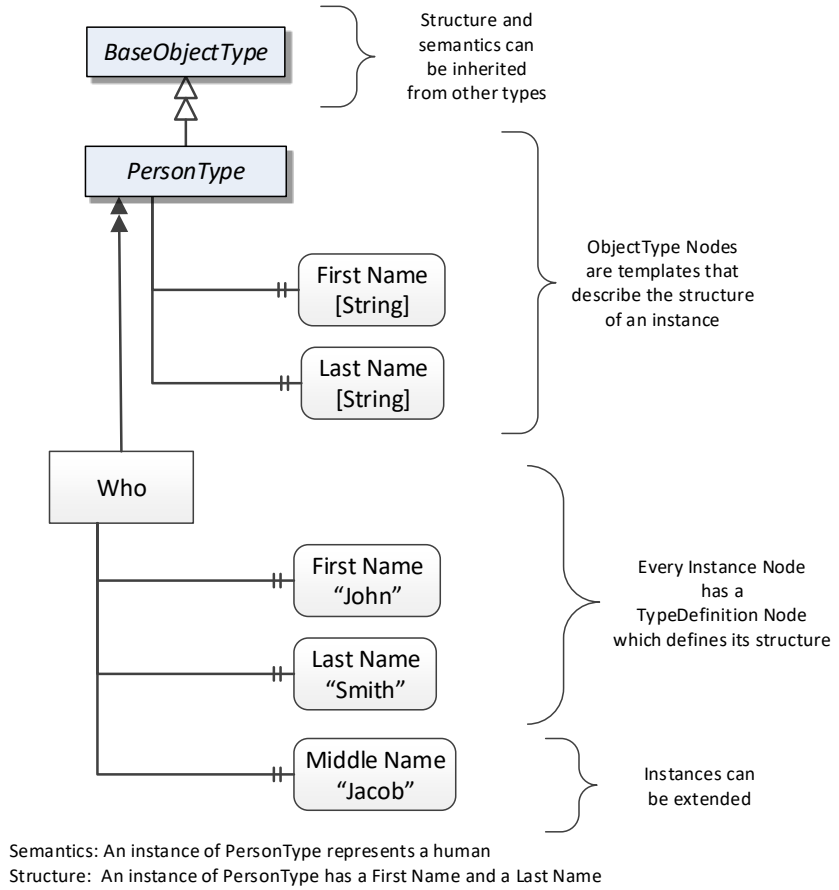

Figure 3 – The Relationship between Type Definitions and Instances

References allow *Nodes* to be connected in ways that describe their relationships. All *References* have a *ReferenceType* that specifies the semantics of the relationship. *References* can be hierarchical or non-hierarchical. Hierarchical references are used to create the structure of *Objects* and *Variables*. Non-hierarchical are used to create arbitrary associations. Applications can define their own *ReferenceType* by creating subtypes of an existing *ReferenceType*. Subtypes inherit the semantics of the parent but may add additional restrictions. Figure 4 depicts several *References,* connecting different *Objects*.

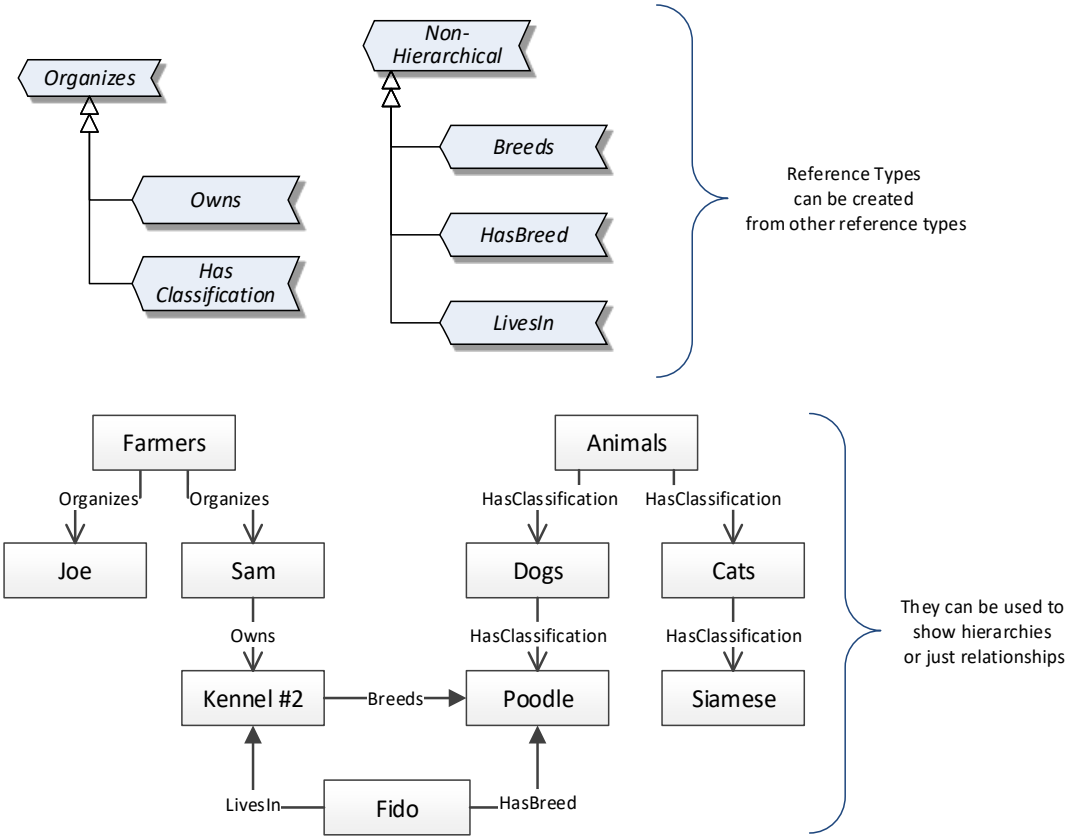

Figure 4 – Examples of References between Objects

The figures above use a notation that was developed for the OPC UA specification. The notation is summarized in Figure 5. UML representations can also be used; however, the OPC UA notation is less ambiguous because there is a direct mapping from the elements in the figures to *Nodes* in the *AddressSpace* of an OPC UA *Server*.

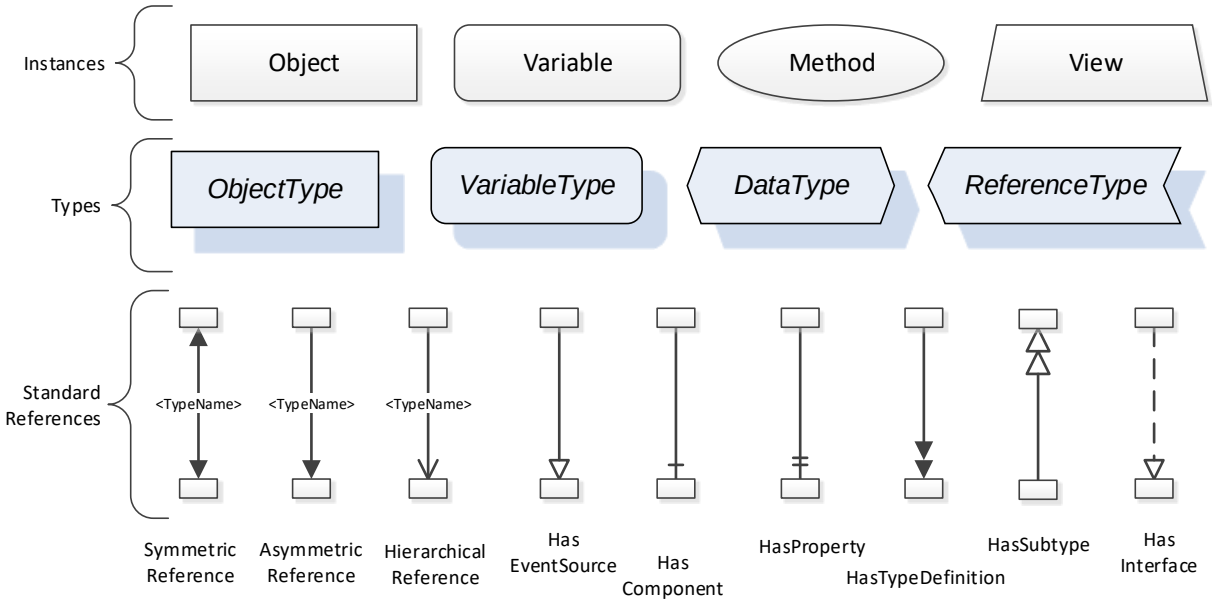

Figure 5 – The OPC UA Information Model Notation

A complete description of the different types of Nodes and References can be found in OPC 10000-3 and the base structure is described in OPC 10000-5.

OPC UA specification defines a very wide range of functionality in its basic information model. It is not required that all *Clients* or *Servers* support all functionality in the OPC UA specifications. OPC UA includes the concept of *Profiles*, which segment the functionality into testable certifiable units. This allows the definition of functional subsets (that are expected to be implemented) within a companion specification. The *Profiles* do not restrict functionality, but generate requirements for a minimum set of functionality (see OPC 10000-7).

4.2.3.2 Namespaces

OPC UA allows information from many different sources to be combined into a single coherent *AddressSpace*. Namespaces are used to make this possible by eliminating naming and id conflicts between information from different sources. Each namespace in OPC UA has a globally unique string called a NamespaceUri which identifies a naming authority and a locally unique integer called a *NamespaceIndex*, which is an index into the *Server's* table of *NamespaceUris*. The *NamespaceIndex* is unique only within the context of a *Session* between an OPC UA *Client* and an OPC UA *Server*- the *NamespaceIndex* can change between *Sessions* and still identify the same item even though the NamespaceUri's location in the table has changed. The *Services* defined for OPC UA use the *NamespaceIndex* to specify the Namespace for qualified values.

There are two types of structured values in OPC UA that are qualified with *NamespaceIndexes*: NodeIds and *QualifiedNames*. NodeIds are locally unique (and sometimes globally unique) identifiers for *Nodes*. The same globally unique *NodeId* can be used as the identifier in a node in many *Servers* – the node's instance data may vary but its semantic meaning is the same regardless of the *Server* it appears in. This means *Clients* can have built-in knowledge of what the data means in these *Nodes*. OPC UA *Information Models* generally define globally unique *NodeIds* for the *TypeDefinitions* defined by the *Information Model*.

QualifiedNames are non-localized names qualified with a Namespace. They are used for the *BrowseNames* of *Nodes* and allow the same names to be used by different information models without conflict. *TypeDefinitions* are not allowed to have children with duplicate *BrowseNames*; however, instances do not have that restriction.

4.2.3.3 Companion Specifications

An OPC UA companion specification for an industry specific vertical market describes an *Information Model* by defining *ObjectTypes*, *VariableTypes*, *DataTypes* and *ReferenceTypes* that represent the concepts used in the vertical market, and potentially also well-defined Objects as entry points into the AddressSpace.

5 Use cases

5.1 Overview

The following sections describe use cases addressed in this version of the specification. In section 5.11, use cases planned to be addressed in future versions of this specification are listed.

5.2 Identification of Asset (UC001)

ID	UC001
Name	Identification of Asset

Objective	The user wants to identify the assets managed by an OPC UA Server in a globally unique manner, receive standardized identification information about the asset and set user-specific information in order to simplify its usage.
Description	For each asset managed by an OPC UA Server the user wants to - Receive a globally unique identification of the asset that should be identical, even if the asset is managed by various OPC UA servers. - Receive standardized base identification information of the asset like serial number, manufacturer, and revision information. - Set and receive user-specific identification information that is tailored to the needs of the user, like a user-specific name or identification. This allows the user to receive and manage asset information as base for various asset management use cases. The assets can be uniquely identified, even if managed by different applications, and the user gets information about the manufacturer of the asset etc. The user can set its own information into the identification information in order to follow a user-specific identification and naming schema.
Addressed in section	7

5.3 Discovery of Assets (UC002)

ID	UC002
Name	Discovery of Assets
Objective	The user wants to easily access all assets managed in an OPC UA Server.
Description	The user wants to have a mechanism that allows access to information about all assets in an OPC UA Server with a minimum set of roundtrips (service calls). The user wants to identify if new assets have been added to the OPC UA Server or assets have been removed. This requires a unique identification of the asset (see UC001).
Required Use Cases	UC001
Addressed in section	8

5.4 Technical Specification: Version information (UC003-3)

ID	UC003-3

Name	Version information
Objective	The user wants to get Version information of an asset managed by an OPC UA Server.
Description	The user wants to get version information of an asset. This information is needed for maintenance purposes and so on. Required are the following information: - Type or Hardware-Version - Firmware Version (if applicable) - Integrated Software and version of it - Current Patch Version (History) - Revision Counter - Supported OPC UA functionality - Operating time counter - Link to digital record (handbook / delivery status /usable skills and skill changes by upgrades) - Counter about using (e.g. Relay: Counter switching operation switch operations under load) - Standard Basic Error messages
Required Use Cases	UC001
Addressed in section	10.2, 10.3, 10.4, 10.5

5.5 Health Status (UC004)

ID	UC004
Name	Health Status
Objective	The user wants to get information about the current health status of an asset managed by an OPC UA Server. This includes a simple, common overview on each asset as well as detailed information on failure conditions.
Description	For each asset managed by an OPC UA Server the user wants to - Receive a simple, common health status. This includes (from Devices spec and NAMUR NE 107):

	○ Normal – Asset functions normally ○ Failure – Malfunction of the asset ○ Check Function – Function checks are performed, like configuration checks, local operation, etc. ○ Off Spec – The asset is operating outside its specification (e.g. measuring or temperature range) or internal diagnoses indicate deviations ○ Maintenance required – Asset operates normally, the wear reserve is nearly exhausted or a function will soon be restricted due to operational conditions, e.g. build-up of deposits Note: If an asset contains sub-assets, the health status of the sub-assets are of importance for the asset, however, how this is considered is an implementation detail (e.g. if you have redundant sub-assets, one might fail without the overall asset failing). - Receive asset specific information on failure or maintenance conditions. This information should include ○ Time of failure ○ Severity ○ A description, ideally containing cause of a failure and a proposed action, how to solve the failure (e.g. restarting the asset, changing the configuration, replacing asset, doing maintenance operations, etc.) ○ Once the user has investigated the cause, she wants the ability to mask and filter issues. - Information details about maintenance is covered in UC005-1. - The user wants to find the root cause of why the asset is not operating as expected. - Ideally the user may want to know how long the asset has been in its current common health status. - The user wants to get notified if an asset changes its health status.
Required Use Cases	UC001
Addressed in section	9

5.6 Health Status: Health Categories (UC004-1)

ID	UC004-1
Name	Health Categories
Objective	Users want health indications of assets managed by an OPC UA Server to be categorized by importance.
Description	Users want assets managed by an OPC UA Sever to have classified health indications. Proposed indications: • Critical Fault – Something has permanently failed. • Major Recoverable Fault – An asset can no longer perform its function. Intervention is required. • Minor Recoverable Fault – An asset has experienced a problem but is able to continue operation until intervention occurs. • Maintenance Needed – Service is required to keep the asset operating within its designed tolerances. • Limited Resource Capacity Near Limit – A limited resource met a threshold beyond which a more serious fault would occur. Ideally users may want the ability to reassign indications to other categories based on their application.
Required Use Cases	UC001, UC004
Addressed in section	9.3

5.7 Health Status: Specific Health Conditions (UC004-2)

ID	UC004-2
Name	Specific Health Conditions
Objective	Users want health indications of assets managed by an OPC UA Server to be reported in a harmonized way.
Description	Not all assets have the same capabilities, however if an asset can detect the following health conditions, users want them reported in a standardized way. Standard Health Conditions: • One or more connections have failed • Over temperature

	• Calibration due • Self-Test has failed • Flash update in progress • Flash update has failed • Configuration is bad • OutOfResources issues and as specialisation OutOfMemory
Required Use Cases	UC001, UC004, UC004-1
Addressed in section	9.5

5.8 Health Status: Tracking of Health Information (UC004-3)

ID	UC004-3
Name	Tracking of Health Information
Objective	Users want to get information on the health information of an asset managed by an OPC UA Server over time.
Description	For each asset managed by an OPC UA Server the user wants to - determine how long an asset has been in a specific health status and how often it changed over time. - get the information about asset specific information on failure or maintenance conditions over time. - be able to reset the tracked data (starting new from that point in time). - get the information when the tracked data have been reset / started to be tracked
Required Use Cases	UC001, UC004, UC004-1, UC004-2
Addressed in section	9.6

5.9 Maintenance: Log of maintenance activities (UC005-2)

ID	UC005-2
Name	Log of maintenance records
Objective	The user wants to access the information concerning the maintenance activities of an asset managed by an OPC UA Server.

Description	The user wants to have access to the information regarding the maintenance activities of an asset. Through this information the user should be able to identify past and planned maintenance activities. This information should contain (if available): • Maintenance activity date • Estimated downtime (for planned events) • Actual downtime (for past events) • Notification for planned event (inactive/ days before / hours before) • Maintenance activity type (inspection, external check, servicing, repair, improvement) • Description of the maintenance activity • Maintenance supplier • Qualification of the personnel • Parts of the asset replaced • Parts of the asset serviced • Maintenance method (remote/local) • Configuration change (indicates whether the configuration has changed)
Required Use Cases	CU001
Addressed in section	12

5.10 Asset Classification (UC009)

ID	UC009
Name	Asset Classification
Objective	The user wants to access classification information of assets managed in an OPC UA Server.
Description	Assets can be classified in many different ways, for example by the manufacturing process (see DIN 8580), by the type of physical asset (see ISA 95), ECLASS, IEC 61360, etc. The user wants to access the classification information of each asset. An asset may be classified by different classification

	schemas. The user wants to access that information, including the used classification schema per classification.
Required Use Cases	CU001
Addressed in section	11

5.11 Use cases addressed in future versions of this specification

5.11.1 Overview

The following sections describe use cases that are not addressed in this version of the specification, but are planned to be addressed in future versions. Note that future versions are not limited to those additional use cases.

5.11.2 Technical Specification: Skills / Capabilities (UC003-1)

ID	UC0003-1
Name	Skills / Capabilities
Objective	The user wants to plan the usage of an asset managed by an OPC UA Server.
Description	For planning the usage of each asset, the user wants to receive information about: - general skills of an asset such as supported manufacturing processes, supported tools, … - products an asset can produce or transform
Required Use Cases	UC001
Addressed in section	Not addressed in this version of the specification

5.11.3 Technical Specification: Requirements (UC003-2)

ID	UC003-2
Name	Requirements
Objective	The user wants to use, position or reposition assets managed by an OPC UA Server.
Description	To use, position or reposition an asset managed by an OPC UA Server the user wants to get information about: - Connection Values such as electricity requirements (voltage, current, …) and media connection pressure if required - Connection and plug types

	- Required environmental conditions such as temperature and air pressure - Dimension and weight
Required Use Cases	UC001
Addressed in section	Not addressed in this version of the specification

5.11.4 Location/Contextualisation: Functional Contextualisation (UC007-2)

ID	UC007-2
Name	Functional Contextualisation
Objective	The user needs knowledge about the functional contextualization of assets managed by an OPC UA Server.
Description	The user needs detailed information concerning the functional contextualisation of each asset to be able to apply actions or receive information only to a dedicated group of assets which are in the same functional context, e.g. applying an update to all controllers dedicated to a specific function or receiving information only from sensors of a specific type dedicated to a specific function, e.g. pressure sensors in the paint shop area.
Required Use Cases	UC001, UC008-1, UC009
Addressed in section	Not addressed in this version of the specification

5.11.5 Location/Contextualisation: Hierarchical Location (UC007-3)

ID	UC007-3
Name	Hierarchical Location
Objective	The user needs knowledge about the hierarchical location of assets managed by an OPC UA Server.
Description	The user needs detailed information concerning the hierarchical location of each asset to be able to apply actions or receive information only from a specific part inside his plant, e.g. all assets within a specific line or emergency area. Examples for standardized hierarchies can be IEC 62264 (ISA 95), IEC 61512 (ISA 88), or IEC PAS 63088:2017 (Smart Manufacturing/Industry 4.0).
Required Use Cases	UC001
Addressed in section	Not addressed in this version of the specification

5.11.6 Location/Contextualisation: Time/Local Time Location (UC007-4)

ID	UC007-4
Name	Time / Local Time Location
Objective	The user needs knowledge about the time/local-time location of assets managed by an OPC UA Server.
Description	The user needs detailed information concerning the time/local-time location of each asset to be able to apply configuration changes related to a time zone, e.g. reconfiguration of an NTP server or applying a specific maintenance plan.
Required Use Cases	UC001
Addressed in section	Not addressed in this version of the specification

5.11.7 Location/Contextualisation: Digital Location (UC007-6)

ID	UC007-6
Name	Digital Location
Objective	The user needs knowledge about the digital location of digital assets managed in an OPC UA Server.
Description	The user needs detailed information concerning the digital location of each digital asset. The digital location can be a URI e.g. in case of a file asset.
Required Use Cases	UC001
Addressed in section	Not addressed in this version of the specification

5.11.8 Structure of Assets: Identifying the Structure of Assets (UC008-1)

ID	UC008-1
Name	Identifying the Structure of Assets
Objective	The user wants to identify the internal structure of an asset managed by an OPC UA Server, based on its sub-assets.
Description	For any selected asset, the user shall be able to identify and browse the internal structure based on the hierarchy of sub-assets. The user shall be able to identify 1) all the exposed sub-assets directly linked to the asset in a parent-child-like relation and 2) other relations between sub-assets that are exposed to the asset.

Required Use Cases	UC001
Addressed in section	Not addressed in this version of the specification

6 Asset Management Basics Information Model overview

6.1 General

This specification defines base functionality for asset management to fulfil the use cases described in section 5. It is intended to be a base specification, that can be extended by domain-specific companion specifications and vendor-specific information models. Therefore, this specification does not define, what an asset is. It can be, for example, a piece of hardware like actuators, sensors, PLCs, network equipment, or software, like a PLC program or firmware. It can contain an OPC UA Server providing the information model, like on a field device or PLC, or it can be some passive component like a calibration target, gear, or pipe. In general, it can be anything that is worth to be manged by an asset management system.

This specification defines general concepts how to manage the assets, so that a *Client* can manage those assets, independent of the type of asset. The information model is built in a modular way, defining concepts for the different use cases, that can be used independent of each other. The *ConformanceUnits* defined in section 13.1 reflect those individual functionalities. As not all concepts are independent of each other, for example the identification of an asset is needed in most cases, the *Profiles* defined in section 13.2 combine the *ConformanceUnits* to functionality that is required to be provided together.

6.2 Where to apply the Asset Management Basics Information Model?

This specification defines an Information Model to represent one or several assets. It does not define, where the Information Model should be deployed. In some cases, the asset itself might provide a *Server*, and this *Server* might provide asset management related information about the asset as defined in this specification. Other assets might not have any electronics and no capability to even run a *Server*, and any *Server* providing information about the asset is running outside the asset.

Independent of that, even if the asset provides a *Server*, not all asset management related information has to be provided by that *Server*, and some information might be provided by some higher-level system potentially aggregating the information of many assets. For example, the information when the next maintenance activity is planned might not be known by the asset but only by some higher-level asset management systems.

7 Identification of Asset

For the identification of an asset, this specification mainly uses *Properties* defined in OPC 10000-100. In order to support already released companion specifications and products as well as the different options provided by OPC 10000-100, this specification does not define or require a specific *ObjectType* or *Interface*, but *ConformanceUnits* and *Profiles* referencing mandatory and optional *Properties* as defined in OPC 10000-100.

The *Properties* defined in OPC 10000-100 for identification can either be implemented directly on the *Object* representing the asset, or in a grouping *Object* called 2:*Identification*, also defined in OPC 10000-100.

This leads to the following options, as shown in Figure 6:

1. Usage of 2:*ComponentType* or 2:*DeviceType*. An *Object* representing an asset can have the *TypeDefinition* 2:*ComponentType*, 2:*DeviceType*, or a subtype of one of those

ObjectTypes and thus containing the identification *Properties* defined in OPC 10000-100.

2. Usage of 2:*IVendorNameplateType* and 2:*ITagNameplateType*. An *Object* representing an asset can either directly implement those interfaces, or has a *TypeDefinition* directly or indirectly implementing the interfaces.

3. Usage of 2:*Identification Object*. An *Object* representing an asset provides the 2:*Identification Object*, and the 2:*Identification Object* provides the *Properties*, either by implementing the 2:*IVendorNameplateType* and 2:*ITagNameplateType* or by just providing the *Properties*.

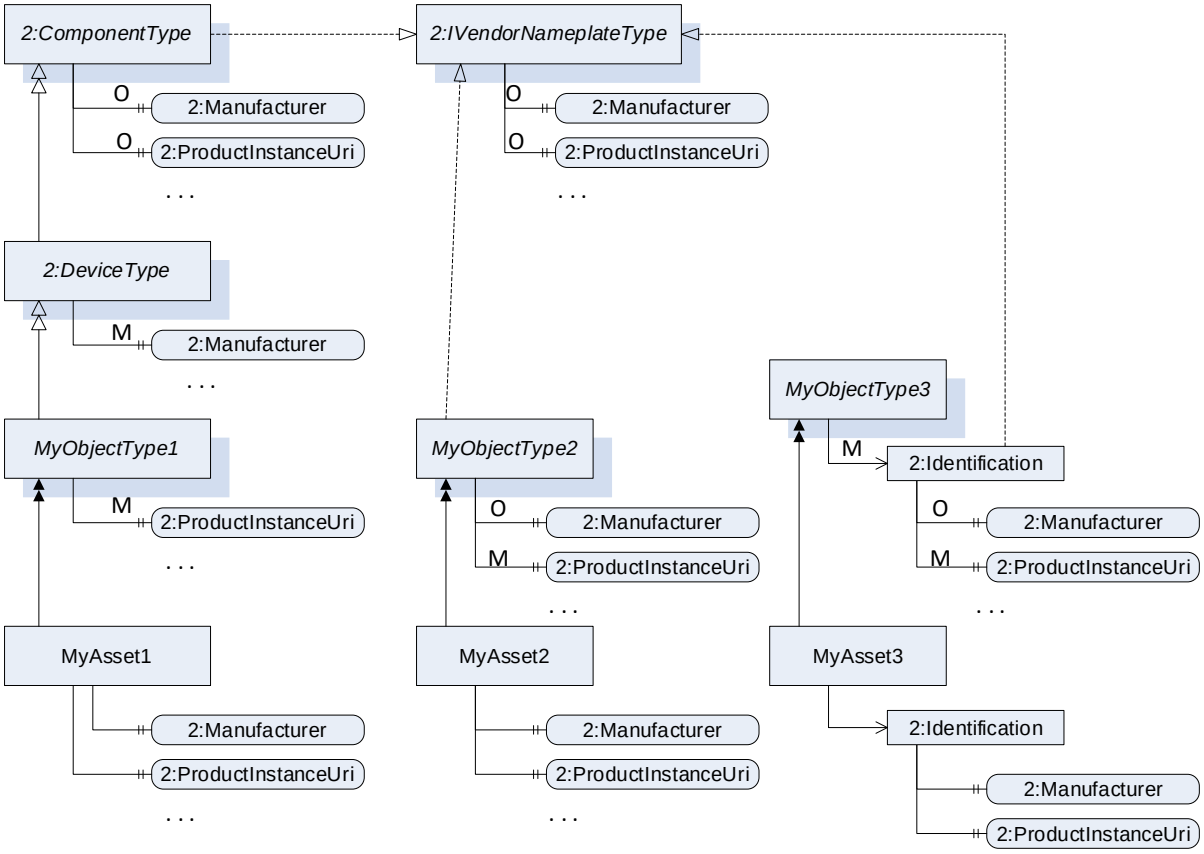

Figure 6 – Different Options for Identification of Asset

The 2:*Identification Object* shall have the *BrowseName* 2:*Identification* and shall be referenced with a hierarchical *Reference* from the *Object* representing an asset. The *ObjectType* of the 2:*Identification Object* shall be 2:*FunctionalGroupType* or a subtype of 2:*FunctionalGroupType*.

The preferred and recommended approach is to use the 2:*Identification Object*. However, *Clients* shall be aware of the other approaches and consider those for the identification of an asset as well.

OPC 10000-100 defines most of the *Properties* as mandatory on the 2:*DeviceType*, with default values when the value cannot be provided. The 2:*ComponentType* and the *Interfaces* define the *Properties* as optional, i.e., they should be omitted when the information cannot be provided. The preferred and recommended approach is to omit *Properties* if the information is not available. However, *Clients* shall be aware that there are default values and shall consider those.

In order to allow a globally unique identification of an asset, the 2:*ProductInstanceUri* shall be provided (see *ConformanceUnit* AMB Asset Identification in 13.1). It shall be either a *Property*

of the *Object* representing the asset or referenced with a hierarchical *Reference* from the 2:*Identification Object*. The *Property* value shall not be an empty String.

All other *Properties* of 2:*IVendorNameplateType* should be provided if the information is available. Other companion specifications will define additional, domain-specific identification information and vendors may add information as well. This additional information should preferably be put on the 2:*Identification Object*. A specialisation of 2:*IVendorNameplateType* can also be used to identify this additional information is identification information.

If the *2:AssetId* of the 2:*ITagNameplateType* is provided it shall be writable to allow a configurable identification of the asset (see *ConformanceUnit* AMB Configurable Asset Identification in 13.1). Other Properties of 2:*ITagNameplateType* and additional configurable Properties may be provided as well.

Older versions of OPC 10000-100 did not define the 2:*IVendorNameplateType* and 2:*ITagNameplateType* and thus *Clients* shall not expect that the 2:*IVendorNameplateType* and 2:*ITagNameplateType* are explicitly implemented on the *TypeDefinition*.

8 Discovery of Assets

8.1 Overview

For the discovery of assets, this specification uses the concept of *AliasNames*, as defined in OPC 10000-17. *AliasNames* define an additional name for any *Node* of a *Server* and can be used to categorize *AliasNames* and provide filter functionality on the *AliasNames*. The same *AliasName* can be represented by several *Nodes*, for example, if managed in different *Servers*. Using *AliasNames* simplifies managing information from various *Servers*, where some *Servers* potentially are not aware of *AliasNames. AliasNames* defines a concept how to integrate *AliasNames* into a Global Discovery Server (GDS). *AliasNames* already define the usage of *NodeVersion* and *ModelChangeEvents* to get notification on changes, which is used in this specification as well.

Unlike other usages of *AliasNames*, in this specification the purpose is not to define additional names for a *Node* / asset, but use the provided information of the assets.

That is, this specification defines two standardized categories for *AliasNames*, one using the mandatory 2:*ProductInstanceUri* as *AliasName*, and another one using the writable 2:*AssetId*, which can be set by the user.

In Figure 7, an example of the usage of *AliasNames* is given, to discover assets in a *Server*. This specification defines the standardized *Nodes Assets*, *AssetsByProductInstanceUri* and *AssetsByAssetId* (see 8.2). In order to discover the assets, a *Client* would either browse *AssetsByProductInstanceUri* or *AssetsByAssetId* to receive the *AliasNames* representing the assets, and from there with another browse access the *Nodes* representing the assets. Or it would call the *Method* 0:*FindAlias* on one of the *Objects*, allowing potentially to filter for the 2:*ProductInstanceUri* or the 2:*AssetId*. To get notified if assets have been added or removed, the Client subscribes to the 0:*NodeVersion Property* or subscribes to *ModelChangeEvents*.

As shown in the example in Figure 7, the *Server* manages three assets, X:Asset0, X:Asset1 and X:Asset2, somehow arranged in the vendor-specific hierarchy of the *Servers AddressSpace*. For all three assets, an *AliasName Object* for the 2:*ProductInstanceUri* exists. As X:Asset0 does not support the 2:*AssetId*, *AliasName Objects* for the 2:*AssetId* only exist for X:Asset1 and X:Asset2. Each *AliasName* is based on the identification information of the asset.

Note that a *Server* might not provide both entry points, depending on the supported *ConformanceUnits*.

Note that an *AliasName* could point to several *Nodes* representing the same asset (not shown in the example). The returned structure of 0:*FindAlias* allows to return several *Nodes* for one *AliasName* (see OPC 10000-17).

Note that the concept also allows for *Off-Server References*, for the 0:*AliasFor References* as well as for the 0:*FindAlias Method* (not shown in the example).

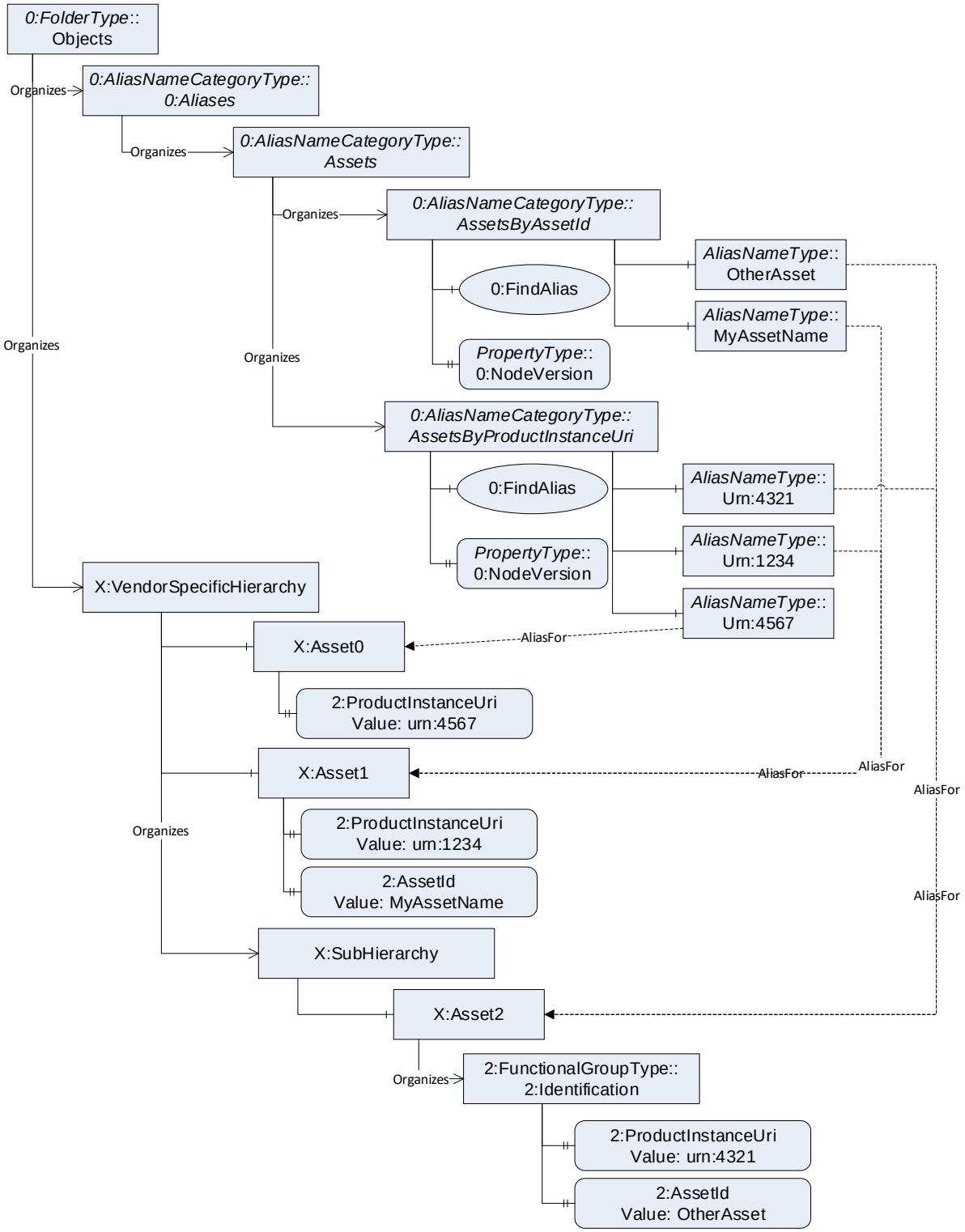

Figure 7 – Example of Discovery of Assets

8.2 Instances

8.2.1 Assets

The *Assets Object* is formally defined in Table 12.

Table 12 – Assets Definition

Attribute	**Value**			
BrowseName	Assets			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**
OrganizedBy by the 0:Aliases Object defined in OPC 10000-17				
0:HasTypeDefinition	ObjectType	0:AliasNameCategoryType	Defined in OPC 10000-17	
Conformance Units				
AMB Asset Discovery by ProductInstanceUri				
AMB Asset Discovery by AssetId				

8.2.2 AssetsByProductInstanceUri

The *AssetsByProductInstanceUri Object* is formally defined in Table 12. All *AliasNames* organized by this category shall be assets having a 2:*ProductInstanceUri* (see section 7). The string part of the *AliasName* shall be identical to the value of the 2:*ProductInstanceUri,* the namespace of the *AliasName* shall be the one defined by this specification in section 14.

Table 13 – AssetsByProductInstanceUri Definition

Attribute	**Value**			
BrowseName	AssetsByProductInstanceUri			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**
OrganizedBy by the Assets Object defined in 8.2.1				
0:HasTypeDefinition	ObjectType	0:AliasNameCategoryType	Defined in OPC 10000-17	
0:HasProperty	Variable	0:NodeVersion	0:String	PropertyType
Conformance Units				
AMB Asset Discovery by ProductInstanceUri				

8.2.3 AssetsByAssetId

The *AssetsByAssetId Object* is formally defined in Table 12. All *AliasNames* organized by this category shall be assets having a 2:*AssetId* (see section 7). The string part of the *AliasName* shall be identical to the value of the 2:*AssetId,* if the value of 2:*AssetId* is not an empty or null String. The namespace of the *AliasName* shall be the one defined by this specification in section 14.

As the default value of the 2:*AssetId* is an empty String, in a system where the 2:*AssetId* is not configured, many assets will have the identical 2:*AssetId* (empty String). In this case, each individual asset should have its own *AliasName Object* and all of those assets should not use the same *AliasName Object*. Same assets shall be, in this case, identified by the 2:*ProductInstanceUri*. In this case, the string part of the *AliasName* shall be "NoAssetIdAssigned" and the namespace of the *AliasName* shall be the one defined by this specification in section 14.

Note: Ideally, 2:*AssetIds* should be unique and two assets with different 2:*ProductInstanceUris* should have different 2:*AssetIds*. Also, if an asset is represented by more than one asset *Node*, meaning several asset *Nodes* have the same 2:*ProductInstanceUri*, the same 2:*AssetId* should be used. As the 2:*AssetId* is set by the user, this might not be the case. Assigning *AliasNames* in the context of this *Object* is done by 2:*AssetId*. The 2:*ProductInstanceUri* should only be considered when the 2:*AssetId* is empty (default value). Therefore, browsing and filtering is also done based on the 2:*AssetId*, not the 2:*ProductInstanceUri*.

Table 14 – AssetsByAssetId Definition

Attribute	Value			
BrowseName	AssetsByAssetId			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**
OrganizedBy by the Assets Object defined in 8.2.1				
0:HasTypeDefinition	ObjectType	0:AliasNameCategoryType	Defined in OPC 10000-17	
0:HasProperty	Variable	0:NodeVersion	0:String	PropertyType
Conformance Units				
AMB Asset Discovery by AssetId				

9 Health Status

9.1 Overview

In order to provide the health status of assets, this specification mainly uses the concepts defined in OPC 10000-100.

9.2 Overall health status of an asset

The *Variable* 2:*DeviceHealth* defined in OPC 10000-100 is used to provide the overall health status of an asset. It uses an enumeration having the values NORMAL, FAILURE, CHECK_FUNCTION, OFF_SPEC and MAINTENANCE_REQUIRED (see OPC 10000-100 for details).

Note that this *Variable* should be used for all assets, independent of what type of asset it is, for example also for software components. For some types of assets, not all values of the enumeration can reasonably be applied.

OPC 10000-100 defines the interface 2:*IDeviceHealthType* containing optionally the *Variable* 2:*DeviceHealth.* In older versions of OPC 10000-100 the *Variable* was only defined on the 2:*DeviceType*.

In order to support older versions of OPC 10000-100, *Objects* representing an asset and providing health status information shall provide the 2:*DeviceHealth Variable.* This should be achieved by implementing the 2:*IDeviceHealthType* interface providing optionally the 2:*DeviceHealth Variable*. This can be done either directly on the instance (X:Asset1) or via the *TypeDefinition* (X:Asset2), as shown in Figure 8. However, it is also allowed to just provide the 2:*DeviceHealth Variable* as done by X:Asset3 by using the 2:*DeviceType*. Note that in the current version of OPC 10000-100 2:*DeviceType* does implement 2:*IDeviceHealthType.*

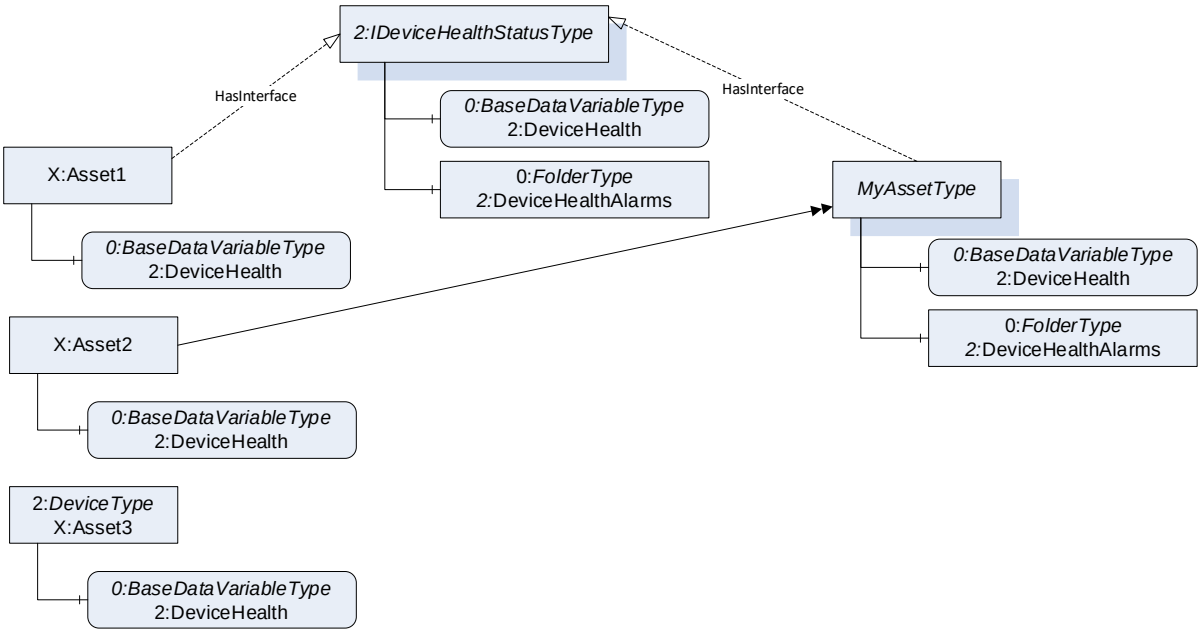

Figure 8 – Examples of assets providing health status

The *SourceTimestamp* of the 2:*DeviceHealth Variable* provides the time when the asset entered that health status. In order to provide an accurate time, *Servers* should preserve the time, also when restarting the *Server*. As this might not always be possible, a *Client* shall be aware that the *SourceTimestamp* is not always the time when the asset switched the health status.

9.3 Asset specific information on health status

To provide asset specific information on failure or maintenance conditions the alarming mechanism of OPC UA is used. The 2:*IDeviceHealthType* already defines the optional 2:*DeviceHealthAlarms* folder, where such alarm *Objects* can be provided. However, it is not required to provide the folder and the alarm *Objects* in the *AddressSpace*. *Servers* should use the *AlarmTypes* defined in OPC 10000-100 (2:*DeviceHealthDiagnosticsAlarmType* and subtypes) to expose the asset specific information. The event fields shall be used as defined in OPC 10000-5 and OPC 10000-9. The 0:*SourceNode* and 0:*SourceName* shall identify the asset. The 0:*Severity* should be used as defined in Table 15. Applications may refine those categories to a more detailed categorisation.

Table 15 – Usage of Severity for Alarms containing asset specific information

Severity	Classification	Description
801-1000	Critical Fault	The asset has permanently failed.
601-800	Major Recoverable Fault	The asset can no longer perform its function. Intervention is required.
401-600	Minor Recoverable Fault	The asset has experienced a problem but is able to continue operation until intervention occurs.
301-400	Maintenance Needed	Service is required to keep the asset operating within its designed tolerances.
201-300	Limited Resource Capacity Near Limit	A limited resource met a threshold beyond which a more serious fault would occur.
1-200	-	Used when the alarm is not active.

9.4 Root cause of asset specific information on health status

9.4.1 Overview

An asset might have several reasons why it's not operating as expected, which often are dependent on each other. For example, the asset MyMachine might indicate the faults that a pump is not working and the oil pressure is too low. The second fault is caused by the first fault. To manage assets, it is desirable to identify the root causes of failures. In order to provide the root cause of why an asset is not operating as expected, the interface *IRootCauseIndicationType* is defined (see 9.4.2). *Servers* providing this information shall use *AlarmTypes* implementing this interface for alarms indicating asset specific information on failures.

9.4.2 IRootCauseIndicationType

The *IRootCauseIndicationType* is an interface and should be applied to *AlarmTypes*. It is formally defined in Table 16.

Table 16 – IRootCauseIndicationType Definition

Attribute	**Value**				
BrowseName	IRootCauseIndicationType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:BaseInterfaceType Type defined in OPC 10000-5					
0:HasProperty	Variable	PotentialRootCauses	RootCauseDataType[]	PropertyType	M
Conformance Units					
AMB Asset Health Status Root Causes					

The mandatory *Property PotentialRootCauses* contains an array of potential root causes of the alarm. This is intended to be a hint to the client and might be a local view on the potential root causes of the alarm. The list might not contain all potential root causes, that is, other potential root causes might exist as well. If the alarm itself is considered to be the root cause, the array shall be empty. If no potential root causes have been identified, there shall be at least one entry in the array indicating that the root cause is unknown.

9.4.3 RootCauseDataType

This structure contains information about the root cause of an alarm. The structure is defined in Table 17.

Table 17 – RootCauseDataType Structure

Name	**Type**	**Description**
RootCauseDataType	structure	
RootCauseId	0:NodeId	The NodeId of the root cause of an alarm. This can point to another Node in the AddressSpace or a ConditionId, that is not necessarily represented as Object in the AddressSpace. Ideally, this points directly to the root cause. Potentially, it points to an alarm that has an additional root cause. Clients shall expect that they need to follow a path to find the root cause. If the root cause is unknown, the NodeId shall be set to NULL.
RootCause	0:LocalizedText	Localized description of the root cause of an alarm. This can be the DisplayName of the Node referenced by RootCauseId or a more descriptive text. If the root cause is unknown, this should be described in the text.

Its representation in the *AddressSpace* is defined in Table 18.

Table 18 – RootCauseDataType Definition

Attribute	Value				
BrowseName	RootCauseDataType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Structure DataType defined in OPC 10000-5					
Conformance Units					
AMB Asset Health Status Root Causes					

9.5 Standardized categories of asset specific information on health status

9.5.1 Overview

Although the alarming mechanism of OPC UA is used to indicate asset specific information on the health status, there are common indications of alarms across assets. To standardize the common indications, and leave options for extensibility by companion specifications and vendors, this specification uses the mechanism of the ConditionClassId defined for conditions in OPC 10000-9. In the following, specific subtypes of BaseConditionClassType are defined that should be used as ConditionClassId for specific alarms. Other companion specifications and vendors might add additional subtypes of BaseConditionClassType and might inherit from the types defined in this specification.

9.5.2 ConnectionFailureConditionClassType

The *ConnectionFailureConditionClassType* is used to classify conditions related to connection failures. It is formally defined in Table 19.

Table 19 – ConnectionFailureConditionClassType Definition

Attribute	Value				
BrowseName	ConnectionFailureConditionClassType				
IsAbstract	True				
Description	One or more connections have failed				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:SystemConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.3 OverTemperatureConditionClassType

The *OverTemperatureConditionClassType* is used to classify conditions related to over temperature. It is formally defined in Table 20.

Table 20 – OverTemperatureConditionClassType Definition

Attribute	Value				
BrowseName	OverTemperatureConditionClassType				
IsAbstract	True				
Description	Over temperature				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:SystemConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.4 CalibrationDueConditionClassType

The *CalibrationDueConditionClassType* is used to classify conditions related to calibration being due. It is formally defined in Table 21.

Table 21 – CalibrationDueConditionClassType Definition

Attribute	Value				
BrowseName	CalibrationDueConditionClassType				
IsAbstract	True				
Description	Calibration is due				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.5 SelfTestFailureConditionClassType

The *SelfTestFailureConditionClassType* is used to classify conditions related to self-test failures. It is formally defined in Table 22.

Table 22 – SelfTestFailureConditionClassType Definition

Attribute	Value				
BrowseName	SelfTestFailureConditionClassType				
IsAbstract	True				
Description	Self-Test failure				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:SystemConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.6 FlashUpdateInProgressConditionClassType

The *FlashUpdateInProgressConditionClassType* is used to classify conditions related to flash updates being in progress. It is formally defined in Table 23.

Table 23 – FlashUpdateInProgressConditionClassType Definition

Attribute	Value				
BrowseName	FlashUpdateInProgressConditionClassType				
IsAbstract	True				
Description	Flash update in progress				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.7 FlashUpdateFailedConditionClassType

The *FlashUpdateFailedConditionClassType* is used to classify conditions related to flash update failures. It is formally defined in Table 24.

Table 24 – FlashUpdateFailedConditionClassType Definition

Attribute	Value				
BrowseName	FlashUpdateFailedConditionClassType				
IsAbstract	True				
Description	Flash update has failed				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:SystemConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.8 BadConfigurationConditionClassType

The *ConfigurationIsBadConditionClassType* is used to classify conditions related to configurations being bad. It is formally defined in Table 25.

Table 25 – BadConfigurationConditionClassType Definition

Attribute	Value				
BrowseName	BadConfigurationConditionClassType				
IsAbstract	True				
Description	Configuration is bad				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:SystemConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.9 OutOfResourcesConditionClassType

The *OutOfResourcesConditionClassType* is used to classify conditions related to running out of resources. It is formally defined in Table 26.

Table 26 – OutOfResourcesConditionClassType Definition

Attribute	Value				
BrowseName	OutOfResourcesConditionClassType				
IsAbstract	True				
Description	Out of resources issues				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:SystemConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.5.10 OutOfMemoryConditionClassType

The *OutOfMemoryConditionClassType* is used to classify conditions related to running out of memory. It is formally defined in Table 27.

Table 27 – OutOfMemoryConditionClassType Definition

Attribute	Value				
BrowseName	OutOfMemoryConditionClassType				
IsAbstract	True				
Description	Out of memory issues				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the OutOfResourcesConditionClassType defined in 9.5.9					
Conformance Units					
AMB Asset Health Status Alarm Categories					

9.6 Tracking of health information

When the *Server* wants to provide the overall health status of an asset over time, the history of the 2:*DeviceHealth Variable* shall be used. For each 2:*DeviceHealth Variable*, where the history is currently tracked, the *Historizing Attribute* is set to True and the *AccessLevel* is set to *HistoryRead*. In order to provide the information, when history tracking started, a *HistoryDataConfiguration* is referenced with a *HasHistoricalConfiguration Reference* (see OPC 10000-11 for details).

When the *Server* wants to provide information for tracking the alarms with detailed health information of an asset over time, the history of the events based on the alarms shall be used. . This specification does not define which *Objects* are used as *EventNotifier*. As defined in the base specification, the *Server Object* shall be an *EventNotifier* when events are supported, which can be used to subscribe to all events of the Server. It is, therefore, server-specific where the history of events can be accessed and what details of the events are stored. The general concepts are defined in OPC 10000-11. The server should provide for each asset the history of events for the same amount of time as the history of the 2:*DeviceHealth*.

10 Technical Specification

10.1 Overview

The use cases for technical specifications consider different information on the assets. Some, like standard error messages, are already defined in other sections of this document and not considered further. This section is divided into subsections addressing the different information aspects.

10.2 Version Information

The version information of an asset can consist of different information like the hardware version, potentially a software version when the asset is or contains software, and a revision counter. Those *Properties* are already defined in OPC 10000-100, at the same level where the identification is defined. Therefore, this specification uses those *Properties* as defined in OPC 10000-100. They can be deployed in the same way as the identification information as defined in section 7.

Those *Properties* include:

- *HardwareRevision* providing the revision level of the hardware
- *SoftwareRevision* providing the revision level of the software
- *RevisionCounter* providing an incremental counter when the configuration of the asset has changed

For the usage of those *Properties,* different cases need to be distinguished.

1. The asset is software without hardware. In this case, only the *SoftwareRevision* is provided, and the *HardwareRevision* is omitted.

2. The asset is hardware without software. In this case, only the *HardwareRevision* is provided, and the *SoftwareRevision* is omitted.

3. The asset is a combination of hard- and software. In this case, the following model approaches can be done

    a. Represent asset as one *Object*: The asset provides the *HardwareRevision* and *SoftwareRevision*

    b. Represent software as sub-assets of hardware *Object*. This modelling approach makes specifically sense, if the software of the asset is not considered as one monolithic piece of software, but consists for example of firmware, drivers, applications, etc. In this case, the hardware is the main asset and the software components sub-assets. The main asset might contain a *SoftwareRevision* which contains the overall revision of all software assets.

In Figure 9, examples for the different cases are shown. In the example, always the *Identification Object* is used for grouping. The HardwareAsset only provides the *HardwareRevision*, the SoftwareAsset only the *SoftwareRevision*. The HardwareAndSoftwareAsset combines both and provides *HardwareRevision* and *SoftwareRevision*. The HardwareAndSoftwareAssetWithSubAssets does provide a *HardwareRevision* and an overall *SoftwareRevision*, and two sub-assets representing software assets, each containing a *SoftwareRevision*.

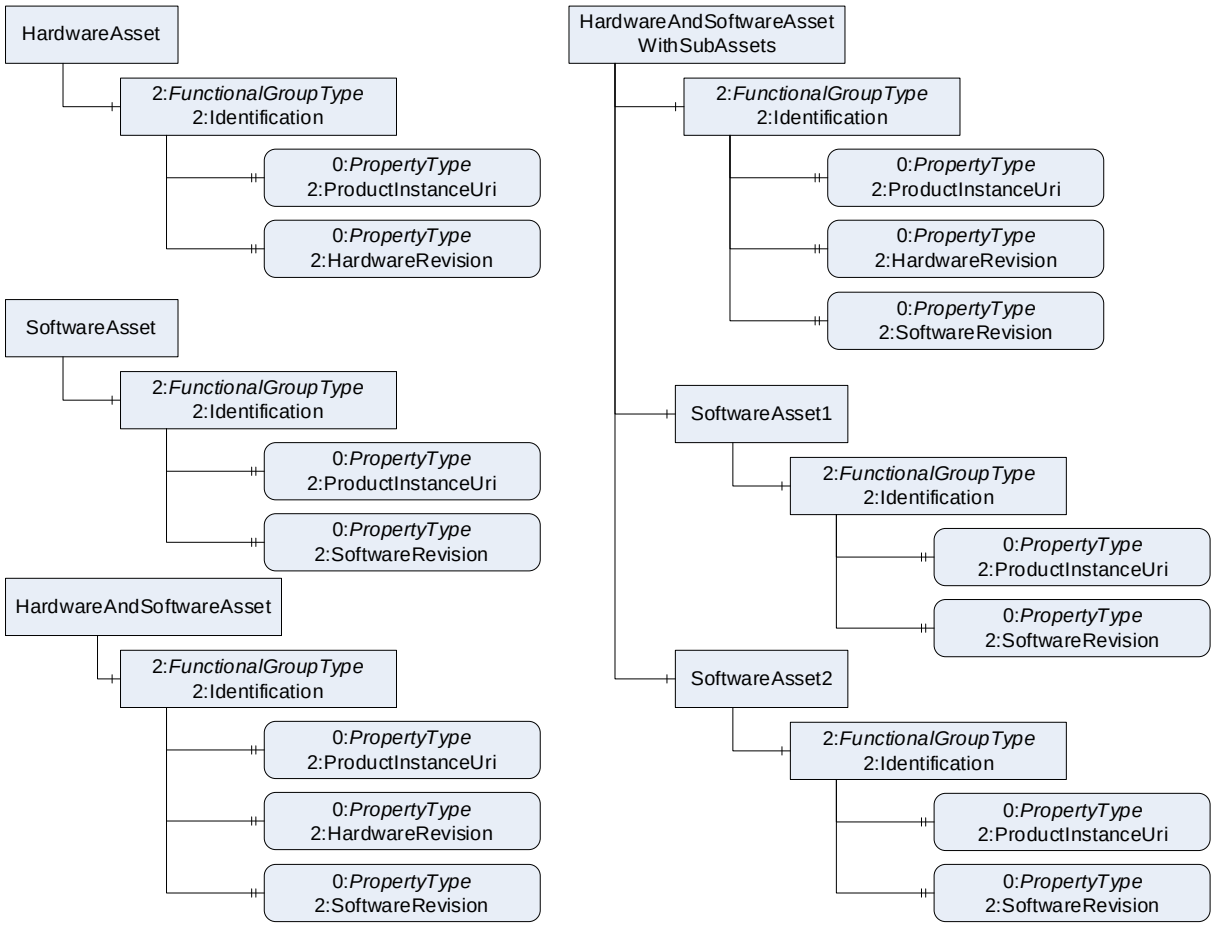

Figure 9 – Examples of the Usage of Version Information

In order to provide patch information about the software components, OPC 10000-100 defines the *SoftwareVersionType*. This is used for updating software via the OPC UA interface, using the *SoftwareUpdate AddIn*. In case the *Server* supports the *AddIn* for the asset, the *SoftwareVersionType* shall be used as defined in OPC 10000-100 to provide the patch information. In case, the Server does not support the *AddIn* for the asset, the *Property* 2:*PatchIdentifiers* should be provided on the same level as *SoftwareRevision* with the same semantic as defined on the *SoftwareVersionType*.

10.3 Operation Counters

When an asset is operating, several numbers might be of interest when managing the asset, like the hours of operation, hours in standby, etc. Those are often the base to calculate KPIs (key performance indicators) like the OEE (overall equipment efficiency). As this specification is just a base specification for asset management, and the counters are domain specific, this specification does not define any of those counters. It just defines a standardized way to find those counters. Other companion specifications or vendors might define those counters.

In order to provide access to the counters, the concept of a *FunctionalGroup* as defined in OPC 10000-100 is used. This specification just defines to use the standardized name "2:OperationCounters". Servers might provide the counters using different access paths, but they should provide the 2:OperationCounters *FunctionalGroup* as well. This *FunctionalGroup* might be organized into other *FunctionalGroups*, so *Clients* shall expect that they need to browse several hops to get to all counters. In Figure 10, an example is given.

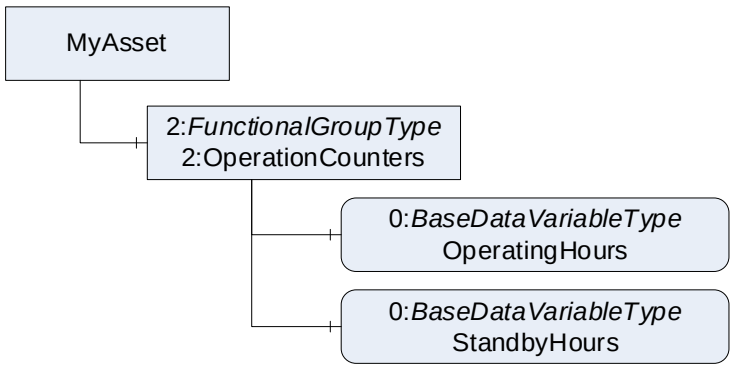

Figure 10 – Example providing Operational Counters

10.4 Supported OPC UA Functionality

If an asset supports *Server* functionality directly, it means that it does support the standardized *Server Object* defined in OPC 10000-5. This *Object* can contain information about the supported OPC UA functionality in the *ServerCapabilites Object*, like the supported OPC UA *Profiles* in the *ServerProfileArray*. The corresponding *ConformanceUnit* is already defined in OPC 10000-7. Therefore, this specification does not further address this topic.

10.5 Link to Digital Records

10.5.1 Overview

To provide links to digital records, the *ObjectType DocumentationLinksType* is introduced, which is deployed as *AddIn* on *Objects* representing assets. The *AddIn* serves as entry point to *Variables* linking to digital records managed outside the *Server*. The *Variable* contains a URI, which is typically a URL. Vendors might add additional *Properties* defining meta data of the linked document, e.g. type of file or size. In addition, it optionally provides the capabilities to add and remove editable links to user-defined documents like operator's handbooks.

In Figure 11, an example of using the DocumentationLinks *AddIn* is given.

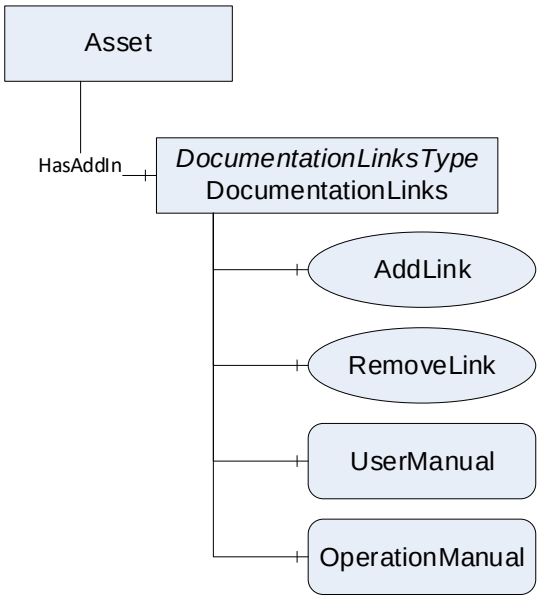

Figure 11 – Example of the DocumentationLinks AddIn

Instead of providing the optional *Methods* to add and remove links, a server not capable of the dynamics in its *AddressSpace* might also just provide some writeable *Variables* that the user can edit and thus manage its links.

10.5.2 DocumentationLinksType

The *DocumentationLinksType* is an AddIn and should be applied to *Objects* or *ObjectTypes* representing Assets. It is formally defined in Table 28.

Table 28 – DocumentationLinksType Definition

Attribute	**Value**				
BrowseName	DocumentationLinksType				
IsAbstract	False				
Description	AddIn to link documentation provided by the manufacturer and / or end-user.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:BaseObjectType Type defined in OPC 10000-5					
0:HasProperty	Variable	0:DefaultInstanceBrowseName	0:QualifiedName	PropertyType	
0:HasComponent	Variable	<Link>	0:UriString	BaseDataVariableType	OP
0:HasComponent	Method	AddLink			O
0:HasComponent	Method	RemoveLink			O
Conformance Units					
AMB DocumentationLinks Base					

The *Property* 0:DefaultInstanceBrowseName defines the default *BrowseName* of instances of this *ObjectType*.

The <Link> *Variable* represents links provided by instances of this type.

The components of the *DocumentationLinksType* have additional *Attributes* defined in Table 29.

Table 29 – DocumentationLinksType Attribute values for child Nodes

BrowsePath	Value Attribute	Description Attribute
0:DefaultInstanceBrowseName	DocumentationLinks	
<Link>		Represents links to externally managed documentation, typically URLs.
AddLink		Method to add an end-user specific link that is stored persistently in the server.
RemoveLink		Method to remove an end-user specific link that is managed in the server.

10.5.3 AddLink Method

This *Method* provides the capability to persistently add a link to an externally managed resource by adding a Variable according to the input arguments with a *HasComponent Reference* to the *Object* the *Method* is called on. The server shall manage the added link persistently, i.e., if the *Method* was executed successful, the *Variable* representing the link shall be persistent and available after restart of the *Server*. It might disappear after resetting or reconfiguring the Server or after calling the *RemoveLink Method*. The *Value Attribute* of the created *Node* should be writable, so the end-user can change the URI. In addition, the *BrowseName*, *DisplayName*, and *Description* might be writable. Servers might restrict the execution of this *Method* as well as writing the *Attributes* to specific users / roles.

The signature of this *Method* is specified below. Table 30 and Table 31 specify the *Arguments* and *AddressSpace* representation, respectively.

Signature

```
AddLink (
  [in]  0:UriString       LinkToExternalSource,
  [in]  0:QualifiedName   BrowseName,
  [in]  0:LocalizedText   DisplayName,
  [in]  0:LocalizedText   Description,
  [out] 0:NodeId          LinkVariable);
```

Table 30 – AddLink Method Arguments

Argument	Description
LinkToExternalSource	Link to an external source. The server might or might not check if a correct URI is provided, or if the URI is available/reachable.
BrowseName	The BrowseName of the new created Node. Method fails if a Variable with the same BrowseName already exists.
DisplayName	The DisplayName of the new created Node. If the server supports multiple locales, and the Client wants to provide more than one locale, the Write operation on the Variable shall be used.
Description	The Description of the new created Node. If the server supports multiple locales, and the Client wants to provide more than one locale, the Write operation on the Variable shall be used.
LinkVariable	The NodeId of the newly created Variable.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.
Bad_InvalidArgument	See OPC 10000-4 for a general description.

Table 31 – AddLink Method AddressSpace definition

Attribute	Value				
BrowseName	AddLink				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
0:HasProperty	Variable	0:InputArguments	0:Argument[]	0:PropertyType	0:Mandatory
0:HasProperty	Variable	0:OutputArguments	0:Argument[]	0:PropertyType	0:Mandatory
Conformance Units					
AMB DocumentationLinks Base					

10.5.4 RemoveLink Method

This *Method* provides the capability to remove a *Variable* representing a link. The *Method* shall only succeed when the *Variable* to be deleted is referenced from the *Object* the *Method* is called on. *Servers* might restrict the execution of this *Method* to specific users / roles. Servers will typically reject the deletion of *Nodes* not added with the *AddLink Method*.

The signature of this *Method* is specified below. Table 32 and Table 33 specify the *Arguments* and *AddressSpace* representation, respectively.

Signature

```
RemoveLink (
  [in]  0:NodeId       VariableToBeDeleted
);
```

Table 32 – RemoveLink Method Arguments

Argument	Description
VariableToBeDeleted	NodeId of the Variable containing a link, that should be deleted. Variable shall be referenced from the Object with a HasComponent Reference where the Method is called on.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.
Bad_InvalidArgument	See OPC 10000-4 for a general description.

Table 33 – RemoveLink Method AddressSpace definition

Attribute	Value				
BrowseName	RemoveLink				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
0:HasProperty	Variable	0:InputArguments	0:Argument[]	0:PropertyType	0:Mandatory
Conformance Units					
AMB DocumentationLinks Base					

11 Asset Classification

For the classification of assets, this specification uses dictionary references, as defined in OPC 10000-19. Dictionary references allow to reference external dictionaries like IEC Common Data Dictionary or ECLASS. This can be used to classify the assets according to those external dictionaries. In Figure 12, an example is given. There are two different classification schemas: “SomeClassification” and “OtherClassification”, the first one using IRDIs and the second URIs. In the example, two of the assets, Asset1 and Asset2, use both classification schemas, one classifying as flow transmitter, and the other as field device. The third asset, Asset3, only uses the second classification schema and classifies as machine. The concept of dictionary

references allows to expose many classification schemes, all managed under the standardized *Dictionaries Object* of the *Server Object*.

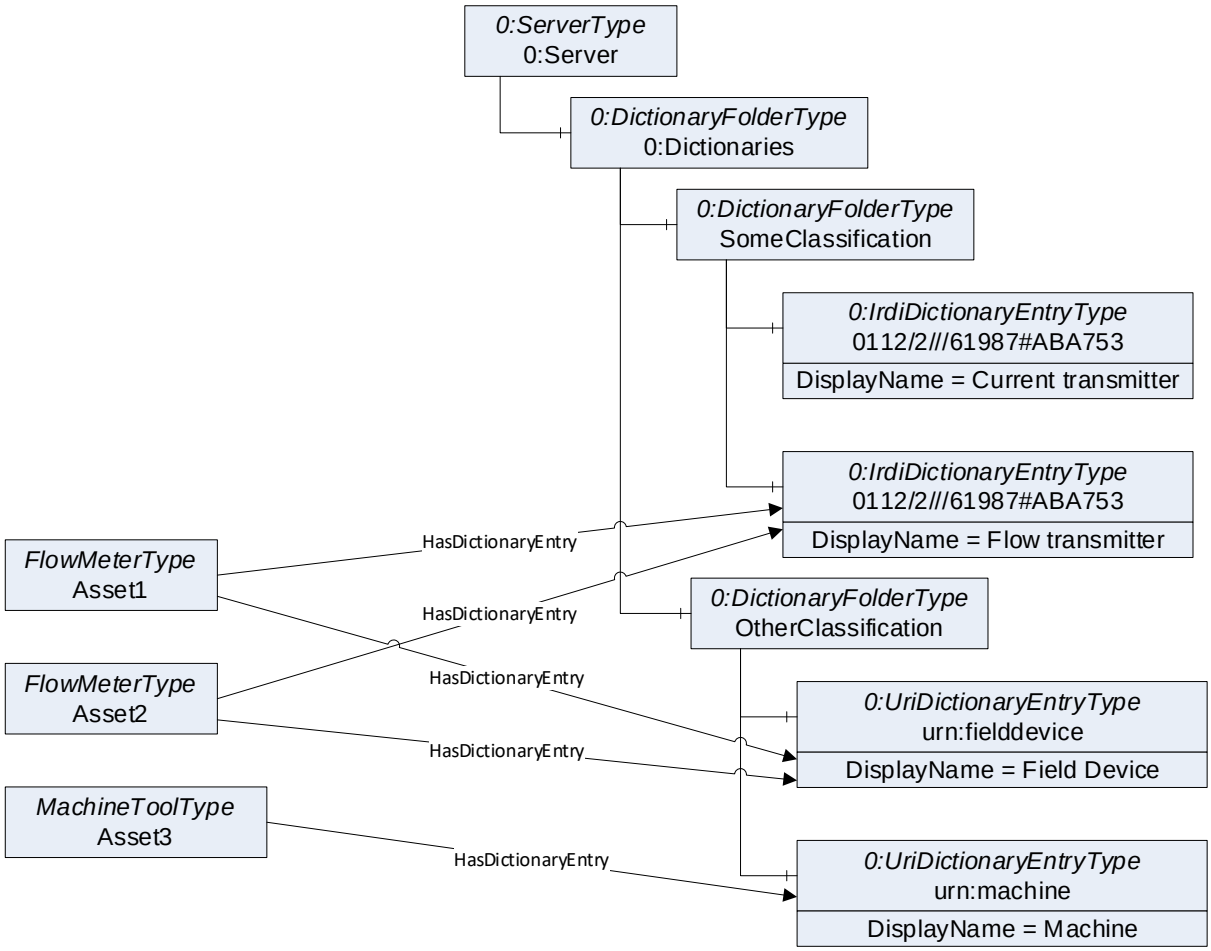

Figure 12 – Example of Asset Classification

This specification does not further define what external dictionaries should be used for classification. Companion specifications may suggest or mandate specific external dictionaries for specific types of assets.

12 Log of Maintenance Activities

12.1 Overview

To provide information about the maintenance of an asset, the alarming mechanism of OPC UA is used. This allows providing information on upcoming or not executed maintenance activities by making the conditions representing the upcoming maintenance activity active. Even if the maintenance activity has not been executed (planned or in execution), the information can already be provided to the *Client*. In this case, the 0:*Retain Property* of the condition shall be set to true, allowing *Clients* to access the information.

Using the alarming mechanism of OPC UA also allows providing information about past maintenance activities by providing access to the history of *Events* for those maintenance activities.

The conditions representing maintenance activities might be represented as *Objects* in the *AddressSpace*, however, this is not required. If they are provided, it is recommended to use the 2:*DeviceHealthAlarms* folder defined in 2:*IDeviceHealthType* as container for those Objects.

This specification does not define a specific 0:*ConditionType* for maintenance activities. Companion specifications or vendors might define their own 0:*ConditionTypes* for specific types of maintenance activities. It is recommended to create those as subtypes of 2:*MaintenanceRequiredAlarmType* defined in OPC 10000-100.

To allow clients to easily identify or filter for maintenance activities, the 0:*ConditionClassId* is used as defined for conditions in OPC 10000-9. Maintenance activities shall use the 0:*MaintenanceConditionClassType* or a subtype as 0:*ConditionClassId*. In 12.4, specific 0:*ConditionClassIds* are defined. Companion specifications or vendors might create subtypes of 0:*MaintenanceConditionClassType* or its subtypes for a more detailed classification.

The maintenance activities should provide specific information. This specification defines an interface (see 12.2) that should be implemented by all *ConditionTypes* used as maintenance activities.

For a recurring maintenance activity it is recommended to use one condition and change the state of the condition to the next planned occurrence of the maintenance activity once the maintenance activity has taken place.

When the execution of a maintenance activity fails (e.g., the correct part to exchange is not available) the message field of the event should be used to indicate the failure. The event may be extended with additional event fields for more specific information. However, this specification does not define more specific event fields for this case.

12.2 IMaintenanceEventType

The *IMaintenanceEventType* is an interface and should be applied to 0:*ConditionTypes*. It is formally defined in Table 34.

Table 34 – IMaintenanceEventType Definition

Attribute	**Value**				
BrowseName	IMaintenanceEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:BaseInterfaceType Type defined in OPC 10000-5					
0:HasComponent	Object	MaintenanceState		MaintenanceEventStateMachineType	M
0:HasProperty	Variable	PlannedDate	0:UtcTime	PropertyType	O
0:HasProperty	Variable	EstimatedDowntime	0:Duration	PropertyType	O
0:HasProperty	Variable	MaintenanceSupplier	NameNodeIdDataType	PropertyType	O
0:HasProperty	Variable	QualificationOfPersonnel	NameNodeIdDataType	PropertyType	O
0:HasProperty	Variable	PartsOfAssetReplaced	NameNodeIdDataType[]	PropertyType	O
0:HasProperty	Variable	PartsOfAssetServiced	NameNodeIdDataType[]	PropertyType	O
0:HasProperty	Variable	MaintenanceMethod	MaintenanceMethodEnum	PropertyType	O
0:HasProperty	Variable	ConfigurationChanged	0:Boolean	PropertyType	O
Conformance Units					
AMB Current and Future Maintenance Activities					

The *MaintenanceState* provides the information if the maintenance activity is still planned, currently in execution, or has already been executed.

The *PlannedDate* provides the date for which the maintenance activity has been scheduled.. In case of replanning, it is allowed to change the *PlannedDate*. However, it is not the intention that the *PlannedDate* is modified because the maintenance activity starts to get executed. If the *PlannedDate* depends for example on the operation hours of the asset, it might get adapted depending on the passed operation hours.

The *EstimatedDowntime* provides the estimated time the execution of the maintenance activity will take. In case of replanning, it is allowed to change the *EstimatedDowntime*. If during the execution of the maintenance activity the *EstimatedDowntime* can be adjusted (e.g., the asset needs to be repaired because an inspection found some issues) this should be done. Clients can access the history of Events to receive the information on the original estimates when the maintenance activity started.

The *MaintenanceSupplier* provides information on the supplier that is planned to execute, currently executing or has executed the maintenance activity. The content may change during the different *MaintenanceStates*. By accessing the history of *Events* a *Client* can distinguish between the planned and actual supplier that executed the maintenance activity. The value contains always a human-readable name of the supplier and optionally references a *Node* representing the supplier in the *AddressSpace*.

The *QualificationOfPersonnel* provides information on the qualification of the personnel that is planned to execute, currently executing or has executed the maintenance activity. The content may change during the different *MaintenanceStates*. By accessing the history of *Events* a *Client* can distinguish between the planned and actual qualification of the personnel that executed the maintenance activity. The value contains always a human-readable name of the qualification of the personnel and optionally references a *Node* representing the qualification of the personnel in the *AddressSpace*.

The *PartsOfAssetReplaced* provides information on the parts of the assets that are planned to be replaced during the maintenance activity, currently in replacement or have been replaced, depending on the different *MaintenanceStates.* The content may change during the different *MaintenanceStates*. By accessing the history of *Events* a *Client* can distinguish between the planned and actual parts of the assets replaced during the maintenance activity. The value contains always an array of a human-readable name of the qualification of the parts of the asset to be replaced and optionally references a *Node* representing each part of the asset in the *AddressSpace*.

The *PartsOfAssetServiced* provides information on the parts of the assets that are planned to be serviced during the maintenance activity, currently serviced or have been serviced, depending on the different *MaintenanceStates.* The content may change during the different *MaintenanceStates*. By accessing the history of *Events* a *Client* can distinguish between the planned and actual parts of the assets serviced during the maintenance activity. The value contains always an array of a human-readable name of the qualification of the parts of the asset to be serviced and optionally references a *Node* representing the part of the asset in the *AddressSpace*.

The *MaintenanceMethod* provides information about the planned or used maintenance method. The content may change during the different *MaintenanceStates*. By accessing the history of *Events* a *Client* can distinguish between the planned and actual used maintenance method during the maintenance activity.

The *ConfigurationChanged* provides information if the configuration of the asset is planned to be changed or has changed during the maintenance activity. TRUE indicates no change, and FALSE indicates a change. The content may change during the different *MaintenanceStates*. By accessing the history of *Events* a *Client* can distinguish between the planned and actual configuration changes during the maintenance activity.

The description of the maintenance activity should be put into the *Message* field defined for the *BaseEventType* in OPC 10000-5.

The maintenance activity starts, when the MaintenanceState changes to Execution, and finished, when it changes to Finished. Clients can use this information to identify the overall duration the maintenance took place.

Servers might change the *Severity* or other *Event Fields* as well as the state of the conditions when a maintenance activity becomes nearer to its due date in order notify *Clients* with an *Event* notification.

12.3 MaintenanceEventStateMachineType

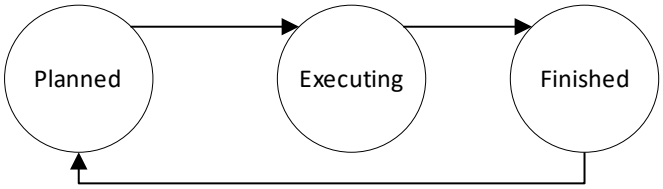

Figure 13 Example State Machine

The *MaintenanceEventStateMachineType* provides information, whether a maintenance activity is planned, currently in execution, of has been executed. A state machine diagram is shown in Figure 13. This specification does not define any effects or actions of the *MaintenanceEventStateMachineType,* the execution of the StateMachine is server-specific. The *Transitions* go from Planned to Executing to Finished. In case or repeating maintenance activities, the StateMachine might go back from Finished to Planned for another cycle. The *ObjectType* and is formally defined in Table 35.

Table 35 – MaintenanceEventStateMachineType Definition

Attribute	**Value**				
BrowseName	MaintenanceEventStateMachineType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the FiniteStateMachineType defined in OPC 10000-16, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	Planned		0:InitialStateType	
0:HasComponent	Object	Executing		0:StateType	
0:HasComponent	Object	Finished		0:StateType	
0:HasComponent	Object	FromPlannedToExecuting		0:TransitionType	
0:HasComponent	Object	FromExecutingToFinished		0:TransitionType	
0:HasComponent	Object	FromFinishedToPlanned		0:TransitionType	
Conformance Units					
AMB Current and Future Maintenance Activities					

The components of the *MaintenanceEventStateMachineType* have additional *References* which are defined in Table 36.

Table 36 – MaintenanceEventStateMachineType Additional References

SourceBrowsePath	**Reference Type**	**Is Forward**	**TargetBrowsePath**
FromPlannedToExecuting	0:FromState	True	Planned
	0:ToState	True	Executing
FromExecutingToFinished	0:FromState	True	Executing
	0:ToState	True	Finished
FromFinishedToPlanned	0:FromState	True	Finished
	0:ToState	True	Planned

The components of the *MaintenanceEventStateMachineType* have additional *Attributes* defined in Table 37.

Table 37 – MaintenanceEventStateMachineType Attribute values for child Nodes

BrowsePath	Value Attribute
Planned 0:StateNumber	1
Executing 0:StateNumber	2
Finished 0:StateNumber	3
FromPlannedToExecuting 0:TransitionNumber	1
FromExecutingToFinished 0:TransitionNumber	2
FromFinishedToPlanned 0:TransitionNumber	3

12.4 Standardized categories of Maintenance Activities

12.4.1 Overview

Maintenance activities can be classified into different categories. This specification uses the mechanism of the *ConditionClassId* defined for conditions in OPC 10000-9, for such a classification of maintenance activities. In the following, specific subtypes of *MaintenanceConditionClassType* are defined, that should be used as *ConditionClassId* for specific maintenance activities. Other companion specifications and vendors might add additional subtypes of *MaintenanceConditionClassType* and might inherit from the types defined in this specification.

12.4.2 InspectionConditionClassType

The *InspectionConditionClassType* is used to classify conditions related to inspection maintenance activities. It is formally defined in Table 38.

Table 38 – InspectionConditionClassType Definition

Attribute	Value				
BrowseName	InspectionConditionClassType				
IsAbstract	True				
Description	An inspection maintenance activity				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Current and Future Maintenance Activities					

12.4.3 ExternalCheckConditionClassType

The *ExternalCheckConditionClassType* is used to classify conditions related to external check maintenance activities. It is formally defined in Table 39.

Table 39 – ExternalCheckConditionClassType Definition

Attribute	Value				
BrowseName	ExternalCheckConditionClassType				
IsAbstract	True				
Description	An external check maintenance activity				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Current and Future Maintenance Activities					

12.4.4 ServicingConditionClassType

The *ServicingConditionClassType* is used to classify conditions related to servicing maintenance activities. It is formally defined in Table 40.

Table 40 – ServicingConditionClassType Definition

Attribute	Value				
BrowseName	ServicingConditionClassType				
IsAbstract	True				
Description	A servicing maintenance activity				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Current and Future Maintenance Activities					

12.4.5 RepairConditionClassType

The *RepairConditionClassType* is used to classify conditions related to repair maintenance activities. It is formally defined in Table 41.

Table 41 – RepairConditionClassType Definition

Attribute	Value				
BrowseName	RepairConditionClassType				
IsAbstract	True				
Description	A repair maintenance activity				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Current and Future Maintenance Activities					

12.4.6 ImprovementConditionClassType

The *ImprovementConditionClassType* is used to classify conditions related to improvement maintenance activities. It is formally defined in Table 42.

Table 42 – ImprovementConditionClassType Definition

Attribute	Value				
BrowseName	ImprovementConditionClassType				
IsAbstract	True				
Description	An improvement maintenance activity				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 0:MaintenanceConditionClassType defined in OPC 10000-9					
Conformance Units					
AMB Current and Future Maintenance Activities					

12.5 NameNodeIdDataType

This structure contains the name and optionally a NodeId. It is used to provide a human-readable name of something plus optionally the NodeId in case the something is represented in the AddressSpace. The structure is defined in Table 17.

Table 43 – NameNodeIdDataType Structure

Name	Type	Description
NameNodeIdDataType	structure	
Name	0:LocalizedText	The human-readable name. Shall be the DisplayName of the NodeId field, in case the NodeId is provided
NodeId	0:NodeId	Optionally provided NodeId, in case the referenced thing is represented as Node in the AddressSpace.

Its representation in the *AddressSpace* is defined in Table 44.

Table 44 – NameNodeIdDataType Definition

Attribute	Value				
BrowseName	NameNodeIdDataType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of Structure defined in OPC 10000-5					
Conformance Units					
AMB Current and Future Maintenance Activities					

12.6 MaintenanceMethodEnum

This enumeration provides information on the maintenance method. The enumeration is defined in Table 45.

Table 45 – MaintenanceMethodEnum Items

Name	Value	Description
Local	0	Maintenance close to the asset
Remote	1	Maintenance from another location

Its representation in the AddressSpace is defined in Table 46.

Table 46 – MaintenanceMethodEnum Definition

Attribute	Value				
BrowseName	MaintenanceMethodEnum				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Enumeration type defined in OPC 10000-5					
0:HasProperty	Variable	0:EnumValues	0:EnumValueType[]	0:PropertyType	
Conformance Units					
AMB Current and Future Maintenance Activities					

13 Profiles and Conformance Units

13.1 Conformance Units

Table 47 defines the corresponding *ConformanceUnits* for the OPC UA Information Model for Asset Management Basics.

Table 47 – Conformance Units for Asset Management Basics

Category	Title	Description
Server	AMB Asset Identification	All manageable assets provided by the server have the "ProductInstanceUri" and optionally additional identification Properties. The Properties are either directly on the Object representing the asset or on the Identification Object grouping the Properties.
Server	AMB Configurable Asset Identification	All manageable assets provided by the server have the "AssetId" and optionally additional configurable identification Properties as writable Properties that store the configuration persistently, providing the written values also after a restart of the server. Servers shall support at least 40 Unicode characters for the clients writing this value. This means clients can expect to be able to write strings with a length of 40 Unicode characters into that field. The Properties are either directly on the Object representing the asset or on the Identification Object grouping the Properties.
Client	AMB Client Asset Identification	The client can make use of asset identification information and is capable to receive that information independent if it is provided per TypeDefinition, Interface or just as Properties on the instance or under the Identification Object and is able to deal with default values of the Properties.
Server	AMB Asset Discovery by ProductInstanceUri	All manageable assets are provided as AliasNames under to AssetsByProductInstanceUri using the ProductInstanceUri as AliasName. The AssetsByProductInstanceUri Object supports NodeVersion and ModelChangeEvents.
Server	AMB Asset Discovery by AssetId	All manageable assets supporting the AssetId Property are provided as AliasNames under to AssetsByAssetId using the AssetId as AliasName. The AssetsByProductInstanceUri Object supports NodeVersion and ModelChangeEvents.

Category	Title	Description
Client	AMB Client Asset Discovery by ProductInstanceUri	The client can make use of the AssetsByProductInstanceUri discovery mechanism to discover the assets managed by the server or other, referenced servers. Whether the client uses the FindAlias method or browses the AliasNames is client-specific. The client is capable to manage Off-Server references in this discovery mechanism.
Client	AMB Client Asset Discovery by AssetId	The client can make use of the AssetsByAssetId discovery mechanism to discover the assets managed by the server or other, referenced servers. Whether the client uses the FindAlias method or browses the AliasNames is client-specific. The client is capable to manage Off-Server references in this discovery mechanism.
Server	AMB Asset Health Status Base	All manageable assets provided by the sever have the “DeviceHealth” variable.
Server	AMB Asset Health Status Alarms	All manageable assets provided by the server provide alarms with details of their fault indications. Optionally, the alarms are provided as Objects in the DeviceHealthAlarms folder.
Server	AMB Asset Health Status Root Causes	All manageable assets provided by the server provide alarms with details of their fault indications. All those alarms implement the “IRootCauseIndicationType”.
Server	AMB Asset Health Status Alarm Categories	All manageable assets provided by the server provide alarms with a reasonable ConditionClassId.
Server	AMB Asset Health Tracking Overall Asset Status	All manageable assets provided by the sever having the “DeviceHealth” variable have the Historizing attribute of that variable set to true and the AccessLevel set to HistoricalRead.
Server	AMB Asset Health Tracking Events	The server supports access of the history of all alarms representing the details of fault indications of its assets.
Client	AMB Client Asset Health Status	The client can make use of the “DeviceHealth” variable and the alarms providing details on the fault indication.
Client	AMB Client Asset Health Tracking Status	The client can make use of the history of the “DeviceHealth” variable and the history of the alarms providing details on the fault indication.
Server	AMB Version Information	All manageable assets provided by the server have, depending on their nature, the Software- and or HardwareRevision Property and the RevisionCounter Property.
Server	AMB Operation Counters	At least one manageable asset provided by the server provides the OperationCounters Object and at least one operation counter.
Server	AMB DocumentationLinks Base	At least one manageable asset provided by the server provides the DocumentationLinks AddIn and have at least one link.
Server	AMB DocumentationLinks Edit Base	All manageable assets provided by the server provide the DocumentationLinks AddIn and have at least one link that can be edited by the user to provide user-specific links. The Browse- and DisplayName and the Description might be editable. The server supports links of a length of at least 255 chars.
Server	AMB DocumentationLinks Edit Advanced	All manageable assets provided by the server provide the DocumentationLinks AddIn and support the AddLink and RemoveLink methods. The server is capable to manage at least two links added via the AddLink method. The server supports links of a length of at least 255 chars.

Category	Title	Description
Server	AMB Classification	At least one manageable asset provides at least one HasDictionaryEntry reference to classify the asset.
Server	AMB Current and Future Maintenance Activities	All manageable assets provided by the server provide conditions with details of current or upcoming maintenance activities. Optionally, the conditions are provided as Objects in the DeviceHealthAlarms folder. The ConditionTypes used as maintenance activities implement the IMaintenanceEventType.
Server	AMB Past Maintenance Activities	The server provides capability to read the history of events and for all manageable assets provided by the server the history of all maintenance activities is provided. The ConditionTypes used as maintenance activities implement the IMaintenanceEventType.
Client	AMB Client Current and Future Maintenance Activities	The client can make use of the conditions provided as maintenance activities providing details on current and planned maintenance activities.
Client	AMB Client Past Maintenance Activities	The client can make use of the history of the conditions provided as maintenance activities providing details on the past maintenance activities.

13.2 Profiles

13.2.1 Profile list

Table 48 lists all Profiles defined in this document and defines their URIs.

Table 48 – Profile URIs for Asset Management Basics

Profile	URI
AMB Base Asset Management Server Facet	http://opcfoundation.org/UA-Profile/AMB/Server/BaseServer
AMB Base Asset Management Client Facet	http://opcfoundation.org/UA-Profile/AMB/Client/BaseClient

13.2.2 Server Facets

13.2.2.1 Overview

The following sections specify the *Facets* available for *Servers* that implement the Asset Management Basics companion specification. Each section defines and describes a *Facet* or *Profile*.

13.2.2.2 AMB Base Asset Management Server Facet

Table 49 defines a *Profile* that describes the base characteristics for all OPC UA *Servers* that make use of this companion specification.

Table 49 – AMB Base Asset Management Server Facet

Group	Conformance Unit / Profile Title	Mandatory / Optional
Server	AMB Asset Identification	M
Server	AMB Configurable Asset Identification	O
Server	AMB Asset Discovery by ProductInstanceUri	O
Server	AMB Asset Discovery by AssetId	O

Group	Conformance Unit / Profile Title	Mandatory / Optional
Server	AMB Asset Health Status Base	O
Server	AMB Asset Health Status Alarms	O
Server	AMB Asset Health Status Root Causes	O
Server	AMB Asset Health Status Alarm Categories	O
Server	AMB Asset Health Tracking Overall Asset Status	O
Server	AMB Asset Health Tracking Events	O
Server	AMB Version Information	O
Server	AMB Operation Counters	O
Server	AMB DocumentationLinks Base	O
Server	AMB DocumentationLinks Edit Base	O
Server	AMB DocumentationLinks Edit Advanced	O
Server	AMB Classification	O
Server	AMB Current and Future Maintenance Activities	O
Server	AMB Past Maintenance Activities	O

13.2.3 Client Facets

13.2.3.1 Overview

The following tables specify the *Facets* available for *Clients* that implement the Asset Management Basics companion specification.

13.2.3.2 AMB Base Asset Management Client Facet

Table 50 defines a *Facet* that describes the base characteristics for all OPC UA *Clients* that make use of this companion specification.

Table 50 - AMB Base Asset Management Client Facet

Group	Conformance Unit / Profile Title	Mandatory / Optional
Client	AMB Client Asset Identification	M
Client	AMB Client Asset Discovery by ProductInstanceUri	O
Client	AMB Client Asset Discovery by AssetId	O
Client	AMB Client Asset Health Status	O
Client	AMB Client Asset Health Tracking Status	O
Client	AMB Client Current and Future Maintenance Activities	O
Client	AMB Client Past Maintenance Activities	O

14 Namespaces

14.1 Namespace Metadata

Table 51 defines the namespace metadata for this document. The *Object* is used to provide version information for the namespace and an indication about static *Nodes*. Static *Nodes* are identical for all *Attributes* in all *Servers*, including the *Value Attribute*. See OPC 10000-5 for more details.

The information is provided as *Object* of type *NamespaceMetadataType*. This *Object* is a component of the *Namespaces Object* that is part of the *Server Object*. The *NamespaceMetadataType ObjectType* and its *Properties* are defined in OPC 10000-5.

The version information is also provided as part of the ModelTableEntry in the UANodeSet XML file. The UANodeSet XML schema is defined in OPC 10000-6.

Table 51 – NamespaceMetadata Object for this Document

Attribute	**Value**	
BrowseName	http://opcfoundation.org/UA/AMB/	
Property	**DataType**	**Value**
NamespaceUri	String	http://opcfoundation.org/UA/AMB/
NamespaceVersion	String	1.00.0
NamespacePublicationDate	DateTime	2022-01-13
IsNamespaceSubset	Boolean	False
StaticNodeIdTypes	IdType[]	0
StaticNumericNodeIdRange	NumericRange[]	
StaticStringNodeIdPattern	String	

Note: The *IsNamespaceSubset Property* is set to False as the UaNodeSet XML file contains the complete Namespace. *Servers* only exposing a subset of the Namespace need to change the value to True.

14.2 Handling of OPC UA Namespaces

Namespaces are used by OPC UA to create unique identifiers across different naming authorities. The *Attributes NodeId* and *BrowseName* are identifiers. A *Node* in the UA *AddressSpace* is unambiguously identified using a *NodeId*. Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes* may have the same *BrowseName*. They are used to build a browse path between two *Nodes* or to define a standard *Property*.

Servers may often choose to use the same namespace for the *NodeId* and the *BrowseName*. However, if they want to provide a standard *Property*, its *BrowseName* shall have the namespace of the standards body although the namespace of the *NodeId* reflects something else, for example the *EngineeringUnits Property*. All *NodeIds* of *Nodes* not defined in this document shall not use the standard namespaces.

Table 52 provides a list of mandatory and optional namespaces used in an Asset Management Basics OPC UA *Server*.

Table 52 – Namespaces used in an Asset Management Basics Server

NamespaceURI	**Description**	**Use**
http://opcfoundation.org/UA/	Namespace for *NodeIds* and *BrowseNames* defined in the OPC UA specification. This namespace shall have namespace index 0.	Mandatory
Local Server URI	Namespace for nodes defined in the local server. This namespace shall have namespace index 1.	Mandatory
http://opcfoundation.org/UA/DI/	Namespace for *NodeIds* and *BrowseNames* defined in OPC 10000-100. The namespace index is *Server* specific.	Mandatory
http://opcfoundation.org/UA/AMB/	Namespace for *NodeIds* and *BrowseNames* defined in this document. The namespace index is *Server* specific.	Mandatory
Vendor specific types	A *Server* may provide vendor-specific types like types derived from *ObjectTypes* defined in this document in a vendor-specific namespace.	Optional

NamespaceURI	Description	Use
Vendor specific instances	A *Server* provides vendor-specific instances of the standard types or vendor-specific instances of vendor-specific types in a vendor-specific namespace. It is recommended to separate vendor specific types and vendor specific instances into two or more namespaces.	Mandatory

Table 53 provides a list of namespaces and their index used for *BrowseNames* in this document. The default namespace of this document is not listed since all *BrowseNames* without prefix use this default namespace.

Table 53 – Namespaces used in this document

NamespaceURI	Namespace Index	Example
http://opcfoundation.org/UA/	0	0:EngineeringUnits
http://opcfoundation.org/UA/DI/	2	2:DeviceRevision

Annex A
(normative)

Asset Management Basics Namespace and mappings

A.1 Namespace and identifiers for Asset Management Basics Information Model

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined in this document. The identifiers are specified in a CSV file with the following syntax:

```
<SymbolName>, <Identifier>, <NodeClass>
```

Where the *SymbolName* is either the *BrowseName* of a *Type Node* or the *BrowsePath* for an *Instance Node* that appears in the specification and the *Identifier* is the numeric value for the *NodeId*.

The *BrowsePath* for an *Instance Node* is constructed by appending the *BrowseName* of the instance *Node* to the *BrowseName* for the containing instance or type. An underscore character is used to separate each *BrowseName* in the path. Let's take for example, the *IRootCauseIndicationType ObjectType Node* which has the *RootCause Property*. The **Name** for the *RootCause InstanceDeclaration* within the *IRootCauseIndicationType* declaration is: *IRootCauseIndicationType_PotentialRootCauses*.

The *NamespaceUri* for all *NodeIds* defined here is http://opcfoundation.org/UA/AMB/

The CSV released with this version of the specification can be found here:
http://www.opcfoundation.org/UA/schemas/AMB/1.0/NodeIds.csv

NOTE The latest CSV that is compatible with this version of the specification can be found here:
http://www.opcfoundation.org/UA/schemas/AMB/NodeIds.csv

A computer processible version of the complete Information Model defined in this document is also provided. It follows the XML Information Model schema syntax defined in OPC 10000-6.

The Information Model Schema for this version of the document (including any revisions, amendments or errata) can be found here:
http://www.opcfoundation.org/UA/schemas/AMB/1.0/Opc.Ua.AMB.NodeSet2.xml

NOTE The latest Information Model schema that is compatible with this version of the document can be found here:
http://www.opcfoundation.org/UA/schemas/AMB/Opc.Ua.AMB.NodeSet2.xml

_____