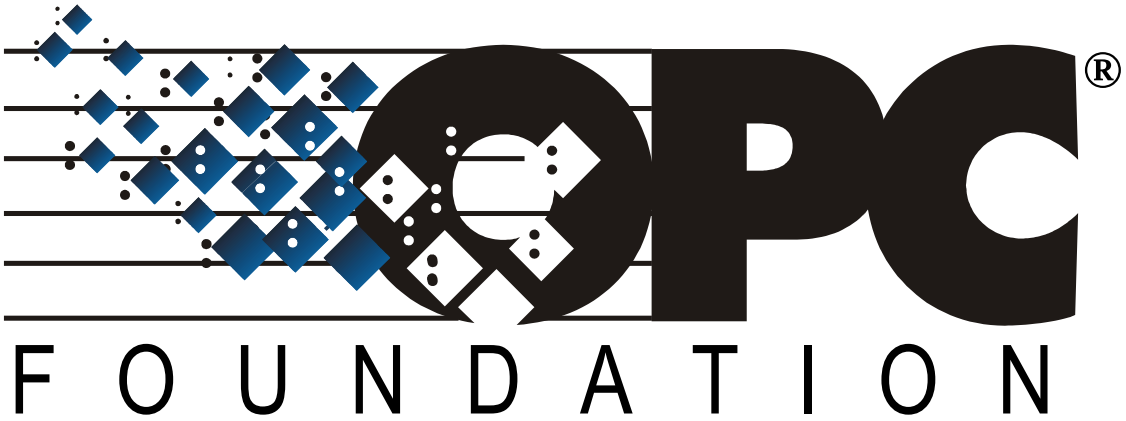

OPC Unified Architecture

for

Devices (DI)

Companion Specification

Release 1.00

December 19, 2009

Specification Type	Industry Standard Specification		
Title:	OPC Unified Architecture Devices	Date:	December 19, 2009
Version:	Release 1.00	Software Source:	MS-Word OPC UA For Devices 1.00 Companion Specification.doc
Author:	OPC Foundation	Status:	Release

CONTENTS

FIGURES

TABLES

OPC Foundation

UNIFIED ARCHITECTURE –

FOREWORD

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

AGREEMENT OF USE

only if the software satisfactorily meets the certification requirements set by the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

TRADEMARKS

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

GENERAL PROVISIONS

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

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

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

ISSUE REPORTING

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

1 Scope

In today's automation systems devices from many different manufacturers have to be integrated resulting in effort for installation, version management and device operation. This challenge can be faced best with an open and standardized device model.

This specification is an extension of the overall OPC Unified Architecture specification series and defines the information model associated with *Devices*. The model described in this specification is intended to provide a unified view of devices irrespective of the underlying device protocols.

2 Reference documents

[UA Part 1] OPC UA Specification: Part 1 – Concepts, Version 1.0 or later.
http://www.opcfoundation.org/UA/Part1/

[UA Part 2] OPC UA Specification: Part 2 – Security Model, Version 1.0 or later.
http://www.opcfoundation.org/UA/Part2/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model, Version 1.0 or later.
http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services, Version 1.0 or later.
http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model, Version 1.0 or later.
http://www.opcfoundation.org/UA/Part5/

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

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

[UA Part 8] OPC UA Specification: Part 8 – Data Access, Version 1.0 or later.
http://www.opcfoundation.org/UA/Part8/

Additional external references used to provide information model suggestions for this document:

[IEC 61131] IEC standard for Programmable Logic Controllers (PLCs).

[UA Companion ADI] OPC UA Companion Specification for Analytical Devices.

[UA Companion PLCopen] OPC UA Companion Specification for PLCopen

3 Terms, definitions, and abbreviations

3.1 OPC UA Part 1 terms

The following terms defined in [UA Part 1] of this multi-part specification apply.

1) AddressSpace
2) Attribute
3) Client
4) Complex Data
5) Information Model
6) Method
7) Node
8) NodeClass
9) Object
10) ObjectType
11) Reference
12) ReferenceType
13) Server
14) Service
15) Variable

3.2 OPC UA Part 3 terms

The following terms defined in [UA Part 3] of this multi-part specification apply.

1) DataVariable
2) ModellingRule
3) Property

3.3 OPC UA Part 8 terms

The following terms defined in [UA Part 8] of this multi-part specification apply.

1) DataItem
2) AnalogItem
3) DiscreteItem

3.4 OPC UA Devices terms

3.4.1 Block

In this specification the term Block refers to a functional *parameter* grouping entity. It could map to a function block (see [IEC 61131]) or to the resource parameters of the device itself.

3.4.2 Block Mode

Block Mode specifies the mode of operation (target mode, permitted modes, actual mode, normal mode) for a *Block*. Further details about *Block Modes* are defined by the FieldBus organisations.

3.4.3 Device

An entity that provides sensing, actuating, communication, and/or control functionality. Examples include transmitters, valve controllers, drives, motor controllers, PLCs, and communication gateways.

3.4.4 FieldBus

A specific communication bus used by the *Device*.

3.4.5 Parameter

A parameter is a variable of the *Device* that can be used for configuration, monitoring or control purposes. In the information model it is synonymous to an OPC UA *DataVariable*.

3.5 Abbreviations and symbols

DA	Data Access
DI	Device Integration (the short name for this specification)
UA	Unified Architecture
UML	Unified Modelling Language
XML	Extensible Mark-up Language

3.6 Used data types

Table 1 describes the *DataTypes* that are used through out this document.

Table 1 –DataTypes defined in [UA Part 3]

Parameter Type
LocalizedText
String
Int32

4 Fundamentals

This specification defines an OPC UA *Information Model* to represent *Devices*.

The OPC UA *Information Model* provides a standard way for *Servers* to expose *Objects* to *Clients*. *Objects* in OPC UA terms are composed of other *Objects*, *Variables* and *Methods*. OPC UA also allows relationships to other *Objects* to be expressed.

The set of *Objects* and related information that an OPC UA *Server* makes available to *Clients* is referred to as its *AddressSpace*. The elements of the OPC UA *Object* Model are represented in the *AddressSpace* as a set of *Nodes* described by *Attributes* and interconnected by *References*.

This specification makes use of two essential OPC UA *NodeClasses*: *Objects* and *Variables*.

Objects are used to represent real-world entities such as *Devices*, and software entities such as *Blocks*. An *Object* is associated to a corresponding *ObjectType* that provides definitions for that *Object*.

Variables are used to represent values. Two categories of *Variables* are defined, *Properties* and *DataVariables*.

Properties are *Server*-defined characteristics of *Objects*, *DataVariables* and other *Nodes*. *Properties* are not allowed to have *Properties* defined for them. Examples for *Properties* of *Objects* are the field device serial number and the block tag.

DataVariables represent the contents of an *Object*. *DataVariables* may have component *DataVariables*. This is typically used by *Servers* to expose individual elements of arrays and structures. This specification uses *DataVariables* to represent the *Parameters* of both *Blocks* and *Devices*.

5 Model

5.1 General

Figure 1 depicts the main *ObjectTypes* of this specification and their relationship. The drawing is not intended to be complete. For the sake of simplicity only a few components and relations were captured so as to give a rough idea of the overall structure of the DI *Information Model*.

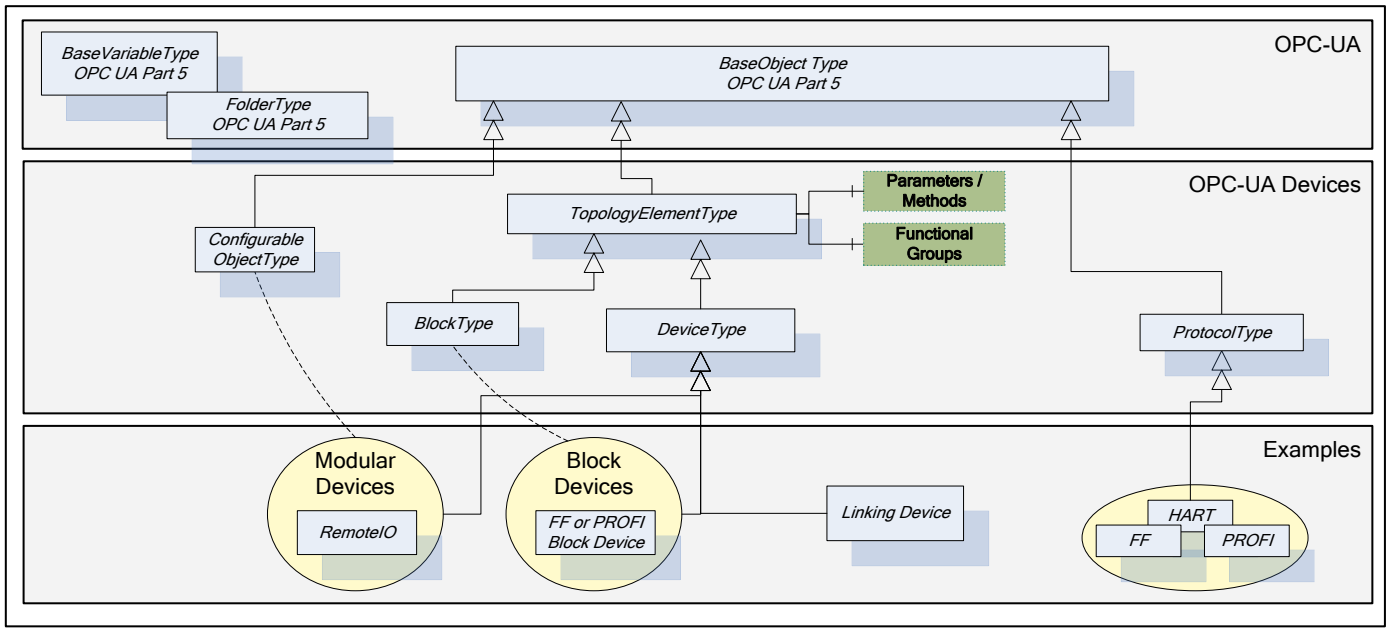

Figure 1 – *OPC UA Address Space* Overview

The boxes in this drawing show the *ObjectTypes* used in this specification as well as some elements from other specifications that help understand some modelling decisions. The upper grey box shows the OPC UA core *ObjectTypes* from which the *Device ObjectTypes* are derived. The grey box in the second level shows the *Device ObjectTypes* that this specification introduces. The components of those *ObjectTypes* are illustrated only in an abstract way in this overall picture.

The grey box in the third level shows real-world examples as they will be used in plants. In general, such subtypes are defined by other organizations.

TopologyElementType is the base *ObjectType* for elements in a device topology. It introduces *Parameters* and *Methods*. This specification also defines a functional grouping concept to provide alternative viewpoints.

The *DeviceType ObjectType* provides a general type definition for any Device. *Devices* – in addition to *Parameters* and *Methods* - may support sub-devices and may support *Blocks*. *Blocks* are typically used as means to organize the *AddressSpace*. Specific types of *Blocks* will be specified by the various *FieldBus* organizations.

A *Protocol ObjectType* represents a specific communication / *FieldBus* protocol implemented by a certain *TopologyElement*. Examples are *FFBusType*, *HARTBusType*, or PROFIBUS/ PROFINETType.

The *ConfigurableObjectType* is used as a general means to create modular topology units. If needed an instance of this type will be added to the head object of the modular unit. Modular

Devices, for example, will use this *ObjectType* to organize their *Modules*. Block-oriented *Devices* use it to expose and organize their *Blocks*.

5.2 Extensibility

Typically, the components of an *ObjectType* are fixed and can be extended by subtyping. This specification allows extending the *ObjectTypes* defined in this specification with additional components. However, it is not allowed to restrict the components of the standard *ObjectTypes* defined in this specification. An example of extending the *ObjectTypes* is user interface elements into the *TopologyElementType*.

5.3 TopologyElementType

This *ObjectType* defines the basic information components for all configurable elements in a device topology. It introduces *Parameters* and *FunctionalGroups*. Figure 2 shows the *TopologyElementType*. It is formally defined in Table 2.

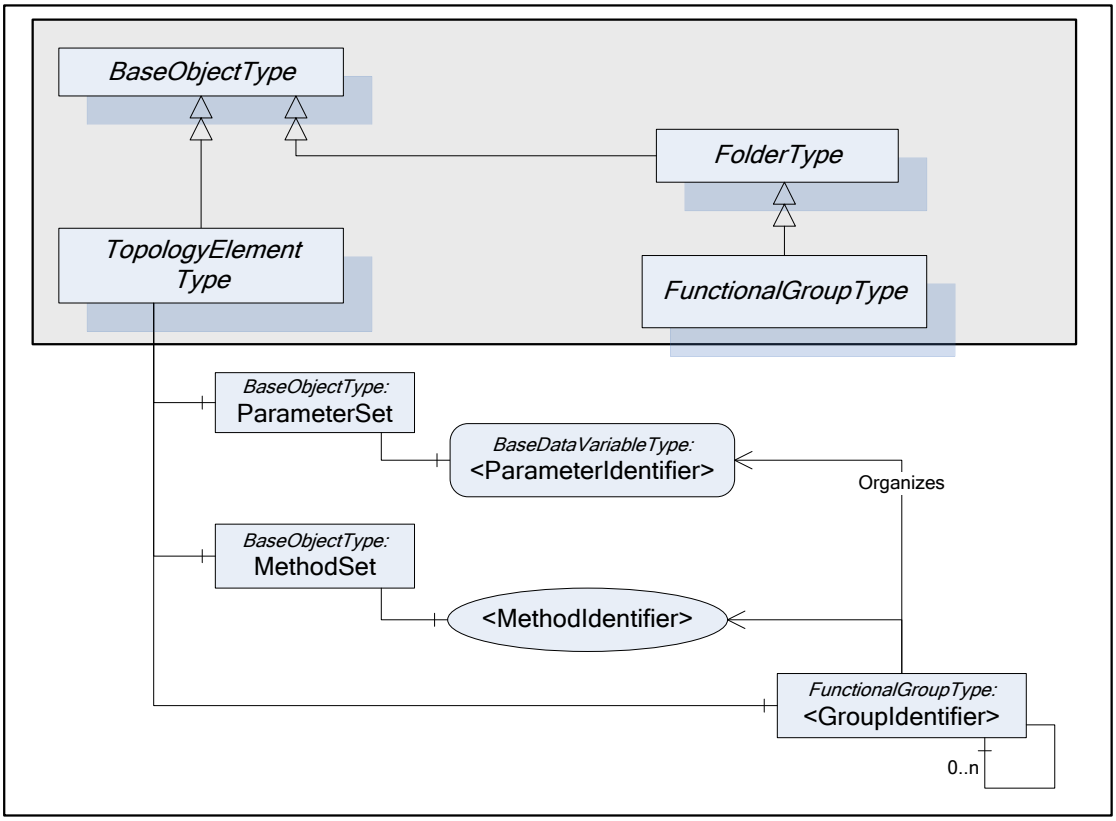

Figure 2 – *TopologyElementType* –*Parameters*, *Methods* and *FunctionalGroups*

All elements in a topology may have *Parameters* and *Methods*. If such an element has *Parameters* they are kept in an *Object* called "ParameterSet" as a flat list of *Parameters*. If it has *Methods* they are kept the same way in an *Object* called "MethodSet". *FunctionalGroups* can be used to organise the *Parameters* and *Methods* to reflect the structure of the *TopologyElement*. The same *Parameter* or *Method* might be referenced from more than one *FunctionalGroup*. *FunctionalGroups* are specified in clause 5.4.

Rule for all *Methods* organized in *FunctionalGroups*:
Since *Methods* are components of the *MethodSet Object*, *Clients* have to specify the *NodeId* of the *MethodSet* instance in the objectId argument of the Call *Service*.

Parameters are modelled with OPC UA *DataVariable* nodes. A *Parameter* can be an instance of *BaseDataVariableType* (defined in [UA Part 5], *DataItemType, AnalogType,* or any of the *DiscreteTypes* (all defined in [UA Part 8]. Sub-types of these can be defined by other standard

organisations. In this specification, the term *Parameter* is used generically - regardless of *VariableType*.

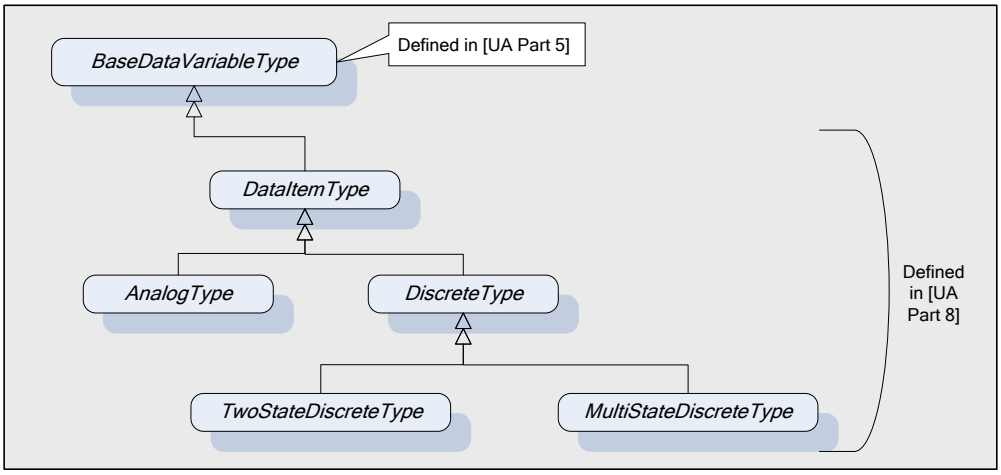

Figure 3 – OPC UA *VariableType* Hierarchy

The *TopologyElement ObjectType* is formally defined in Table 2

Table 2 – *TopologyElementType* Definition

Attribute	**Value**				
BrowseName	TopologyElementType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *BaseObjectType* defined in [UA Part 5]					
HasSubtype	ObjectType	DeviceType	Defined in Clause 5.4		
HasSubtype	ObjectType	BlockType	Defined in Clause 5.10		
HasComponent	Object	ParameterSet		BaseObjectType	Optional
HasComponent	Object	MethodSet		BaseObjectType	Optional
HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	Optional
HasComponent	Object	Identification		FunctionalGroupType	Optional

The *TopologyElementType* is abstract. There will be no instances of a *TopologyElementType* itself, but there will be instances of sub-types of this type. In this specification, the term *TopologyElement* generically refers to an instance of any *ObjectType* derived from the *TopologyElementType*.

ParameterSet gathers the references to all *Parameters* that are exposed to the *Client*. The *ParameterSet* is mandatory if *Parameters* exist for a *TopologyElement*.

MethodSet gathers the references to all *Methods* that are exposed to the *Client*.

TopologyElements may have an arbitrary number of *FunctionalGroups* to organize *Parameters* and *Methods* (see section 5.4).

A special *FunctionalGroup* called **Identification** shall be used to organize *Parameter*s for identification of this *TopologyElement* (see section 5.5).

5.4 FunctionalGroupType

This sub-type of the OPC UA *FolderType* is used to organize the *Parameters* and *Methods* from the complete set (*ParameterSet*, *MethodSet*) with regard to their application. The same *Parameter* or *Method* might be referenced from more than one *FunctionalGroup*.

FunctionalGroups can be nested.

Figure 4 shows the *FunctionalGroupType* components. It is formally defined in Table 3.

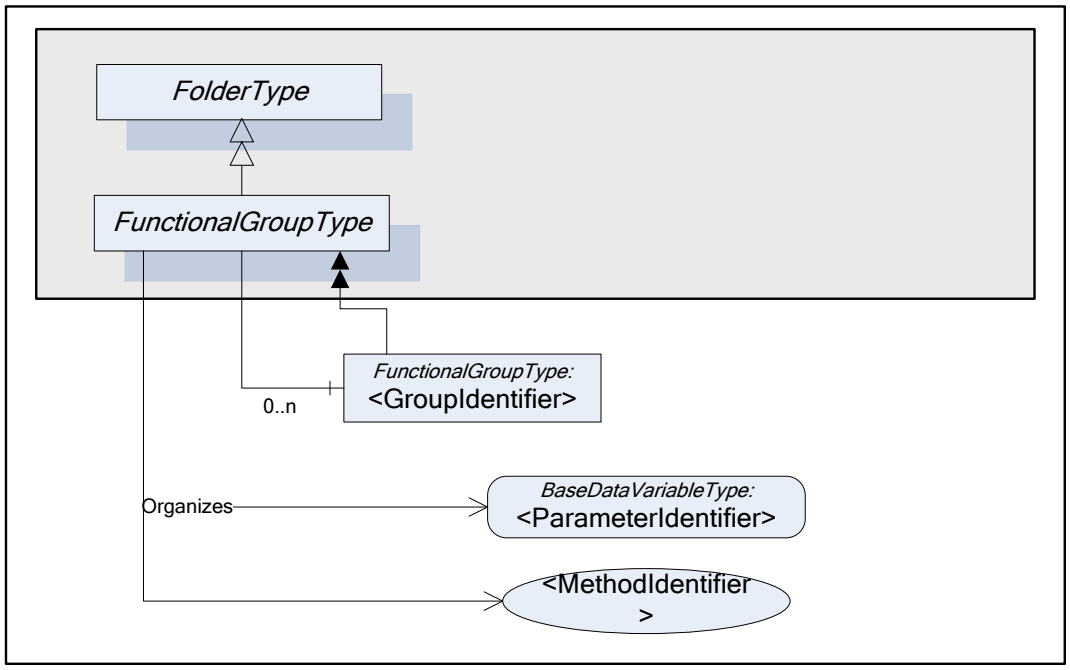

Figure 4 – *FunctionalGroupType*

Table 3 – *FunctionalGroupType* Definition

Attribute	**Value**				
BrowseName	FunctionalGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *FolderType* defined in [UA Part 5]					
HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	Optional
Organizes	Variable	<ParameterIdentifier>		BaseDataVariableType	Optional
Organizes	Method	<MethodIdentifier>			Optional

The Organizes *References* may be present only at the instance, not the type. Depending on the current state of the *TopologyElement* the *Server* may decide to hide or unhide certain *FunctionalGroup*s or (part of) their *References*.

The *Description Attribute* is used to describe the intended purpose of the *FunctionalGroup*.

The examples in Figure 5 and Figure 6 illustrate the use of *FunctionalGroups:*

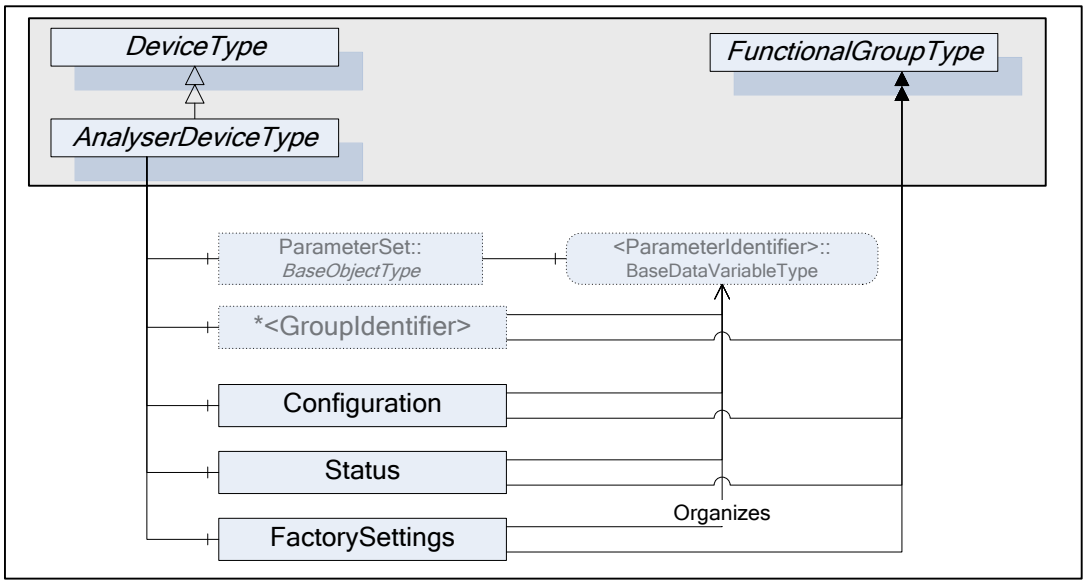

Figure 5 – Analyser Device use for FunctionalGroups ([UA Companion ADI])

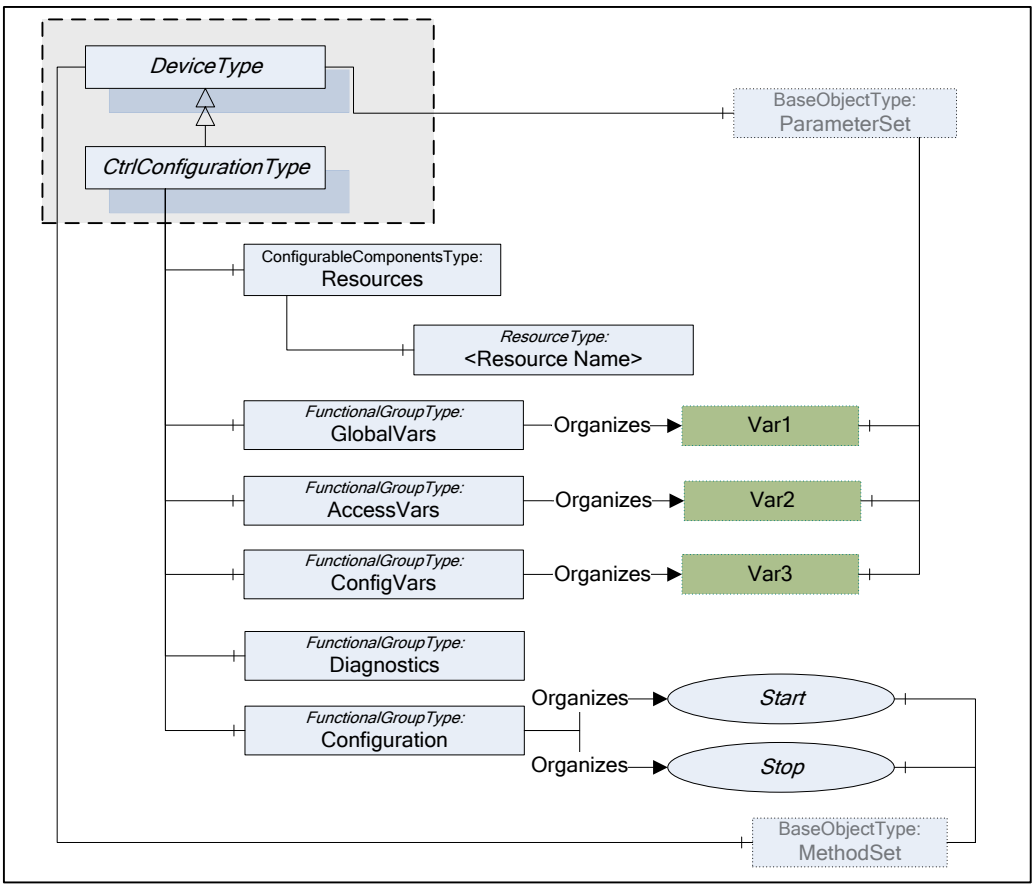

Figure 6 – PLCopen use for FunctionalGroups ([UA Companion PLCopen])

5.5 Identification Functional Group

*Parameter*s for identification of a *TopologyElement* shall be organized in a *FunctionalGroup* called **Identification**. As an example *Clients* can use the values of these *Parameter*s to find certain elements or detect mismatches between configuration data and the currently connected element. Figure 2 illustrates the **Identification** *FunctionalGroup* with an example.

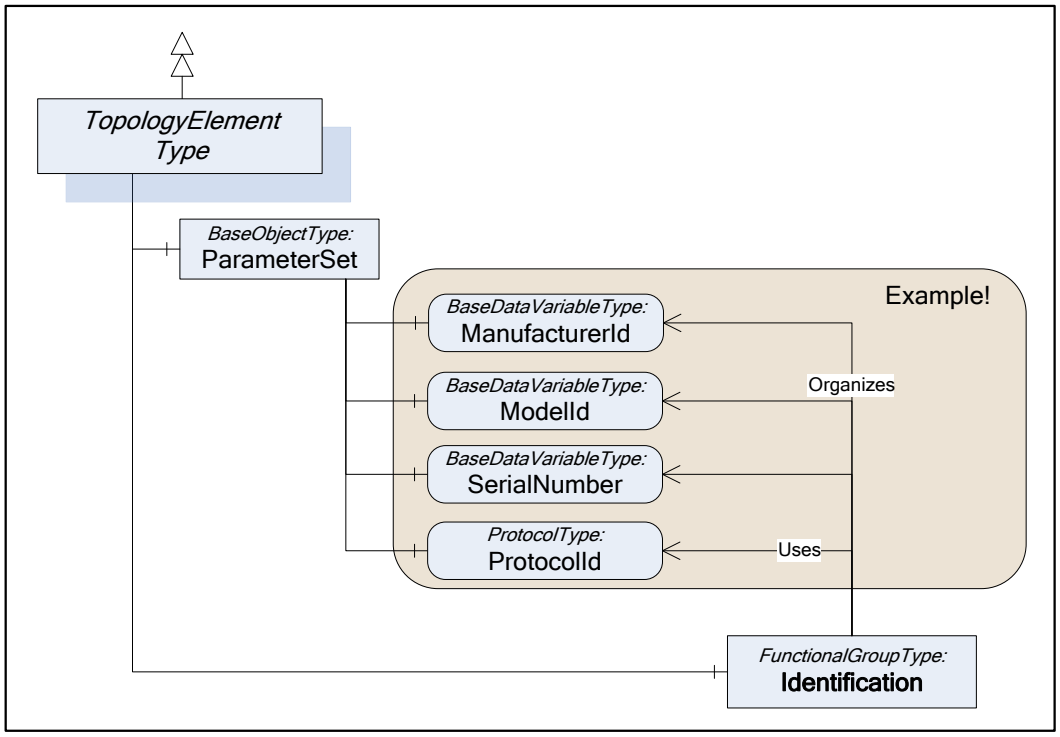

Figure 7 – Example of an *Identification Functional Group*

5.6 DeviceType

This *ObjectType* defines the structure of the *Device Object*. Figure 8 shows the *DeviceType*. It is formally defined in Table 4.

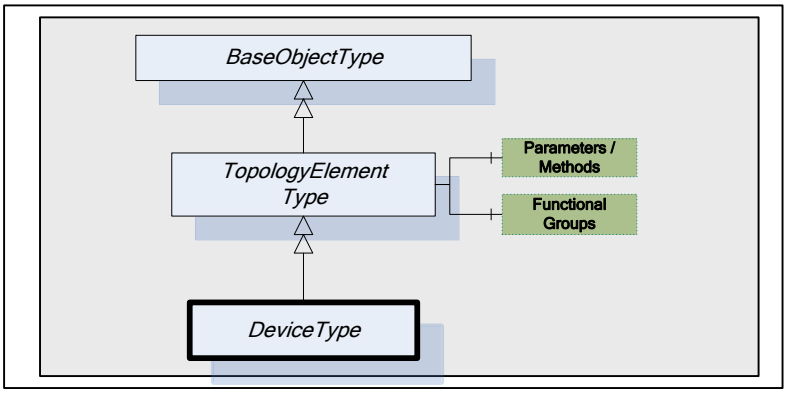

Figure 8 – *DeviceType*

Table 4 – *DeviceType* Definition

Attribute	**Value**				
BrowseName	DeviceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *TopologyElementType* defined in clause 5.3					
HasProperty	Variable	SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	RevisionCounter	Int32	PropertyType	Mandatory
HasProperty	Variable	Manufacturer	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	Model	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	DeviceManual	String	PropertyType	Mandatory
HasProperty	Variable	DeviceRevision	String	PropertyType	Mandatory
HasProperty	Variable	SoftwareRevision	String	PropertyType	Mandatory
HasProperty	Variable	HardwareRevision	String	PropertyType	Mandatory

The *DeviceType ObjectType* is abstract. There will be no instances of a *DeviceType* itself, but there will be instances of sub-types of this type. In this specification, the term *Device* generically refers to an instance of any *ObjectType* derived from the *DeviceType*.

Devices may have *Parameters* and *FunctionalGroup*s as defined for the *TopologyElementType*.

The following *Properties* provide a way for a *Client* to get common *Device* information. This is not necessarily a replacement for this information appearing in the ParameterSet and/or *FunctionalGroups* of the *Device*. Note that this specification does not make other than the following assumptions concerning the semantic. Organisations (e.g. *FieldBus* organisations) may further specify the contents.

Although all *Properties* are mandatory for all *DeviceType* instances, some of them may not be available for certain types of devices. In this case vendors shall provide the following defaults:

- *Properties* with DataType String: **empty string**
- *Properties* with DataType LocalizedText: **empty text field**
- *RevisionCounter Property*: **- 1**

The *SerialNumber Property* is a unique production number of the manufacturer of the *Device* manufacturer. This is often stamped on the outside of the *Device* and may be used for traceability and warranty purposes.

The *RevisionCounter* property is an incremental counter indicating the number of times the static data within the *Device* has been modified.

The *Manufacturer* property provides the name of the company that manufactured the *Device*.

The *Model* property provides the model name of the *Device*.

The *DeviceManual* property allows specifying address of user manual for the device. It may be a pathname in the file system or a URL (Web address).

The *DeviceRevision Property* provides the overall revision level of the *Device*.

The *SoftwareRevision Property* provides the revision level of the software/firmware of the *Device*.

The *HardwareRevision Property* provides the revision level of the hardware of the *Device*.

The *Description* attribute of the *Device Object* provides Information that serves to further identify, manage, locate, and/or explain the device whose contents are defined by the user.

Other organisations may specify additional semantics for the contents of these *Properties*.

Parameters like *ManufacturerId* (numeric identifier of the company), *ModelId* and *SubModelId* (numeric identifiers of the model) are not provided as *Properties*. Instead, these *Parameters* shall be included into the **Identification** *FunctionalGroup* defined in clause 5.5.

5.7 DeviceSet Entry Point

To promote interoperability of *Client*s and *Server*s, all instantiated *Devices* shall be aggregated in an *Object* called “DeviceSet”**.**

Figure 8 shows the *AddressSpace* organisation with this standard entry point.

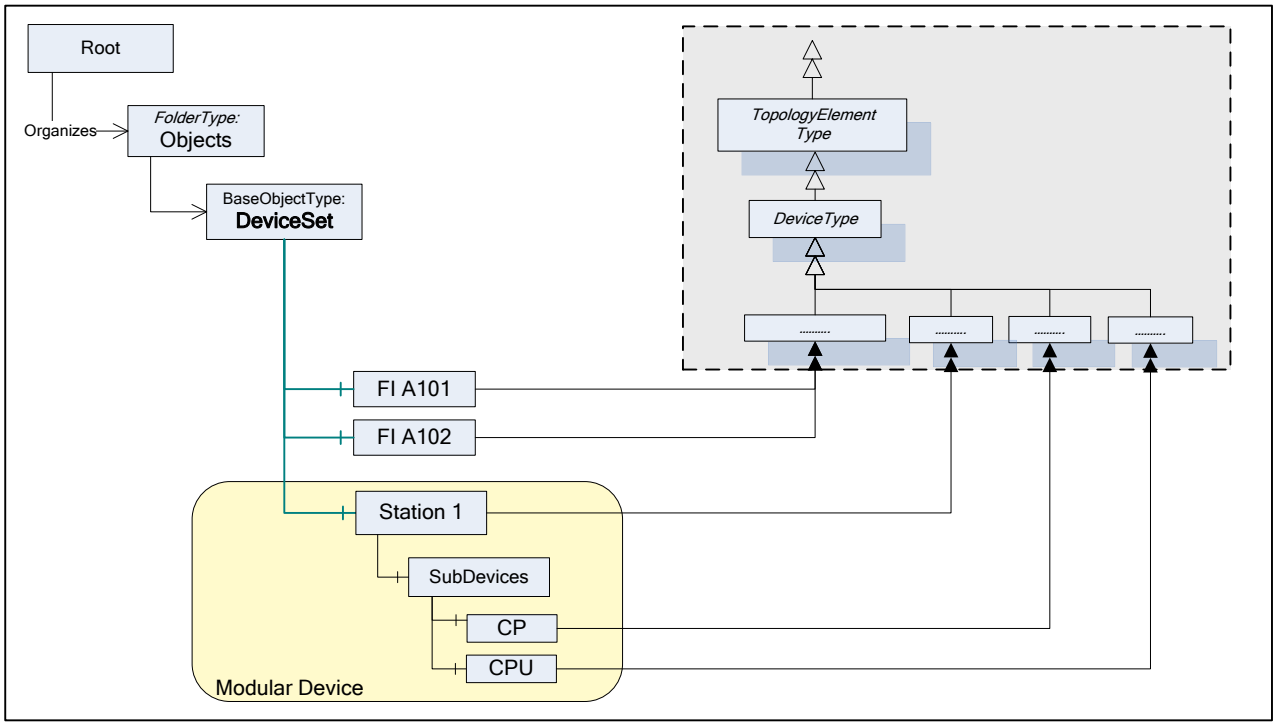

Figure 9 – Standard Entry Point for *Devices*

5.8 ProtocolType

A *Protocol ObjectType* represents a specific communication / *FieldBus* protocol implemented by a certain *TopologyElement*. The supported protocol shall be referenced with a *Uses Reference* in the **Identification** *FunctionalGroup*. For example an FF *Device* references an *FFBusType*; a HART *Device* references a *HARTBusType*; a PROFIBUS/ PROFINET *Device* references a PROFIBUS/ PROFINet Type.

Figure 10 shows the *ProtocolType* including some specific types. It is formally defined Table 5.

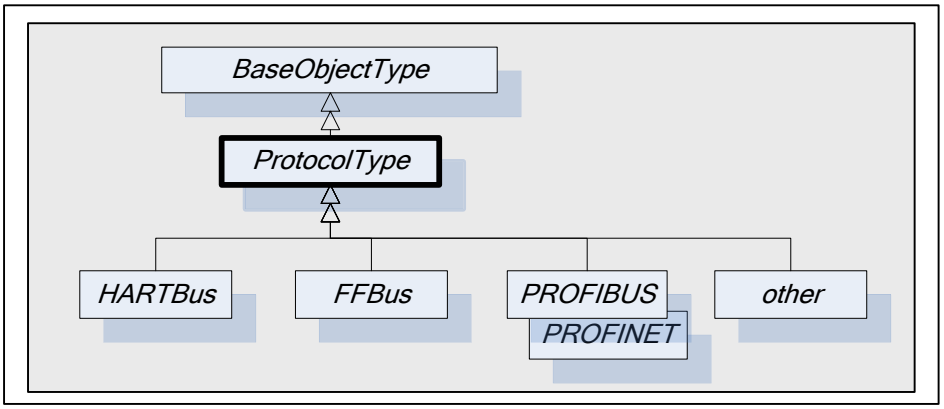

Figure 10 – *ProtocolType* Hierarchy

Table 5 – *ProtocolType* Definition

Attribute	**Value**				
BrowseName	ProtocolType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *BaseObjectType* defined in [UA Part 5]					

5.9 Uses ReferenceType

The *Uses ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *Organizes ReferenceType* and will be used to refer from the Identification *FunctionalGroup* to a *ProtocolType*.

The semantic indicates that the target *Node* is “used” by the source *Node* of the *Reference*. Figure 11 informally describes the location of this *ReferenceType* in the OPC UA hierarchy. Its representation in the *AddressSpace* is specified in Table 6.

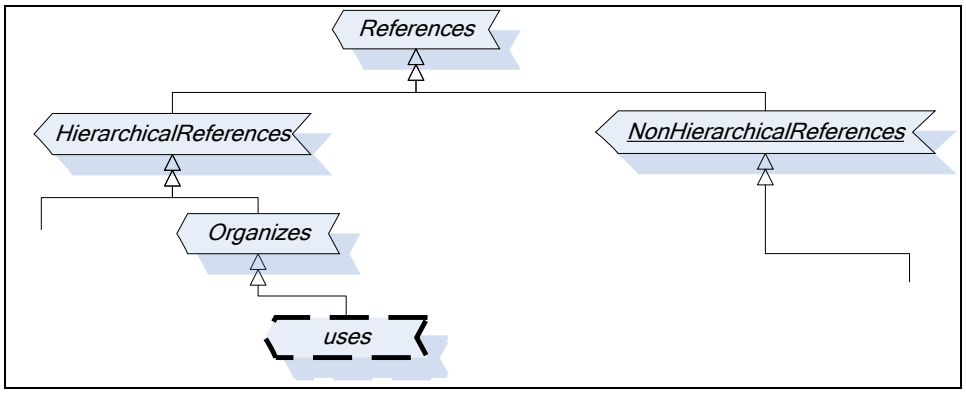

Figure 11 – ReferenceType Hierarchy for Device Integration

Table 6 – Uses ReferenceType

Attributes	**Value**		
BrowseName	Uses		
InverseName	UsedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of Organizes Reference*Type* defined in [UA Part 5].			

5.10 BlockType

This *ObjectType* defines the structure of a *Block Object*. Figure 12 depicts the *BlockType* hierarchy. It is formally defined in Table 7.

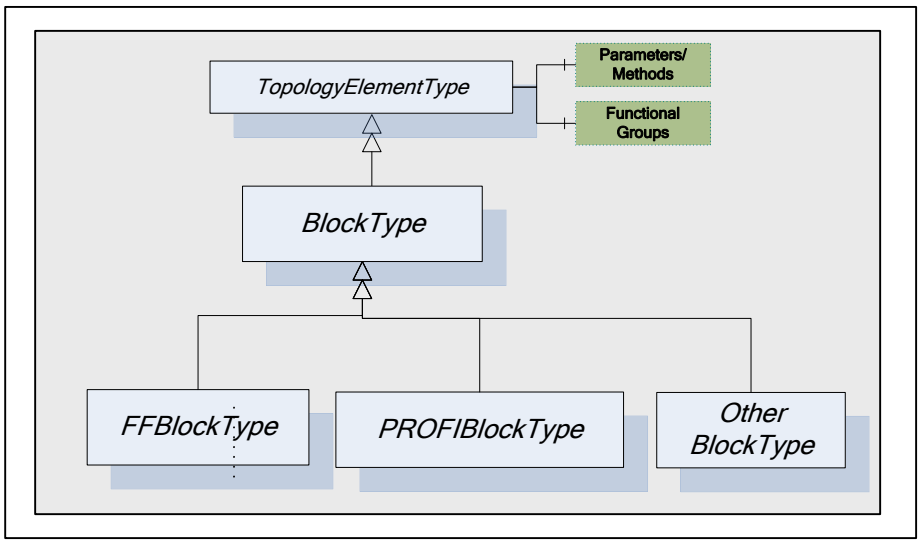

Figure 12 – *BlockType* Hierarchy

FFBlockType and PROFIBlockType are examples. They are not further defined in this specification. It is expected that industry groups, like the various *FieldBus* organisations, will standardize general purpose *BlockTypes*.

Table 7 – *BlockType* Definition

Attribute	**Value**				
BrowseName	BlockType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *TopologyElementType* defined in clause 5.3					
HasSubtype	ObjectType	FFBlockType			
HasSubtype	ObjectType	PROFIBlockType			
HasSubtype	ObjectType	CtrlProgramOrganizationUnitType	Defined in [UA Companion PLCopen].		
HasProperty	Variable	RevisionCounter	Int32	PropertyType	Mandatory
HasProperty	Variable	ActualMode	LocalizedText	PropertyType	Optional
HasProperty	Variable	PermittedMode	LocalizedText[]	PropertyType	Optional
HasProperty	Variable	NormalMode	LocalizedText[]	PropertyType	Optional
HasProperty	Variable	TargetMode	LocalizedText[]	PropertyType	Optional

BlockType is a sub-type of *TopologyElement* and inherits the elements for *Parameters*, *Methods* and *FunctionalGroups*.

The *BlockType* is abstract. There will be no instances of a *BlockType* itself, but there will be instances of sub-types of this *Type*. In this specification, the term *Block* generically refers to an instance of any sub-type of the *BlockType*.

The *RevisionCounter* is an incremental counter indicating the number of times the static data within the *Block* has been modified. A value of -1 indicates that no revision information is available.

The following *Properties* refer to the *Block Mode* (e.g. “Manual”, “Out of Service”).

The *ActualMode Property* reflects the current mode of operation the *Block* is able to achieve.

PermittedMode defines the modes of operation that are allowed for the *Block* based on application requirements.

The *NormalMode* is the mode the *Block* should be set to during normal operating conditions. Depending on the *Block* configuration, multiple modes may exist.

The *TargetMode* indicates the mode of operation that is desired for the *Block*. Depending on the *Block* configuration, multiple modes may exist.

5.11 Configurable Components

5.11.1 General Pattern

This section defines a generic pattern to expose and configure components. It defines the following principles:

- A configurable *Object* shall contain a folder called *SupportedTypes* that references the list of *Types* available for configuring components using *Organizes References*. Sub-Folders can be used for further structuring of the set. The names of these sub-folders are vendor specific.
- The configured instances shall be components of the configurable object.

Figure 13 illustrates these principles.

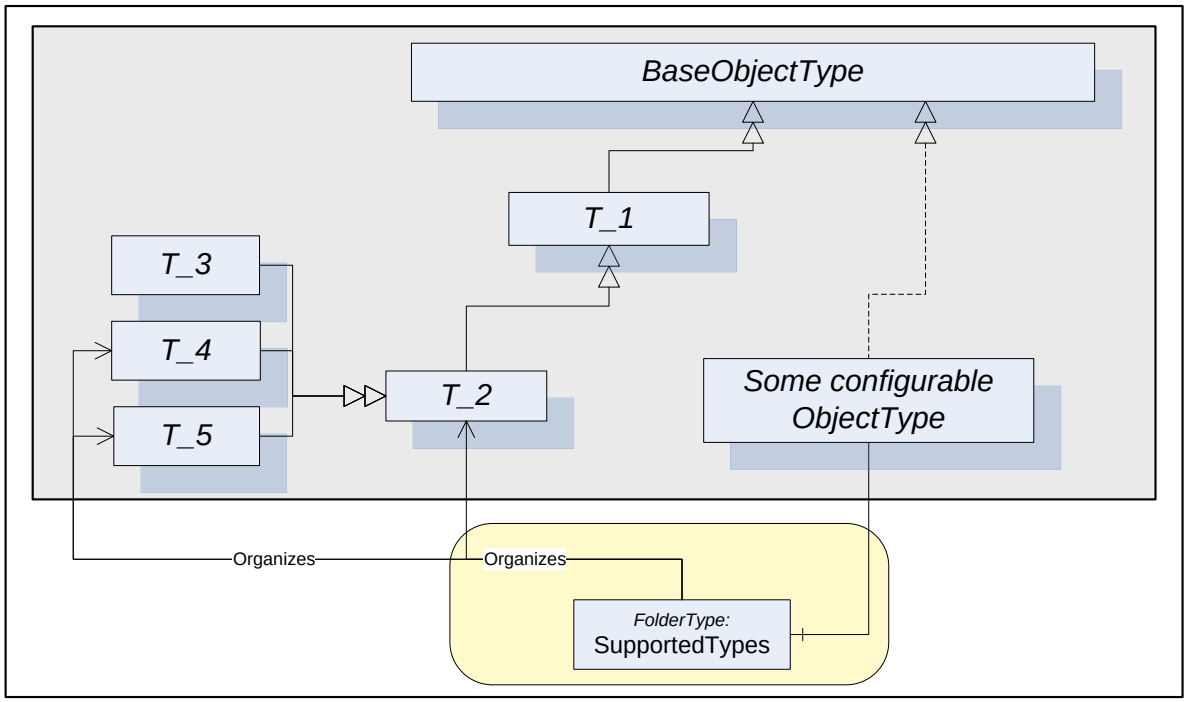

Figure 13 – *Configurable Component Pattern*

In some cases the *SupportedTypes* folder on the instance may be different to the one on the *Type* and may contain only a subset. It may be for example that only one instance of each *Type* can be configured. In this case the list of supported *Types* will shrink with each configured component.

5.11.2 ConfigurableObjectType

This *ObjectType* implements the Configurable Component pattern and is used when an *Object* or an instance declaration needs nothing but configuration capability. Figure 14 illustrates the *ConfigurableObjectType*. It is formally defined in Table 8. A concrete example is provided in section 6.

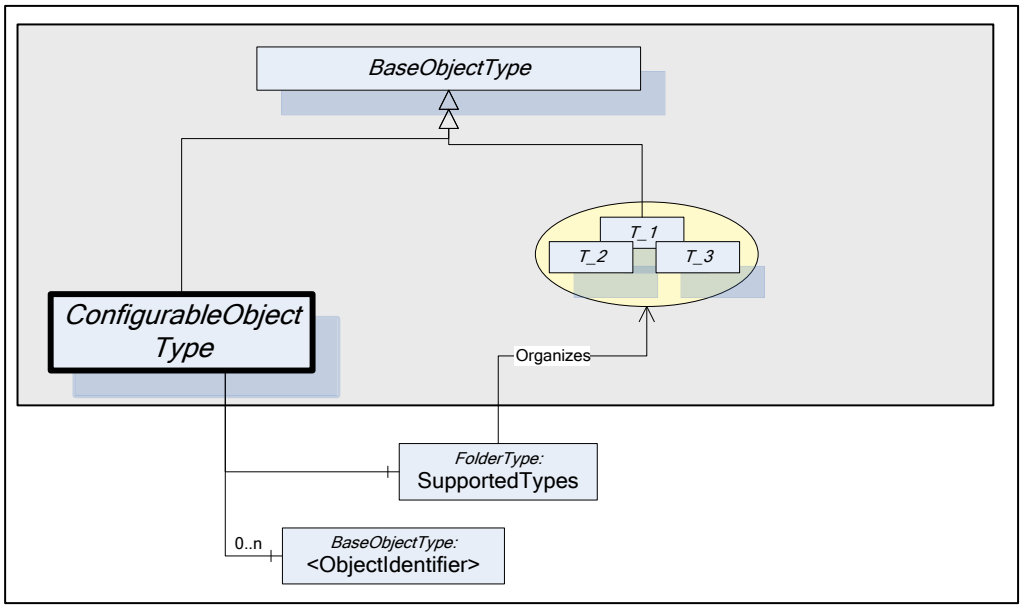

Figure 14 – *ConfigurableObjectType*

Table 8 – *ConfigurableObjectType* Definition

Attribute	**Value**				
BrowseName	ConfigurableObjectType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *BaseObjectType* defined in [UA Part 5]					
HasComponent	Object	SupportedTypes		FolderType	Mandatory
HasComponent	Object	<ObjectIdentifier>		BaseObjectType	Optional

The *SupportedTypes* folder is used to maintain the set of (sub-types of) *BaseObjectTypes* that can be instantiated in this configurable *Object* (the course of action to instantiate components is outside the scope of this specification).

The configured instances will be components of the *ConfigurableObject*.

6 Block Devices (BlockOriented DeviceType)

A block-oriented *Device* can be composed using the modelling elements defined in this specification. No specific *ObjectType* is needed. A block-oriented *Device* includes a configurable set of *Blocks*. Figure 15 shows the general structure of block-oriented *Devices*.

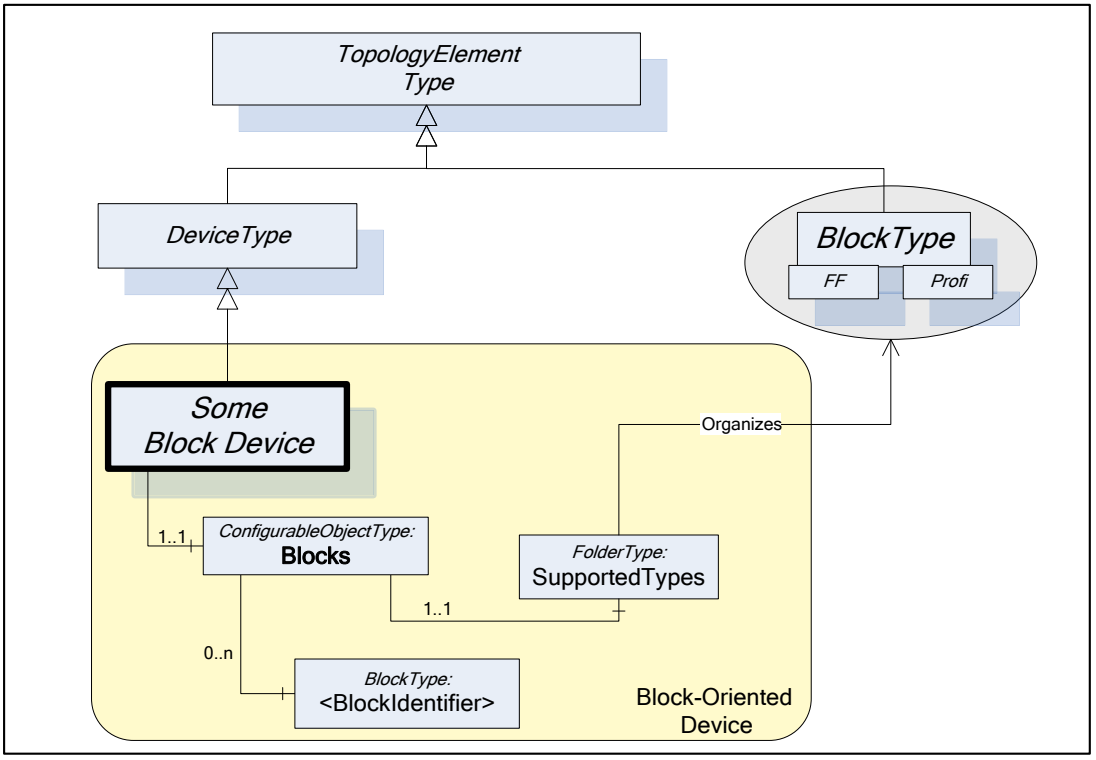

Figure 15 – Block-Oriented Device Structure

An *Object* called ***Blocks*** is used as container for the actual *BlockType* instances. It is of the *ConfigurableObjectType* which includes the *SupportedTypes* folder. The *SupportedTypes* folder for ***Blocks*** is used to maintain the set of (sub-types of) *BlockTypes* that can be instantiated. The supported *Blocks* may be restricted by the block-oriented *Device*. See clause 5.11.1 for the complete definition of the *ConfigurableObjectType*.

<BlockIdentifier> refers to the *Blocks* that have already been configured for this block oriented *Device*.

7 Profiles

Profiles are named groupings of *ConformanceUnits* as defined in [UA Part 7]. The term Facet in the title of a *Profile* indicates that this *Profile* is expected to be part of another larger *Profile* or concerns a specific aspect of OPC UA. *Profiles* with the term Facet in their title are expected to be combined with other *Profiles* to define the complete functionality of an OPC UA *Server* or *Client*

This specification defines OPC UA *Profile* facets for *Servers* or *Clients* when they plan to support the OPC UA *Devices* companion standard. They are described in the following paragraphs.

7.1.1 Device Server Facets

The following tables specify the facets available for *Servers* that implement the *Devices* companion standard. Table 9 describes *Conformance Units* included in the minimum needed facet. It includes the organisation of instantiated *Devices* in the *Server AddressSpace*.

Table 9 – *BaseDevice_Server_Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
DI Information Model	Support *Objects* that conform to the types specified in the Device companion standard. This includes in particular *Objects* of *DeviceType* and *FunctionalGroups*.	M
DI DeviceSet	Support the **DeviceSet** object to aggregate all *Device* instances.	M

Table 10 defines a facet for the identification *FunctionalGroup* of *Devices*. This includes the option of identifying the *Protocol*(s).

Table 10 – *DeviceIdentification_Server_Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
DI Identification	Support the **Identification** *FunctionalGroup* for *Devices*.	M
DI_Protocol	Support the *ProtocolType* and the *Uses Reference* to identify the supported protocol(s) for specific instances.	O

Table 11 defines extensions specifically needed for *BlockDevices*.

Table 11 – *BlockDevice_Server_Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
DI Blocks	Support the *BlockType* (or sub-types respectively) and the *Blocks Object* in some of the instantiated *Devices*.	M

7.1.2 Device Client Facets

The following tables specify the facets available for *Clients* that implement the *Devices* companion standard. Table 12 describes *Conformance Units* included in the minimum needed facet.

Table 12 – *BaseDevice_Client_Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
DI Client Information Model	Consume *Objects* that conform to the types specified in the *Device* companion standard. This includes in particular *Objects* of *DeviceType* and *FunctionalGroups*.	M
DI Client DeviceSet	Use the **DeviceSet** *Object* to detect available *Devices*.	M

Table 13 defines a facet for the **identification** *FunctionalGroup* of *Devices*. This includes the option of identifying the *Protocol*(s).

Table 13 – *DeviceIdentification_Client_Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
DI Client Identification	Consume the **Identification** *FunctionalGroup* for *Devices* including the (optional) reference to supported protocol(s).	M

Table 14 defines extensions specifically needed for BlockDevices.

Table 14 – *BlockDevice_Client_Facet* Definition

Conformance Unit	Description	Optional/ Mandatory
DI Client Blocks	Understand and use BlockDevices and their *Blocks* including *FunctionalGroups* on both device and block level.	M

Appendix A: Namespace and Mappings

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined by the OPC UA Devices Companion Standard. 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 *BrowseName* for the containing instance or type. A '_' character is used to separate each *BrowseName* in the path. Lets take for example, the *DeviceType ObjectType Node* which has the *SerialNumber Property*. The *SymbolName* for the *SerialNumber InstanceDeclaration* within the *DeviceType* declaration is: *DeviceType_SerialNumber*.

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

The CSV associated with this version of the specification can be found here:

http://www.opcfoundation.org/UADevices/2009/08/NodeIds.csv

The most recent set of *NodeIds* can be found here:

http://www.opcfoundation.org/UADevices/NodeIds.csv