



<AutomationML/>

**The Glue for Seamless
Automation Engineering**

**Application Recommendations:
Automation Project Configuration**

Document Identifier: AR APC, V 1.2.0

State: April 2020

©AutomationML consortium

Version 1.2.0, April 2020

Contact: www.automationml.org

Table of contents

Table of contents	3
List of figures	5
List of tables	6
1 Introduction.....	7
1.1 Basics.....	7
1.2 Scope	7
1.3 References.....	8
2 General notes regarding exchange of Automation Project Configuration data	9
2.1 Data exchange workflow.....	9
2.2 Possibilities of configuration.....	10
2.3 Recommended workflow.....	10
2.3.1 Providing an initial PLC project as basis for electrical engineering	10
2.3.2 ECAD engineering.....	10
2.3.3 PLC engineering.....	14
3 Automation Project Configuration data structures in AutomationML	15
3.1 Basic concept.....	15
3.1.1 Export from ECAD to AutomationML	15
3.1.2 Import from AutomationML into PLC.....	16
3.2 The neutral model: Automation Project Configuration data.....	16
3.2.1 Basic ideas	17
3.2.2 Contents of data exchange	19
3.2.3 Automation Project Configuration data exchange data model.....	19
4 Guideline for the use of the ECAD model in practical applications	29
5 Modelling of Automation Project Configuration data with AutomationML	30
5.1 RoleClassLibrary	30
5.1.1 AutomationProject	34
5.1.2 DeviceUserFolder.....	36
5.1.3 Subnet	36
5.1.4 Device	36
5.1.5 DeviceItem	38
5.1.6 TagTable	41
5.1.7 TagUserFolder	43
5.1.8 Node.....	43
5.1.9 CommunicationInterface	43
5.1.10 IoSystem	44
5.1.11 CommunicationPort.....	45
5.1.12 PowerPort.....	45
5.1.13 SensorPort	46
5.1.14 DeviceItemBusExtension	46
5.1.15 NodeBusExtension.....	47
5.1.16 CommunicationInterfaceBusExtension	47
5.2 InterfaceClassLibrary	48
5.2.1 Tag	50

5.2.2	Channel	51
5.2.3	CommunicationPortInterface.....	51
5.2.4	PowerPortInterface.....	51
5.2.5	SensorPortInterface	52
5.2.6	ModuleAssignment.....	52
5.2.7	Naming and Escaping	52
Appendix A	Roundtrip Engineering and Identification of Logical AR APC Objects.....	54

List of figures

Figure 1 – Automation Project Configuration between ECAD and PLC tool	9
Figure 2 – Data exchange workflow	10
Figure 3 – Example for PLC configuration and graphical placement	11
Figure 4 – Example for stations and bus data configuration	12
Figure 5 – Example for symbolic address configuration	12
Figure 6 – Example for error check	13
Figure 7 – Example for export from an ECAD tool	13
Figure 8 – Example for import into a PLC.....	14
Figure 9 – Example for result of import into a PLC.....	14
Figure 10 – Basic concept for ECAD-PLC data exchange	15
Figure 11 – Basic concept for ECAD-export (EPLAN example).....	16
Figure 12 – Basic concept for PLC-import (Step 7 example)	16
Figure 13 – Coupling CAE and PLC	17
Figure 14 – Objects and parameters of the Automation Project Configuration data exchange	20
Figure 15 – Objects and parameters of the Automation Project Configuration data exchange for extensions.....	21
Figure 16 – Procedure for use of ECAD in AutomationML	29
Figure 17 – AutomationProjectConfigurationRoleClassLib in AutomationML Editor view	30
Figure 18 – AutomationProjectConfigurationRoleClassLib as XML representation	34
Figure 19 – AutomationProjectConfigurationInterfaceClassLib in AutomationML Editor view	48
Figure 20 – AutomationProjectConfigurationInterfaceClassLib as XML representation	49

List of tables

Table 1 – Overview of AutomationML parts.....	7
Table 2 – Definition AutomationProject	34
Table 3 – Definition DeviceUserFolder	36
Table 4 – Definition Subnet.....	36
Table 5 – Definition Device	36
Table 6 – Definition DeviceItem	38
Table 7 – Definition TagTable	41
Table 8 – Definition TagUserFolder	43
Table 9 – Definition Node	43
Table 10 – Definition CommunicationInterface.....	44
Table 11 – Definition IoSystem	44
Table 12 – Definition CommunicationPort	45
Table 13 – Definition PowerPort	45
Table 14 – Definition SensorPort	46
Table 15 – Definition DeviceItemBusExtension.....	46
Table 16 – Definition NodeBusExtension	47
Table 17 – Definition CommunicationInterfaceBusExtension.....	47
Table 18 – Definition Tag.....	50
Table 19 – Definition Channel.....	51
Table 20 – Definition CommunicationPortInterface	51
Table 21 – Definition PowerPortInterface	52
Table 22 – Definition SensorPortInterface.....	52
Table 23 – Definition ModuleAssignment	52

1 Introduction

A very frequently occurring task within the planning process of production and automation systems is the exchange of automation project configuration information of automation system devices between ECAD and PLC systems. To avoid multiple engineering in the participating systems ECAD and PLC systems need an interface for sharing this information.

In case of beginning engineering in the ECAD tool certain rules must be observed to get the hardware information in the correct location in the PLC tool. In case of beginning engineering in the PLC tool non placed functions must be placed and operated in the ECAD tool.

This application recommendation describes these workflows and the method of hardware configuration modelling using AutomationML.

1.1 Basics

The data exchange format AutomationML which is standardising in the IEC 62714 standard is a neutral, free, and XML-based data format. It has been developed in order to support the data exchange between engineering tools in a heterogeneous engineering tool landscape.

Due to the different aspects of AutomationML the IEC 62714 consists of different parts.

Table 1 – Overview of AutomationML parts

Part / Document Identifier	Title	Description
Part 1 / WP Arch, V 2.0.0	Architecture and general requirements	This part specifies the general AutomationML architecture, the modelling of the engineering data, classes, instances, relations, references, hierarchies, basic AutomationML libraries and extended AutomationML concepts.
Part 2 / WP Lib V 2.0.0	Role class libraries	This part specifies additional AutomationML libraries.
Whitepaper / WP Comm V 1.0.0	Communication	This Whitepaper describes the modelling of Communication mechanisms in AutomationML
Whitepaper / WP eClass V 1.0.0	AutomationML and eCI@ss integration	This Whitepaper describes the integration of eCI@ss in AutomationML
Best Practice Recommendation / BPR MlingExp V 1.0.0	Multilingual expressions in AutomationML	This Whitepaper describes the handling of different texts for different languages in AutomationML
Best Practice Recommendation / BPR RefDes V 1.0.0	Modelling of Reference Designations	This Whitepaper describes the handling of reference designations following IEC 81346-1:2009-07 within AutomationML

Further parts may be added in the future in order to e.g. interconnect further data standards to AutomationML.

1.2 Scope

This application recommendation proposes a modelling method of automation project configuration data by means of the engineering data format AutomationML. It will describe the recommended use of role and interface classes as well as the recommended structures to be considered within the instance hierarchy of an AutomationML project.

1.3 References

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

Extensible Markup Language (XML) 1.0:2004, W3C Recommendation (available at <<http://www.w3.org/TR/2004/REC-xml-20040204/>>)

IEC 62424:2008, Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools

Whitepaper AutomationML Part 1 – AutomationML Architecture, November 2018

Whitepaper AutomationML Part 2 – AutomationML Role Libraries, October 2014

Whitepaper AutomationML – AutomationML Communication, September 2014

Whitepaper AutomationML – AutomationML and eCI@ss Integration, November 2017

Best Practice Recommendation Multilingual expressions in AutomationML, March 2017

Best Practice Recommendation Modelling of Reference Designations, September 2017

2 General notes regarding exchange of Automation Project Configuration data

ECAD tools and PLC tools have different views of automation system information. Whereas ECAD tools depict all electrical detail information of devices applied within automation systems in PLC tools only a logical compilation of the automation devices is used. So in ECAD tools there are defined e.g. devices which are involved in an automation systems, voltage connectors which are used for power supply of the devices, and wire types which are used to connect devices. But these are not used in PLC tools. On the other side in PLC tools there are device and control application specific conditions defined e.g. baud rates which are used within the communication connections, control code variables which are associated to control device inputs and outputs, and control application codes. But these are not needed in ECAD tools. Nevertheless, both types of tools have some information in common. For example, the wiring of a certain automation device to a PLC defines the address the device can be accessed within the PLC. This must be considered by development of import and export tools. Figure 1 shows the scope of this application recommendation.

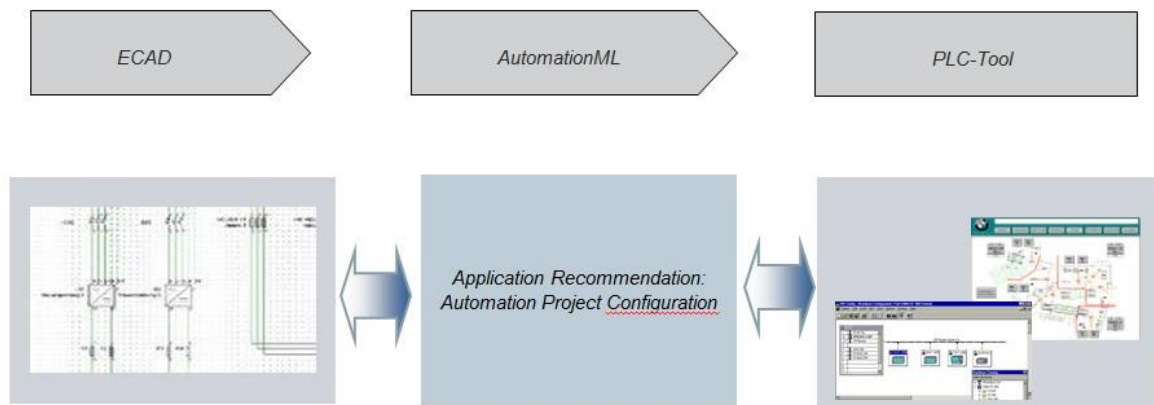


Figure 1 – Automation Project Configuration between ECAD and PLC tool

Beyond the named engineering tools for ECAD and PLC programming also other tools can be interested in the common data set of both tools. For example, tools for mechanical engineering (MCAD) can be interested in the devices to be wired and documentation tools can be interested in the wiring structure reached. Nevertheless, within this document only ECAD and PLC programming tools are considered knowing that more engineering tools can benefit from importing the modelled information. Systems containing drives may comprise of different aspects other than electrical configuration. Therefore, an own Application Recommendation “Drive for MCAD” (AR Drive MCAD) will propose a modelling method of mechanical aspects of drive configurations whereas a specific drive extension will handle the electrical configuration of drives. It will describe the recommended use of role and interface classes for drives as well as the recommended structures to be considered within the instance hierarchy of an AutomationML project.

2.1 Data exchange workflow

Usually, in a production system engineering process the construction phase in the PLC project will begin later than in the ECAD system because the completion of the ECAD documents is the base for the production of the control cabinet. The combination with the software within the plant and the following commissioning will not take place before all control cabinets are completed.

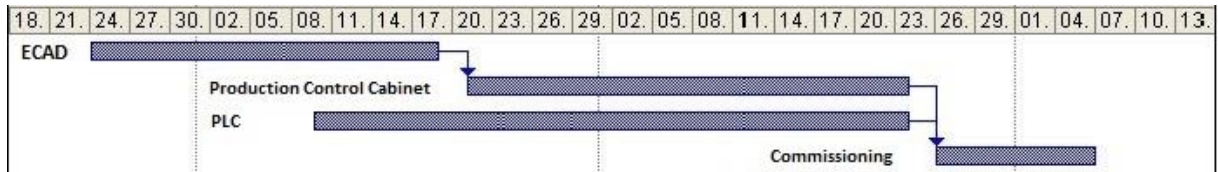


Figure 2 – Data exchange workflow

So the PLC engineer will usually attend later to the project than the ECAD engineer. Nevertheless at an early point of time (during ECAD engineering) the automation project configuration of the plant must be defined because the ECAD documents must be generated and the parts must be ordered.

2.2 Possibilities of configuration

ECAD systems normally can handle the components of different PLC manufacturers which have certain analogies from a point of view of electrical hardware. But additionally, there are system specific / manufacturer specific parameters. Therefore, only the engineering system of the PLC manufacturer can guarantee a complete and comfortable handling of all parameters of a hardware component. So, the configuration of the PLC system should be done as far as possible within the engineering system of the PLC manufacturer.

2.3 Recommended workflow

Accordingly to the described criteria in most cases the following workflow is established.

- Engineering the basic device configuration within the PLC project of the PLC programming tool and exporting it to ECAD tool
- Importing PLC project to ECAD tool, engineering of the ECAD project, and exporting the ECAD project to PLC programming tool
- Importing ECAD project into PLC programming tool and engineering of the PLC project

2.3.1 Providing an initial PLC project as basis for electrical engineering

If no ECAD project exists so far, the ECAD engineer first of all defines a raw project within the engineering system of the PLC manufacturer, the PLC programming tool. The ECAD engineer selects all needed components and defines the bus topology in close cooperation with the PLC engineer who has to implement the requested functions later on. This close cooperation ensures a high consistency regarding the selected hardware components. The automation project configuration will be exported from the engineering system of the PLC manufacturer and imported into the ECAD tool.

2.3.2 ECAD engineering

Based on the existing ECAD project the ECAD engineer executes the complete hardware construction, sometimes with slight adaptations. During this process the symbolic names for variables, tags or signals can be defined too. So the PLC configuration is done under the following conditions:

- PLC configuration can be imported from PLC programming system
- Configuration via graphical placement on overview page or navigator
- PLC-device selection carried out from ECAD database
- Drag&Drop on pages from navigator

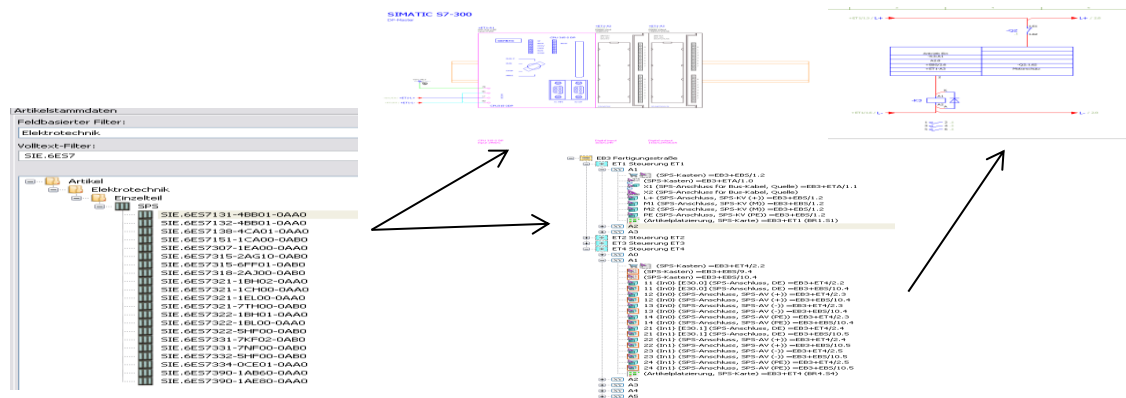


Figure 3 – Example for PLC configuration and graphical placement

When the PLC configuration is completed the stations and bus-data are engineered. This is normally done by manual assignment of cards to CPUs and devices to bus via device properties.

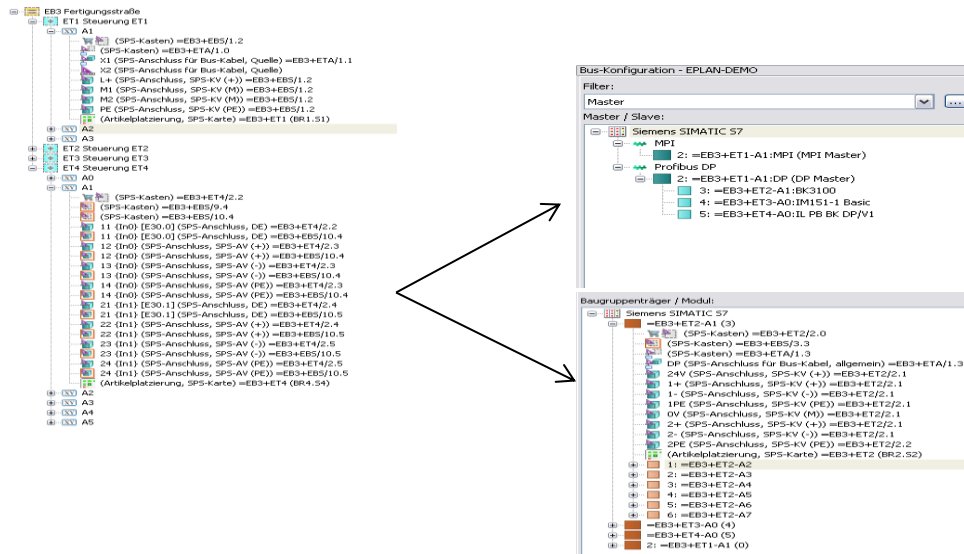


Figure 4 – Example for stations and bus data configuration

In the next step the tag list is to be engineered, i.e. a list of tags, variables or signals (symbolic address) of hardware related tags:

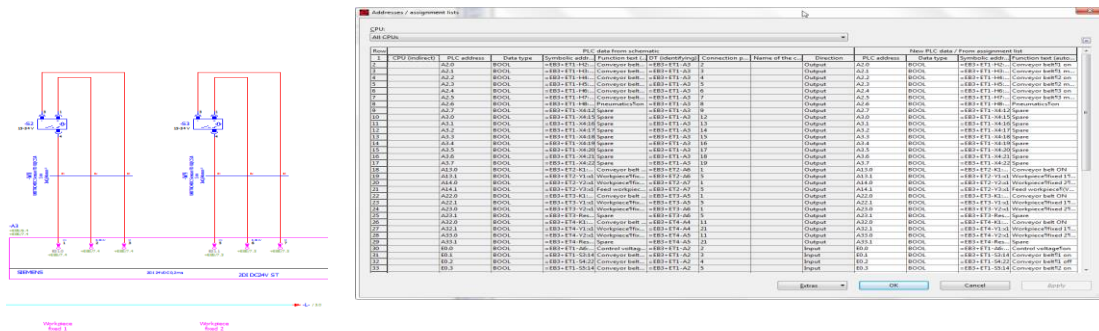


Figure 5 – Example for symbolic address configuration

Finally a simple error check should be performed, e.g.:

- Double usage of I/O / bus or symbolic addresses
- Missing addresses.
- Simple rules, e.g. slot 3 reserved for IM-module

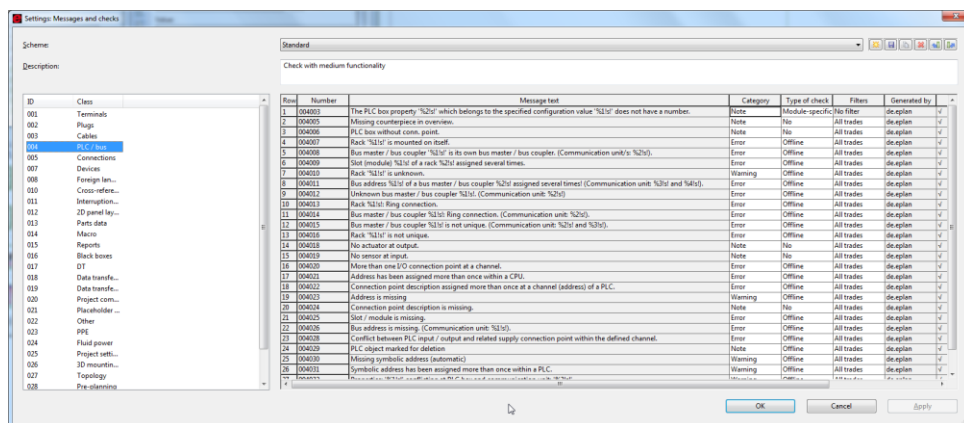


Figure 6 – Example for error check

Now the ECAD project can be exported to AutomationML dependent from the implementation in the ECAD tool and imported to the PLC programming tool.

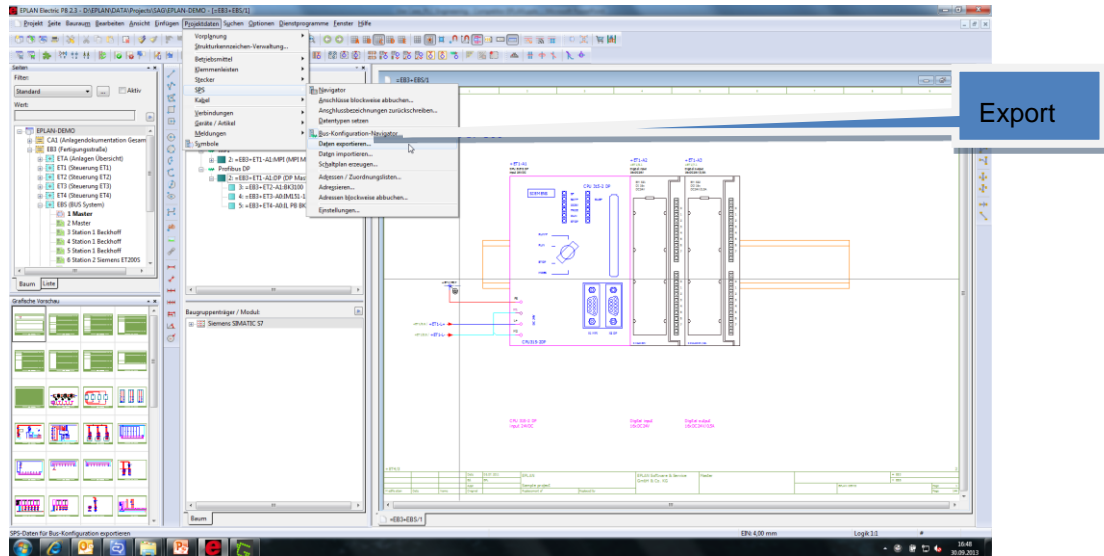


Figure 7 – Example for export from an ECAD tool

2.3.3 PLC engineering

At a later point in time the PLC programmer will begin the engineering based on the already developed ECAD project. So at this point of the engineering process the export function of the ECAD system and the import function of the PLC system should be used to check possible changes and verify the equality of both configurations.

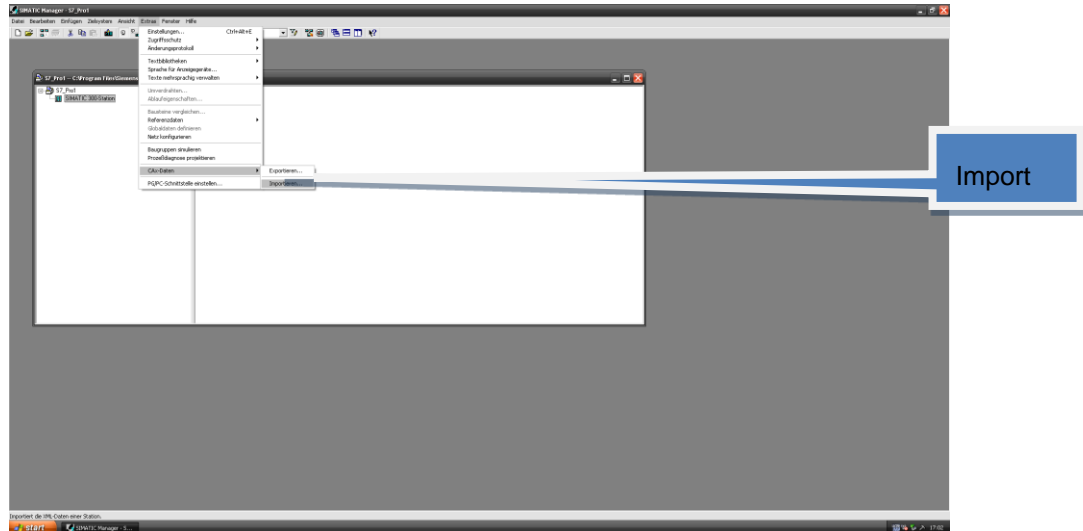


Figure 8 – Example for import into a PLC

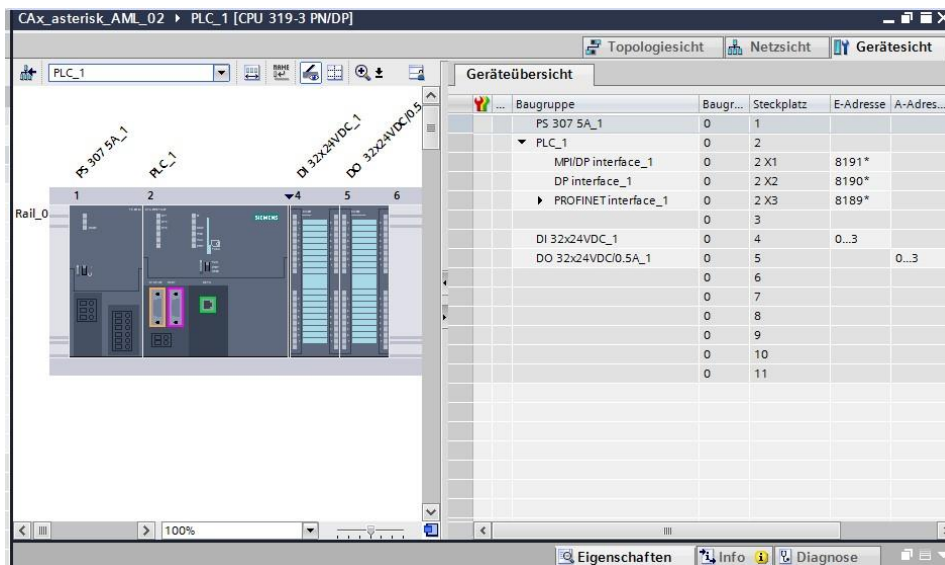


Figure 9 – Example for result of import into a PLC

3 Automation Project Configuration data structures in AutomationML

In the following chapter a concept is defined how Automation Project Configuration data can be represented in AutomationML.

3.1 Basic concept

For using AutomationML as a neutral exchange format for Automation Project Configuration data the PLC-specific interfaces of the different PLC manufacturers must be decoupled. This guarantees an independence of the further development of PLC-tools as well as of the further development of ECAD tools. Furthermore the transformation and implementation should be as easy as possible for ECAD and PLC-vendors likewise. Therefore already existing models should be used as far as possible.

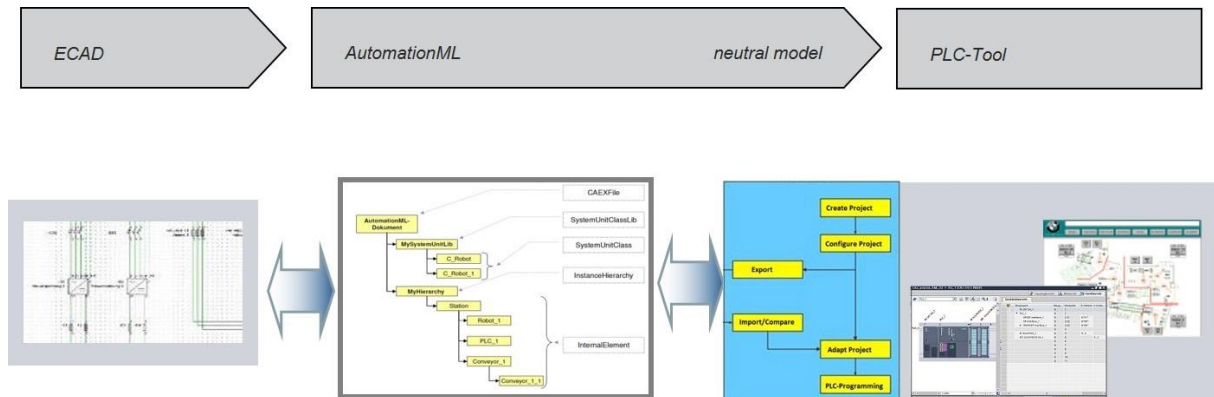


Figure 10 – Basic concept for ECAD-PLC data exchange

Using a neutral model allows

- definition of PLC-Tool independent roles in AutomationML
- definition of PLC-specific SystemUnitClasses for different ECAD- and PLC-Tools / vendors in AutomationML
- definition of PLC-specific InterfaceClasses in AutomationML

3.1.1 Export from ECAD to AutomationML

The following figure shows the detailed export of Automation Project Configuration data based on an EPLAN example:

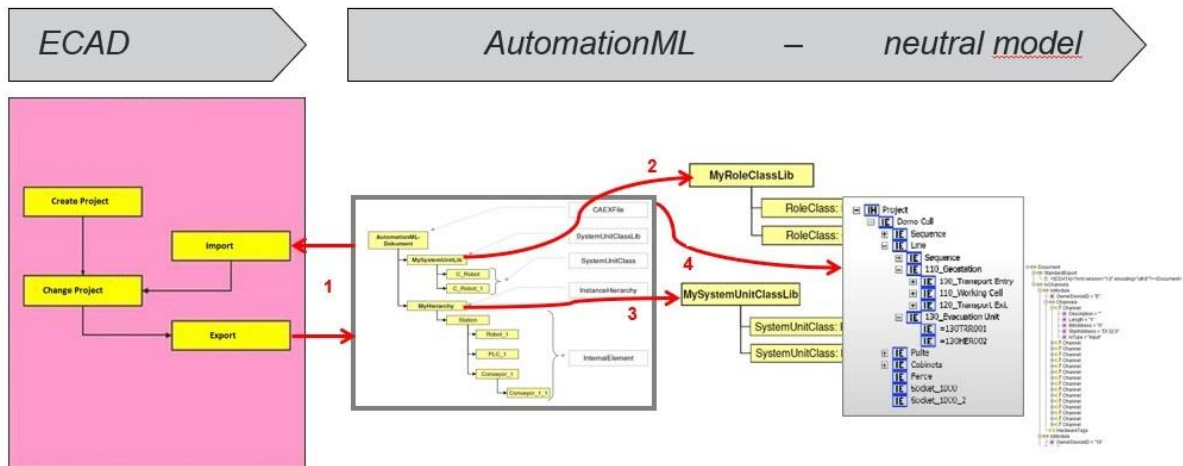


Figure 11 – Basic concept for ECAD-export (EPLAN example)

1. Export / Import of Automation Project Configuration data from / to AutomationML
2. Manufacturer independent roles in AutomationML
3. Neutral SystemUnitClasses in AutomationML
4. Topology in AutomationML (neutral model)

3.1.2 Import from AutomationML into PLC

The following figure shows the detailed import of Automation Project Configuration data based on an Step 7 example:

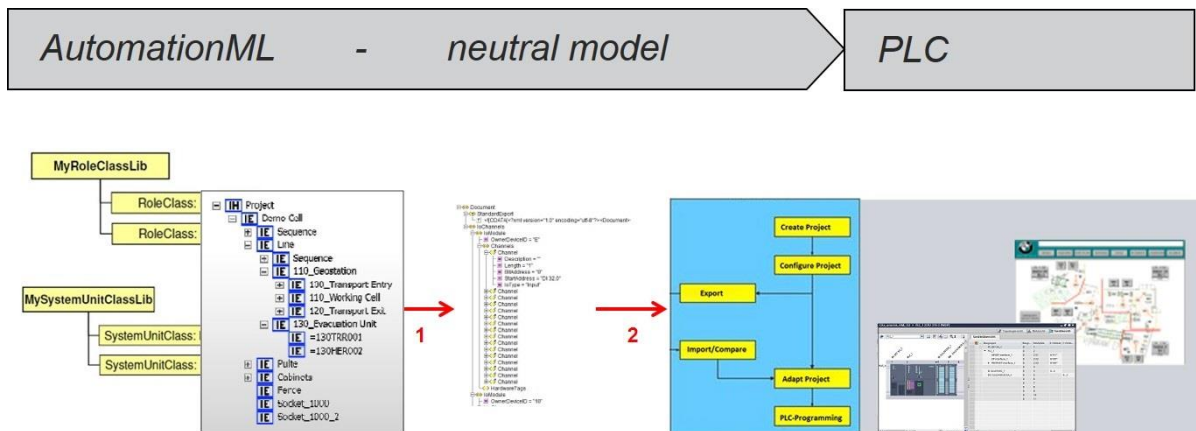


Figure 12 – Basic concept for PLC-import (Step 7 example)

1. Import of ECAD data from AutomationML (neutral model)
2. Import from neutral model into manufacturer specific PLC tool (Example S7)

3.2 The neutral model: Automation Project Configuration data

The aim is to support the engineering workflow between ECAD systems and PLC engineering systems. Providing standardized interfaces for the data exchange between PLC and ECAD systems are mandatory. The interfacing to ECAD systems has as further target group all ECAD manufacturers:

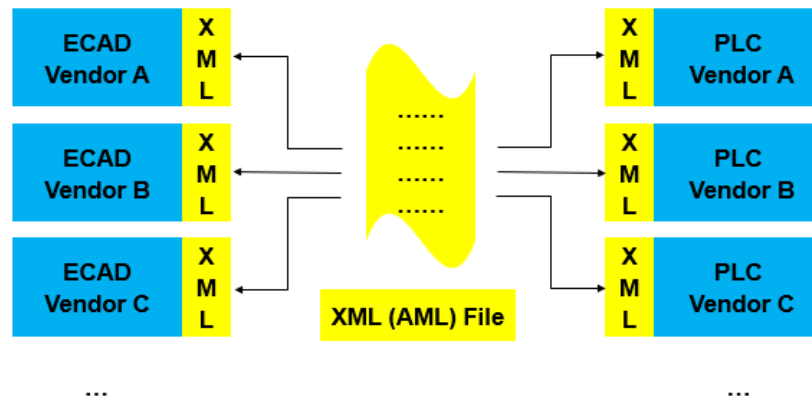


Figure 13 – Coupling CAE and PLC

So, we must consider the different ECAD systems and CAE manufacturer. Therefore, the term “ECAD” stands for the different CAE, E-CAD and E-CAE formats depending on the different existing ECAD systems. Based on this the implementation of a neutral “proxy ECAD”-format in AutomationML is defined in the following chapter. Furthermore, the already existing concepts of the leading ECAD-Tool manufacturer and PLC manufacturer shall be considered to ensure an “as easy as possible” implementation of this neutral model for all ECAD and PLC manufacturer.

3.2.1 Basic ideas

The Automation Project Configuration data can be modelled by using AutomationML. Therefore, the modelling methodology is based on the concepts of AutomationML topology modelling using CAEX defined in Part 1 of the AutomationML standard. Additional provisions are added to the basic definitions to fulfil the special requirements that arise from data exchange with ECAD tools. The Automation Project Configuration data modelling methodology enables the development of a self-containing model. No dependencies to other models are mandatory.

For modelling of Automation Project Configuration data a vendor neutral Automation Project Configuration data structure will be defined. It represents in its base structure a neutral object model of PLC systems.

The data exchange is based on complete information about the objects. This means that the Automation Project Configuration data always holds the complete data of the object itself and not only delta information.

The export/import granularity is at the level of hardware stations as PLC engineering systems always operate at a station level. The export/import will either support exchange of data referring to one or more stations (e.g. a complete project) or only parts of a station (e.g. one single module). The requirement here always is that the “environment“, the part lives in, is also part of the data exchange.

Only PLC hardware configuration information of automation devices including some relevant parameter and symbols/tags related to the hardware objects are in scope of this data exchange. Additionally information about the networks these hardware configurations are connected to is part of this exchange format.

Only a subset of all data provided by PLC hardware configuration is relevant for data exchange with ECAD systems. Due to the electrical view of the plant handled by ECAD tools, these tools can only deliver a very general subset of information belonging to the PLC hardware objects. Specific parameter settings are the domain of PLC specific hardware-configuration tools and the specific object managers. They only can be handled in the manufacturer specific tool.

Besides the standard devices in the PLC hardware catalogues, there are some types of device items that need additional descriptions. Examples for this are GSD or GSDML descriptions. Devices and

device items like norm slaves or GSDML based IODevices can only be instantiated in a PLC configuration if the appropriate description / package is installed. It is the responsibility of the PLC programmer to make sure that the correct and up to date device item information (GSD, etc.) is available. But the AutomationML based ECAD data exchange file can provide information about the needed device item information.

Some of the ECAD systems are capable to provide that information about a needed description file. Others also can provide the file itself. Therefore, it is allowed to transmit this additional data from the ECAD system to PLC engineering system. Files are expected to be delivered e.g. as a zip file and are unpacked into the same directory as the import file. Thus, it should be possible to specify and reference the file in the data exchange file. The user of a PLC system, who is importing the file, can expect to find some or all needed descriptions in the same directory as the import-file. The reference to the description file is inserted as separate properties for the module. It is an anchor to the item description.

3.2.2 Contents of data exchange

An analysis of the already existing proprietary XML-based data exchange files of the leading PLC and ECAD manufacturers regarding the Automation Project Configuration data showed that all data to be exchanged can be grouped in three major categories:

1. HW Data:

These are data concerning parts or devices like a central rack, a slave or a switch. Therefore mostly the term “device” is used for this group of parts. Within these devices there are other devices or device items like racks, CPUs, power supply, I/O Modules, submodules. Therefore mostly the term DeviceItem is used for this group of parts. Additional device items like routers, switches, hubs, repeaters will be supported by the export format. Devices are often grouped in a hierarchical “folder” structure.

2. Symbols / Tags:

Exported and imported are “symbols” and “tags” assigned to a device item. Only hardware oriented symbols/tags are considered here. The symbols/tags are exported with the controller target device item (i.e. the CPU) and not with other device items they might refer to (e.g. an I/O module). Like devices also the tags are often grouped in “tag tables” and in a hierarchical “folder” structure.

3. Networks:

Networks are modelled right below the project as global subnet objects. The link between a network and the device items are modelled as a reference to the subnet object. The network parameters are stored at the network object. The parameters concerning a network interface of a given device item, attached to a network, are stored in a node object at that device item. The communication is often regulated using “channels”, “ports” and “interfaces”.

Additionally the “Whitepaper AutomationML Part 5 – AutomationML Communication” already defines an XML based methodology for communication system information exchange among engineering tools developed by AutomationML e.V. These methods shall also be considered when modelling Automation Project Configuration data.

3.2.3 Automation Project Configuration data exchange data model

The consideration of all these mentioned and already existing models leads to the following basic Automation Project Configuration data exchange diagram:

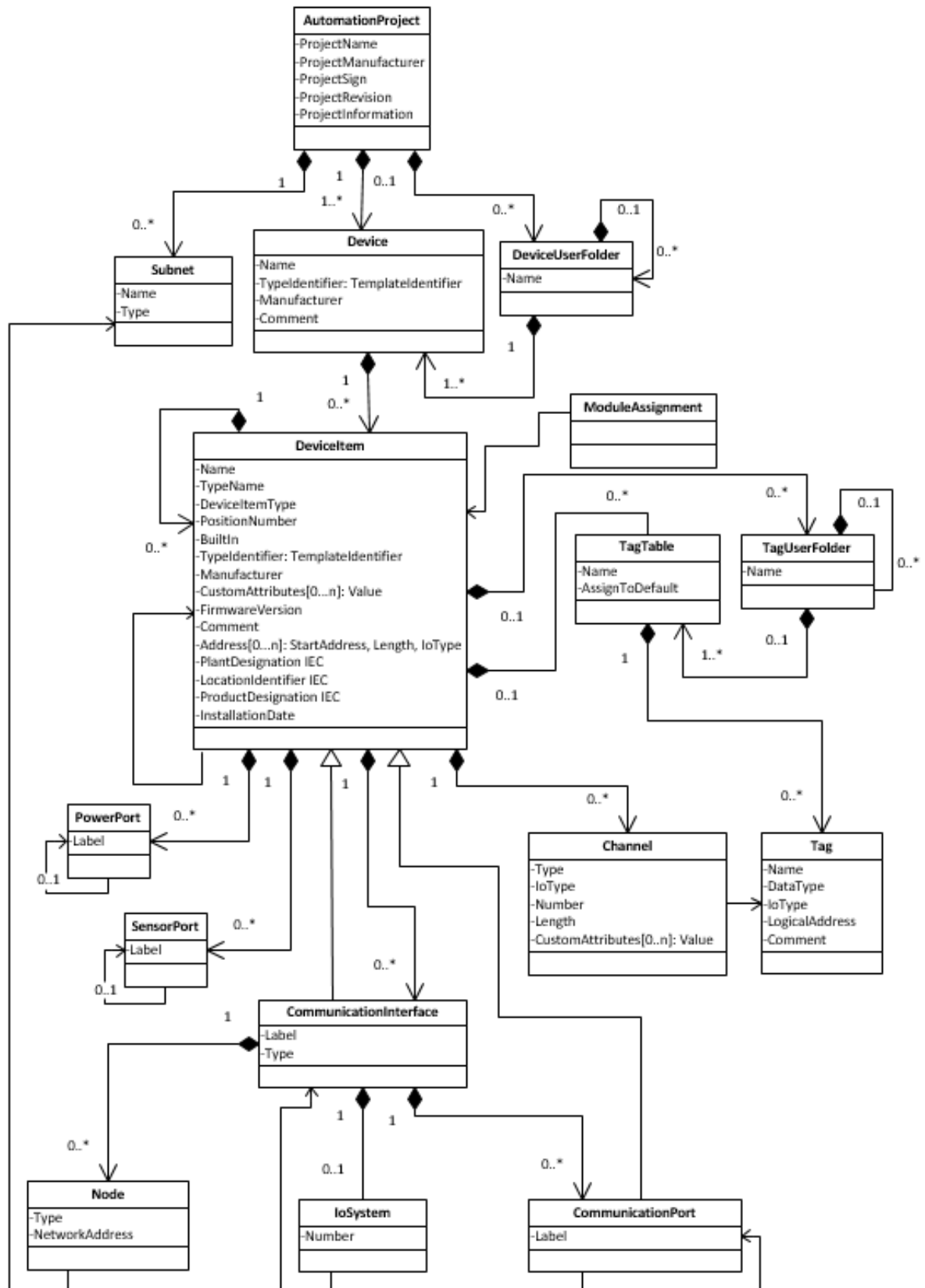


Figure 14 – Objects and parameters of the Automation Project Configuration data exchange

As this document also serves as a basis for bus specific definitions and extension which are defined in separate documents the abstract RoleClasses “DeviceItemBusExtension”, “NodeBusExtension” and “CommunicationInterfaceBusExtension” are defined to prepare an easy implementation for future extensions.

The following figure shows these abstract ExtensionRoleClasses for reasons of clarity only in the area of the parent objects.

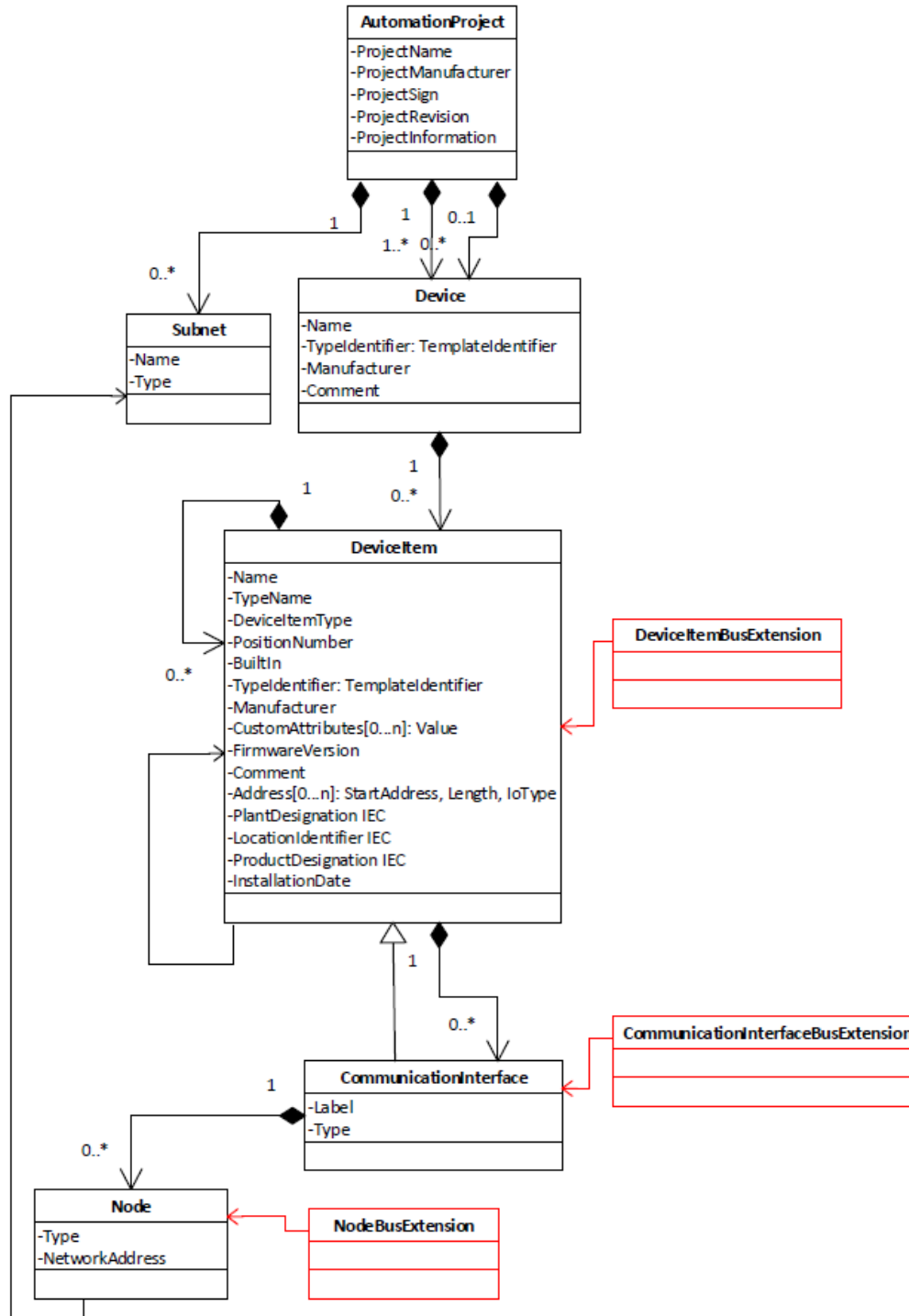


Figure 15 – Objects and parameters of the Automation Project Configuration data exchange for extensions

The AutomationML export of Automation Project Configuration data is based on the use of an InstanceHierarchy covering the exported Automation Project Configuration data. The InternalElements of this instance hierarchy will reference appropriate elements in RoleClass Libraries, SystemUnitClass Libraries, and InterfaceClass Libraries.

The objects and parameters shown in the figure above are described as follows. All objects will be modelled as role classes or interface classes derived from classes defined in Whitepaper AutomationML Communication and completed with additional attributes already used in the tool landscape of PLC manufacturers. Additional parameters can be defined using eCI@ss integration mechanisms as described in “Whitepaper AutomationML – AutomationML and eCI@ss Integration”. Objects and attributes which are not defined by AR APC might be ignored according “AutomationML Whitepaper – Architecture and general requirements” and will not be kept in a roundtrip scenario. Depending on the different communication systems and bus systems some objects may contain additional, unused or restricted attributes. These attributes and the bus specific parameters are described in separate bus specifications. Please refer to these bus specifications for more information.

Some attributes are defined based on strings. All strings are defined in the corresponding definitions exactly to the related spelling (usage of small letters, capital letters...). But due to error tolerance all importing tools should also support a tolerant interpretation (mixed spelling) of these strings as far as possible.

3.2.3.1 AutomationProject

An AutomationProject object represents the project from which the export arises. It aggregates all other objects below. The standard parameter for a Project are its “name” (string), the project manufacturer (string), the project sign (string) of the manufacturer, the project revision number (string), and a project information (string) hosting a comment to the project.

3.2.3.2 DeviceUserFolder

A DeviceUserFolder supports the structure of a device within a project. The only and one standard parameter for a DeviceUserFolder is its “name” (string)

3.2.3.3 Subnet

A Subnet object is responsible for storing and managing properties and functionality of networks like Ethernet, PROFIBUS, MPI, etc. A subnet is defined by the logical availability of all subnet participants. All subnet participants have different, unambiguous addresses. The standard parameters of a Subnet object are listed below:

- Name (string):
Name of the Subnet
- Type (string):
The type of the subnet (e.g. PROFIBUS, MPI...). The type shall be defined in the bus specific description.
- A Subnet has exactly one LogicalEndpoint to connect the subnet to Nodes.

3.2.3.4 Device

A Device object represents a collection in which the individual HW objects of a slave or rack, including the slave or rack HW item, are brought together. Therefore, a Device is a Device Item container that serves as a collection of Device Items (in particular hardware items). A Device has to have a unique name within an automation project. A device can be:

- a central configuration with some racks within (Automation system station with central and extension racks)
- a fix combination of CPU and some I/O modules (e.g. C7),
- a PC station where the PC represents a device,

- a field device,
- a switch

The standard parameters of a Device object are listed below:

- Name (string):
Name of the Device
- TypeIdentifier (string):
Identifier of the device type. An optional additional sub attribute "TemplateIdentifier" references the path of a library element.
- Manufacturer (string):
Additional information to describe the manufacturer of the device.
- Comment (string):
An optional comment for the device.

3.2.3.5 DeviceItem

A DeviceItem is aggregated by a Device and represents a generic class for HW modules and submodules (CPU, I/O module, rack, etc.). Whereas a Device represents the logical bracket, the DeviceItems represent more the physical hardware objects.

A DeviceItem can be plugged in another Device Item (e.g. CPU within a rack, submodule within a module). The relative position to the father object is defined by the PositionNumber.

A DeviceItem can also be built in another DeviceItem. These DeviceItems can model a fix combination that cannot be broken up (e.g. C7). The standard parameters of a DeviceItem object are listed below:

- Name (string):
Name of the DeviceItem
- TypeName (string):
Additional type information. Not mandatory but useful for user in case of error.
- DeviceItemType (string):
Classification of the DeviceItem (e.g. CPU)
The DeviceItemType is an additional information that may be useful for user in case of error.
 - Customized (boolean):
The subattribute "Customized" indicates if the DeviceItemType contains vendor specific information (Customized = "true") or not (Customized = "false"). If the attribute is omitted or set to "false" the DeviceItemType contains already standardized information (e.g. CPU).
- Manufacturer (string):
Additional information to describe the manufacturer of the DeviceItem.
- CustomAttributes (ListType):
Additional manufacturer specific information for the deviceItem. Specifies manufacturer specific property names and values.
- PositionNumber (int):
Slot number where this DeviceItem is plugged in.
- BuiltIn (Boolean):
Flag indicating that this module is a build-in part of another module. This module is automatically created because it is a fixed part of the other module. If omitted this parameter defaults to false.
- TypeIdentifier (string):
Identifier of the device item type. An optional additional sub attribute "TemplateIdentifier" references the path of a library element.

- **FirmwareVersion (string):**
Specifies the firmware version of e.g. a CPU and might be needed to identify the module correctly (sometimes the order number is not sufficient).
- **Comment (string):**
An optional comment for the module.
- **Address (OrderedListType):**
Address information of device item within device. Most modules have address ranges assigned. There may be e.g. address ranges for input, output channels which are described by their start value and length. The Address defines the start, length and IO-type. It is modelled as an ordered list of address parameters. The order is required for identifying the correct sub device item in case of multiple start addresses for one module. The subparameters are listed below:
 - **StartAddress (int):**
Start of the Address
 - **Length (int):**
Total width of the module (vendor specific). In the most cases it corresponds to the width of all channels.
 - **IoType (string):**
Input or Output.
 - **BitOffset (int):**
Start of BitAddress within a Byte
- **InstallationDate (dateTime):**
Installation Date of the device item.

***Note:** In addition to the named standard documents attributes can be added enabling the representation of reference designations following IEC 81346. The following attributes will give three possible examples.*

- **PlantDesignation IEC (string):**
Plant designation for this device item. The PlantDesignation is a product oriented reference designation following "IEC 81346-1:2009-07#5.3 - Function-oriented structure"
- **LocationIdentifier IEC (string):**
Location designation for this device item. The LocationIdentifier is a location oriented reference designation following "IEC 81346-1:2009-07#5.5 - Location-oriented structure".
- **ProductDesignation IEC (string):**
Product designation for this device item. The ProductDesignation is a product oriented reference designation following "IEC 81346-1:2009-07#5.4 - Product-oriented structure".

3.2.3.6 TagTable

A TagTable supports the structuring of tags. The standard parameters of a TagTable object are listed below:

- **Name (string):**
Name of the TagTable
- **AssignToDefault (Boolean):**
While importing if the TagTable has an attribute 'AssignToDefault' with 'True' value, then all the Tags inside will be imported to an existing default TagTable. In this case the name of the TagTable is ignored by the importing tool. By default, False value is assumed for 'AssignToDefault' attribute if it does not exist while importing.

3.2.3.7 TagUserFolder

The TagUserFolder supports the structuring of TagTables within a DeviceItem. The only and one standard parameter for a TagUserFolder is its “name” (string).

3.2.3.8 Tag

A Tag represents the symbolic name of an I/O data. It provides the logical view on the Channel of a module and is referenced by the associated channel directly.

Tags can only be aggregated by a Tag Table of a CPU. The CPU is represented by a concrete Device Item. The standard parameters of a Tag object are listed below:

- Name (string):
Name of the Tag
- DataType (string):
Type of the data
 - Customized (Boolean):
The subattribute “Customized” indicates if the DataType contains vendor specific data types (Customized = “true”) or not (Customized = “false”). If the attribute is omitted or set to “false” the DataType contains already standardized information according IEC 61131 (e.g. BOOL, BYTE, WORD).
- IoType (string):
Input or Output
- LogicalAddress (string):
Logical Address specifies the address of the tag.
- Comment (string):
An optional comment specified for the tag

Tags without assigned channels and channels without assigned tags are possible (incomplete engineering)

3.2.3.9 Channel

A Channel is part of an IO module and represents the process interface (e.g. digital or analogue input/output). A channel is part of the DeviceItem which represents the IO module and can only be used in a DeviceItem. The channel refers to tags using a link. The standard parameters of a channel object are listed below:

- Type (string):
Analog or Digital
- IoType (string):
Input or Output
- Number (int):
Number of the channel, starting with 0
- Length (int):
Width of the channel (e.g. 1 for bit, 8 for byte, 16 for word)

A channel references with a LinkToTag to the associated Tags which are stored at a CPU DeviceItem.

3.2.3.10 CommunicationInterface

A CommunicationInterface is a special type of a DeviceItem acting as a connection point of a device to a network (e.g. network card). Therefore, a CommunicationInterface has a LogicalEndpoint. Depending on the CommunicationInterface type a CommunicationInterface can contain different CommunicationInterface type specific parameters. Therefore, the bus specific parameters are described

in a separate bus specification. The standard parameters of a CommunicationInterface object are listed below:

- Label (string):
Name printed on the item e.g. unique identifier.
- Type (string)
Type of the CommunicationInterface (e.g. ExtensionRack).

3.2.3.11 Node

A Node specifies all the interface related networking information of a network node. (e.g. logical address, subnet mask). A Node belongs to the CommunicationInterface (DeviceItem). The parameters of a node are bus specific characteristics. It is a topology object of a physical connection (cable, glass fibre) between two network stations. Depending on the node type a node can contain different node type specific parameters. Therefore, the bus specific parameters are described in a separate bus specification. The standard parameters of a node object are listed below:

- Type (string):
The Type of the Network (e.g. Ethernet, Mpi) as defined in the bus specification.
- NetworkAddress (string):
Network address of this device item. The format depends on the Node type (e.g. a TCP/IP address for an IP network).

3.2.3.12 CommunicationPort

A CommunicationPort is the physical connection to the network. It is a topology object of a physical connection (cable, glass fibre) between two network stations. The standard parameters of a CommunicationPort object are listed below:

- Label (string):
Name printed on the device item e.g. unique identifier.

Ports are aggregated on an Interface which implicitly defines the relationship between the logical (= Interface) and physical (= Port) network connectivity. This aggregation need not be explicitly modelled in AutomationML because it is available via the derivation of Interface and Port from DeviceItem that already defines a generic DeviceItem to DeviceItem aggregation.

3.2.3.13 IoSystem

An IoSystem object is responsible for representing a master – slave relationship typically found in fieldbus systems. Although this relationship depends on a subnet connection between the interfaces, the object model does not enforce this (incomplete engineering). The parent object of the IoSystem object is the interface that acts as the master. All interfaces that act as slaves for this master are linked to the IoSystem object. Please note that the master interface and the slave interfaces are different object instances although they share the same class in the object model. The standard parameters of an IoSystem object are listed below:

- Number (int):
Number of the IoSystem.

3.2.3.14 PowerPort

A PowerPort is the physical connection between modules for power transfer. It is a topology object of a physical connection (cable) between two power modules. The standard parameters of a PowerPort object are listed below:

- Label (string):
Name printed on the module e.g. unique identifier.

A PowerPort has exactly one PowerPortInterface to link PowerPorts between modules for power transfer. The PowerPortInterface is acting as a connection point of a power module.

3.2.3.15 SensorPort

A SensorPort is the physical connection between modules for transfer of sensor signals. It is a topology object of a physical connection (cable) between two sensor modules. The standard parameters of a SensorPort object are listed below:

- Label (string):
Name printed on the module e.g. unique identifier.

A SensorPort has exactly one SensorPortInterface to link SensorPorts between modules for sensor signal transfer. The SensorPortInterface is acting as a connection point of a sensor module.

3.2.3.16 ModuleAssignment

The interface class "Module Assignment" is used to define the assignment of a module to a CPU in a multi CPU system. By using an internal link between "Module Assignment" interfaces a module is assigned to a CPU. A module can be assigned to several CPU's and a CPU can control several modules. Each DeviceItem shall have maximum one "Module Assignment" interface, which can be used by several internal links. If only one CPU exists, the "Module Assignment" interface can be omitted and is assumed that all modules are controlled by this CPU.

3.2.3.17 Bus specific Sections

Bus specific definitions and extension are defined in separate documents. AR APC defines the following abstract RoleClasses: DeviceItemBusExtension, NodeBusExtension and CommunicationInterfaceBusExtension:

3.2.3.17.1 DeviceItemBusExtension

The abstract RoleClass "DeviceItemBusExtension" is used to define additionally bus specific attributes for a DeviceItem.

Note: DeviceItemBusExtension shall only be used for DeviceItem objects not for derived DeviceItem objects (e.g. like port).

3.2.3.17.2 NodeBusExtension

The abstract RoleClass "NodeBusExtension" is used to define additionally bus specific attributes for a Node.

Note: NodeBusExtension shall only be used for Node objects.

3.2.3.17.3 CommunicationInterfaceBusExtension

The abstract RoleClass "CommunicationInterfaceBusExtension" is used to define additionally bus specific attributes for a CommunicationInterface.

Note: CommunicationInterfaceBusExtension shall only be used for CommunicationInterface objects.

Note:

Derivations from these abstract RoleClasses are defined in separate RoleClassLibraries. The following naming recommendation is recommended:

AutomationProjectConfiguration[BusTypeName]RoleClassLib.

The naming conventions for derivations of the abstract base classes is DeviceItem[BusTypeName], Node[BusTypeName] and CommunicationInterface[BusTypeName].

The BusTypeName shall exactly match the string for the attribute “type”.

If a class needs additional attributes, additional RoleClasses are introduced for deployment as an additional SupportedRoleClass.

As an implementation recommendation additional RoleClasses should be handled tolerant by the tools. This means that an importing tool should respect additional attributes even if the corresponding RoleClass is not specified at the object in the instance hierarchy.

4 Guideline for the use of the ECAD model in practical applications

To use the previously described method for modelling ECAD in AutomationML four steps are required.

In a first step the ECAD RoleClassLib must be generated or imported i.e. the appropriate RoleClassLib has to be defined. This RoleClass Lib contains the derivation of the ECAD Roles from the generic communication model of AutomationML.

Next the Interfaces must be generated or imported, i.e. the appropriate InterfaceClassLib has to be defined. This InterfaceClassLib contains the derivation of the ECAD Roles from the generic communication model of AutomationML.

In step 3 the SystemUnitClasses for the engineering domain must be identified and modelled as templates for further use. Here the structure of the Devices, DeviceItems... can be modelled especially with respect to the relevant properties to be considered. Therefore, appropriate <InternalElement>'s and attributes are added.

Finally, the defined structure can be used to model a practical system in the InstanceHierarchy.

This procedure is depicted in the following figure:

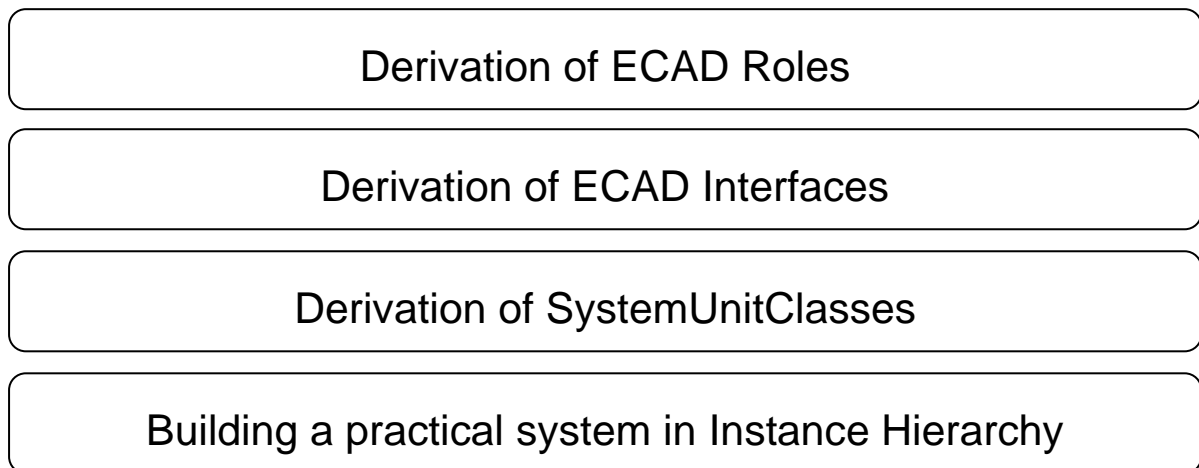


Figure 16 – Procedure for use of ECAD in AutomationML

5 Modelling of Automation Project Configuration data with AutomationML

5.1 RoleClassLibrary

Basement of the modelling are the required role classes. Facing the required model elements there are role classes especially required for Automation Project Configuration data modelling derived from role classes used for communication system modelling defined in AutomationML Whitepaper – Communication or derived from AutomationML basic roles defined in AutomationML Whitepaper – Architecture and general requirements.

The following figures represent the defined role class library.

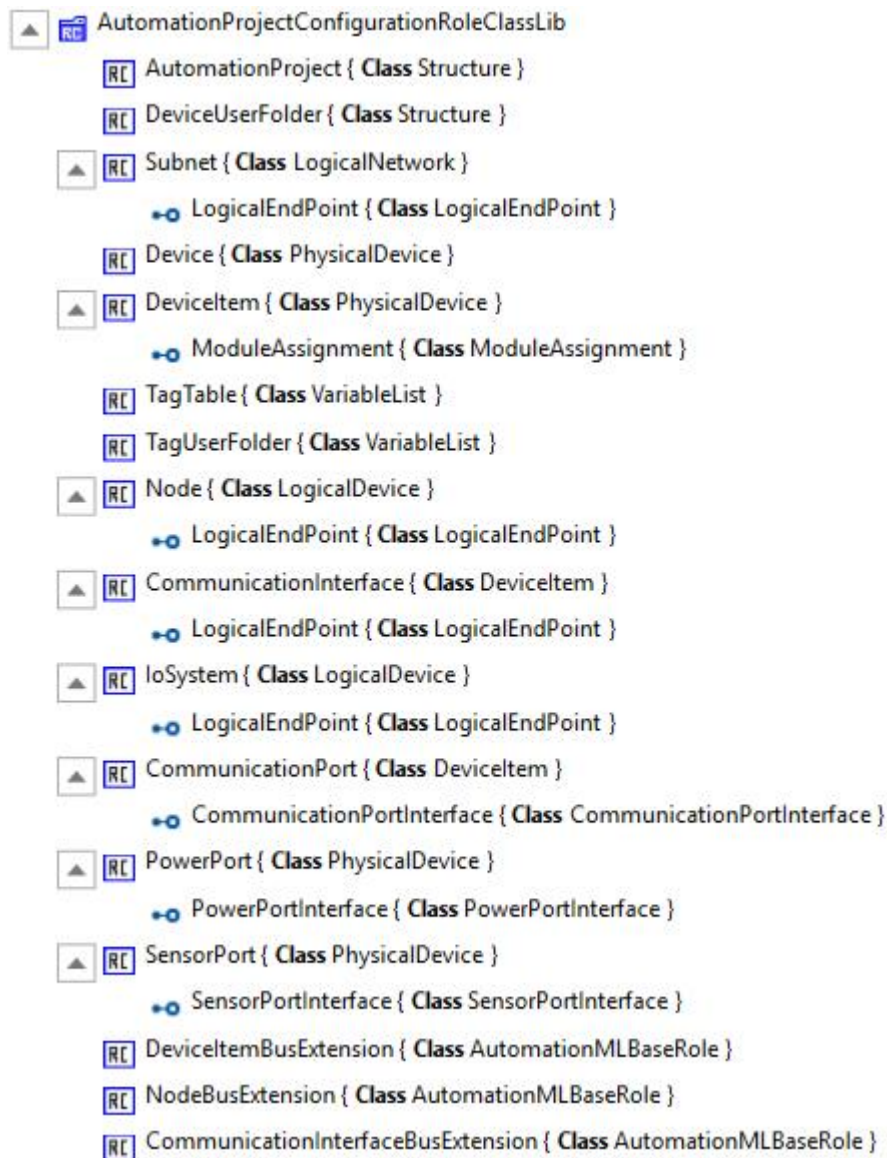


Figure 17 – AutomationProjectConfigurationRoleClassLib in AutomationML Editor view

```

<RoleClassLib
  Name="AutomationProjectConfigurationRoleClassLib">
  <Description>Automation Markup Language Automation Project Configuration Data Class Library</Description>
  <Version>1.2.0</Version>
  <RoleClass
    Name="AutomationProject"
    RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure">

```

```

<Attribute
  Name="ProjectManufacturer"
  AttributeDataType="xs:string"></Attribute>
<Attribute
  Name="ProjectSign"
  AttributeDataType="xs:string"></Attribute>
<Attribute
  Name="ProjectRevision"
  AttributeDataType="xs:string"></Attribute>
<Attribute
  Name="ProjectInformation"
  AttributeDataType="xs:string"></Attribute>
</RoleClass>
<RoleClass
  Name="DeviceUserFolder"
  RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure" />
<RoleClass
  Name="Subnet"
  RefBaseClassPath="CommunicationRoleClassLib/LogicalNetwork">
  <Attribute
    Name="Type"
    AttributeDataType="xs:string"></Attribute>
  <ExternalInterface
    Name="LogicalEndPoint"
    RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"
    ID="3e661cba-acfc-43b8-a02b-14ad7061f137" />
  </RoleClass>
<RoleClass
  Name="Device"
  RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice">
  <Attribute
    Name="TypeIdIdentifier"
    AttributeDataType="xs:string">
  <Attribute
    Name="TemplateIdentifier"
    AttributeDataType="xs:string" />
  </Attribute>
  <Attribute
    Name="Comment"
    AttributeDataType="xs:string" />
  <Attribute
    Name="Manufacturer"
    AttributeDataType="xs:string" />
  </RoleClass>
<RoleClass
  Name="DeviceItem"
  RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice">
  <Attribute
    Name="TypeName"
    AttributeDataType="xs:string"></Attribute>
  <Attribute
    Name="DeviceItem Type"
    AttributeDataType="xs:string">
  <Attribute
    Name="Customized"
    AttributeDataType="xs:boolean">
    <DefaultValue>false</DefaultValue>
  </Attribute>
  </Attribute>
  <Attribute
    Name="PositionNumber"
    AttributeDataType="xs:int"></Attribute>
  <Attribute
    Name="BuiltIn"
    AttributeDataType="xs:boolean">
    <DefaultValue>false</DefaultValue>
  </Attribute>
  <Attribute
    Name="TypeIdIdentifier"
    AttributeDataType="xs:string">
  <Attribute
    Name="TemplateIdentifier"

```

```

    AttributeDataType="xs:string" />
  </Attribute>
  <Attribute
    Name="Manufacturer"
    AttributeDataType="xs:string" />
  <Attribute
    Name="CustomAttributes">
    <RefSemantic
      CorrespondingAttributePath="ListType" />
    <Attribute
      Name="AttributeName1"
      AttributeDataType="xs:string"></Attribute>
    <Attribute
      Name="AttributeName2"
      AttributeDataType="xs:string"></Attribute>
    </Attribute>
  <Attribute
    Name="FirmwareVersion"
    AttributeDataType="xs:string" />
  <Attribute
    Name="PlantDesignation IEC"
    AttributeDataType="xs:string">
    <Description>Function oriented reference designation following IEC 81346</Description>
    <RefSemantic
      CorrespondingAttributePath="IEC 81346-1:2009-07#5.3 - Function-oriented structure" />
    </Attribute>
  <Attribute
    Name="LocationIdentifier IEC"
    AttributeDataType="xs:string">
    <Description>Location oriented reference designation following IEC 81346</Description>
    <RefSemantic
      CorrespondingAttributePath="IEC 81346-1:2009-07#5.5 - Location-oriented structure" />
    </Attribute>
  <Attribute
    Name="ProductDesignation IEC"
    AttributeDataType="xs:string">
    <Description>Product oriented reference designation following IEC 81346</Description>
    <RefSemantic
      CorrespondingAttributePath="IEC 81346-1:2009-07#5.4 - Product-oriented structure" />
    </Attribute>
  <Attribute
    Name="InstallationDate"
    AttributeDataType="xs:dateTime" />
  <Attribute
    Name="Comment"
    AttributeDataType="xs:string" />
  <Attribute
    Name="Address">
    <RefSemantic
      CorrespondingAttributePath="OrderedListType" />
    <Attribute
      Name="1">
      <Attribute
        Name="StartAddress"
        AttributeDataType="xs:int" />
      <Attribute
        Name="Length"
        AttributeDataType="xs:int" />
      <Attribute
        Name="IoType"
        AttributeDataType="xs:string" />
      <Attribute
        Name="BitOffset"
        AttributeDataType="xs:int" />
      </Attribute>
    <Attribute
      Name="2">
      <Attribute
        Name="StartAddress"
        AttributeDataType="xs:int" />
      <Attribute
        Name="Length"

```



```

    AttributeDataType="xs:int" />
  <Attribute
    Name="IoType"
    AttributeDataType="xs:string" />
  <Attribute
    Name="BitOffset"
    AttributeDataType="xs:int" />
</Attribute>
<Attribute
  Name="3">
  <Attribute
    Name="StartAddress"
    AttributeDataType="xs:int" />
  <Attribute
    Name="Length"
    AttributeDataType="xs:int" />
  <Attribute
    Name="IoType"
    AttributeDataType="xs:string" />
  <Attribute
    Name="BitOffset"
    AttributeDataType="xs:int" />
</Attribute>
</Attribute>
<ExternalInterface
  Name="ModuleAssignment"
  RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/ModuleAssignment"
  ID="110c6f0b-75b7-4c3c-9d05-1b28eeec5df" />
</RoleClass>
<RoleClass
  Name="TagTable"
  RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice/VariableList">
  <Attribute
    Name="AssignToDefault"
    AttributeDataType="xs:boolean" />
</RoleClass>
<RoleClass
  Name="TagUserFolder"
  RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice/VariableList" />
<RoleClass
  Name="Node"
  RefBaseClassPath="CommunicationRoleClassLib/LogicalDevice">
  <Attribute
    Name="Type"
    AttributeDataType="xs:string" />
  <Attribute
    Name="NetworkAddress"
    AttributeDataType="xs:string" />
  <ExternalInterface
    Name="LogicalEndPoint"
    RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"
    ID="9562e3ae-8c2b-4055-a327-3ab66f949d5e" />
</RoleClass>
<RoleClass
  Name="CommunicationInterface"
  RefBaseClassPath="AutomationProjectConfigurationRoleClassLib/DeviceItem">
  <Attribute
    Name="Label"
    AttributeDataType="xs:string" />
  <Attribute
    Name="Type"
    AttributeDataType="xs:string" />
  <ExternalInterface
    Name="LogicalEndPoint"
    RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"
    ID="dedad3eb-1a51-4d7e-accb-fdc8213c6c23" />
</RoleClass>
<RoleClass
  Name="IoSystem"
  RefBaseClassPath="CommunicationRoleClassLib/LogicalDevice">
  <Attribute
    Name="Number"

```

```

    AttributeDataType="xs:int" />
  <ExternalInterface
    Name="LogicalEndPoint"
    RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"
    ID="003f6b58-c95a-4346-8a0c-aaad895a6492" />
</RoleClass>
<RoleClass
  Name="CommunicationPort"
  RefBaseClassPath="AutomationProjectConfigurationRoleClassLib/DeviceItem">
  <Attribute
    Name="Label"
    AttributeDataType="xs:string" />
  <ExternalInterface
    Name="CommunicationPortInterface"
    RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/CommunicationPortInterface"
    ID="b0f1bb7c-1df9-494e-8352-0cae067e357d"></ExternalInterface>
</RoleClass>
<RoleClass
  Name="PowerPort"
  RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice">
  <Attribute
    Name="Label"
    AttributeDataType="xs:string" />
  <ExternalInterface
    Name="PowerPortInterface"
    ID="6c3e2230-64d8-42e6-9ddd-3dbdf3310064"
    RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/PowerPortInterface" />
</RoleClass>
<RoleClass
  Name="SensorPort"
  RefBaseClassPath="CommunicationRoleClassLib/PhysicalDevice">
  <Attribute
    Name="Label"
    AttributeDataType="xs:string" />
  <ExternalInterface
    Name="SensorPortInterface"
    ID="32d38c98-80cb-4850-b3a2-ad789e4ab96a"
    RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/SensorPortInterface" />
</RoleClass>
<RoleClass
  Name="DeviceItemBusExtension"
  RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
<RoleClass
  Name="NodeBusExtension"
  RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
<RoleClass
  Name="CommunicationInterfaceBusExtension"
  RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole" />
</RoleClassLib>

```

Figure 18 – AutomationProjectConfigurationRoleClassLib as XML representation

Note: The attributes of the roles can be “mandatory” or “optional”:

- *mandatory:* The exporting tool exports the attribute and the importing tool imports the attribute. The importing tool might correct the value.
- *optional:* The exporting tool may export this attribute. The importing tool imports this attribute if the exporting tool has exported this attribute and it exists on the importer side. The importing tool might correct the value.

5.1.1 AutomationProject

An “**AutomationProject**” is derived from a “Structure” according to AutomationML Whitepaper - Architecture and general requirements. It is defined as follows.

Table 2 – Definition AutomationProject

Role class name	AutomationProject	
Description	The role class "AutomationProject" shall be used in order to represent the project from which the export arises.	
Parent Class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/AutomationProject	
Attributes	"ProjectName" (AttributeDataType="xs:string")	The attribute "ProjectName" defines the name of the project. This attribute is mandatory. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	"ProjectManufacturer" (AttributeDataType="xs:string")	The attribute "ProjectManufacturer" defines the manufacturer of the project. This attribute is optional.
	"ProjectSign" (AttributeDataType="xs:string")	The attribute "ProjectSign" defines the unique identification of the project. This attribute is optional.
	"ProjectRevision" (AttributeDataType="xs:string")	The attribute "ProjectRevision" defines the revision number of the project. This attribute is optional.
	"ProjectInformation" (AttributeDataType="xs:string")	The attribute "ProjectInformation" defines commenting information of the project. This attribute is optional.

5.1.2 DeviceUserFolder

A “**DeviceUserFolder**” is derived from a “Structure” It is defined as follows.

Table 3 – Definition DeviceUserFolder

Role class name	DeviceUserFolder	
Description	The role class “DeviceUserFolder” shall be used in order to support the structure of a device within a project.	
Parent Class	AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/DeviceUserFolder	
Attributes	“Name” (AttributeDataType=”xs:string”)	The attribute “Name” defines the name of the DeviceUserFolder. This attribute is mandatory. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>

5.1.3 Subnet

A “**Subnet**” is derived from a “LogicalNetwork” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 4 – Definition Subnet

Role class name	Subnet	
Description	The role class “Subnet” shall be used in order to represent storing and managing of properties and functionality of networks.	
Parent Class	CommunicationRoleClassLib/LogicalNetwork	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Subnet	
Attributes	“Name” (AttributeDataType=”xs:string”)	The attribute “Name” defines the name of the subnet. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i> This attribute is mandatory.
	“Type” (AttributeDataType=”xs:string”)	The attribute “Type” defines the identifier of the type of the subnet. The value of the type is defined in the bus specific specification. This attribute is mandatory.
Interfaces	“LogicalEndPoint” (RefBaseClassPath=”CommunicationInterfaceClassLib/LogicalEndPoint”)	This interface is used to link node elements to a subnet. The direction of the link is irrelevant. A Subnet has exactly one LogicalEndpoint to connect the subnet to Nodes.

Depending on the different communication systems and bus systems a Subnet may contain additional attributes. These attributes and the bus specific parameters are described in separate bus specifications. Please refer to these bus specifications for more information.

5.1.4 Device

A “**Device**” is derived from a “PhysicalDevice” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 5 – Definition Device

Role class name	Device	
Description	The role class "Device" shall be used in order to represent a collection in which the individual HW objects of a slave or rack, including the slave or rack HW item, are brought together.	
Parent Class	CommunicationRoleClassLib/PhysicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Device	
Attributes	<p>"Name" (AttributeDataType="xs:string")</p> <p>The attribute "Name" defines the name of the device. This attribute is mandatory. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i></p>	
	<p>"TypeIdentifier" (AttributeDataType="xs:string")</p>	<p>"Template Identifier" (AttributeDataType="xs:string")</p> <p>The attribute "TypeIdentifier" defines the family identifier of the device type. This attribute is optional. The attribute "TemplateIdentifier" references the path of a template element, for instance a library. The syntax is defined by the corresponding target system element. This attribute is optional.</p>
	<p>"Manufacturer" (AttributeDataType="xs:string")</p> <p>The attribute "Manufacturer" defines the manufacturer of the device. The name of the manufacturer must be unambiguous and unique assigned to the manufacturer. This attribute is optional.</p>	
	<p>"Comment" (AttributeDataType="xs:string")</p> <p>The attribute "Comment" defines a comment for the device. The attribute "Comment" follows the Best Practice Recommendation Multilingual expressions in AutomationML. This attribute is optional.</p>	

Note: The attribute "TypeIdentifier" has a prefix which describes the semantic of the following identifier separated by ":" The following prefixes are allowed: "OrderNumber", "GSD", "System", "CSP+". In case of a missing TypeIdentifier the importing tool shall apply a substitution strategy. It is not ensured that the substitution strategy succeeds in all cases.

Examples:

TypeIdentifier = "GSD:SIEM8139.GSD/DAP"

TypeIdentifier = "System:Device.Generic"

TypeIdentifier = "System:Device.S7-1500"

TypeIdentifier = "System:Device.IQ-R"

TemplateIdentifier =

"GlobalLib://TemplateLibrary/Master copies/S7-1500/preconfigured/PLCs/s7-1518F"

5.1.5 Deviceltem

A “**Deviceltem**” is derived from a “PhysicalDevice” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 6 – Definition Deviceltem

Role class name	Deviceltem	
Description	The role class “Deviceltem” shall be used in order to represent a general object class for HW modules and submodules.	
Parent Class	CommunicationRoleClassLib/PhysicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Deviceltem	
Attributes	“Name” (AttributeDataType="xs:string")	The attribute “Name” defines the name of the device. This attribute is mandatory. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	“TypeName” (AttributeDataType="xs:string")	The Attribute “TypeName” defines additional type information. This attribute is optional.
	“DeviceltemType” (AttributeDataType="xs:string")	The attribute “DeviceltemType” defines the classification of the Deviceltem. This attribute is optional. The subattribute “Customized” indicates if the DeviceltemType contains vendor specific information (Customized = “true”) or not (Customized = “false”). If customized is omitted or set to “false” the Deviceltem contains already standardized information (Standard values are: CPU, HeadModule, Accessory). This attribute is optional.
	“Customized” (AttributeDataType="xs:boolean")	
	“PositionNumber” (AttributeDataType="xs:int")	The attribute “PositionNumber” defines the slot number where this Deviceltem is plugged in. This attribute is optional. It is mandatory in case of plugable Deviceltems and for Deviceltems which can contain plugable Deviceltems
	“BuiltIn” (AttributeDataType="xs:boolean")	The attribute “BuiltIn” defines that this module is a build-in part of another module. This attribute is optional. The default value is “false”.
	“TypeIdentifier” (AttributeDataType="xs:string")	The attribute “TypeIdentifier” defines the identifier of the Deviceltem type. This attribute is mandatory if “BuiltIn” is “false”. The attribute is optional, if “BuiltIn” is true. A wildcard shall be “*”. The attribute “TemplateIdentifier” references the path of a template element, for instance a library. The syntax is defined by the corresponding target system element. This attribute is optional.
	“Template Identifier” (AttributeDataType="xs:string")	
	“Manufacturer” (AttributeDataType="xs:string")	The attribute “Manufacturer” defines the manufacturer of the Deviceltem. The name of the manufacturer must be unambiguous and unique assigned to the manufacturer. This attribute is optional.

	<p>“CustomAttributes” (ListType)[0..n]</p>		<p>“AttributeName” (AttributeDataType=“xs:string”)</p>	<p>The name of the CustomAttribute defines the property name in the target system. The value defines the value of this CustomAttribute. The AttributeName is mandatory if the attribute “CustomAttributes” exists. The AttributeName shall be unique. Mapping of several custom attributes with the same name to different built-in submodules can only be made by make the key unique by appending a counter like #1, #2, #3. Only simple data types in a flat list are supported in this version.</p>
	<p>“FirmwareVersion” (AttributeDataType=“xs:string”)</p>		<p>The attribute “FirmwareVersion” defines the firmware version of e.g. a CPU to identify the module correctly. This attribute is optional.</p>	
	<p>“Comment” (AttributeDataType=“xs:string”)</p>		<p>The attribute “Comment” defines a comment for the device. The attribute “Comment” follows the Best Practice Recommendation Multilingual expressions in AutomationML. This attribute is optional.</p>	
	<p>“Address” (OrderedListType)[0..n]</p>	<p>“1”</p>	<p>“StartAddress” (AttributeDataType=“xs:int”)</p>	<p>The attribute “Address” is optional. The subattribute “StartAddress” defines the start of the address. The subattribute “StartAddress” is mandatory if the attribute “Address” exists.</p>
			<p>“Length” (AttributeDataType=“xs:int”)</p>	<p>The attribute “Length” defines the total width of all of the channels on the device item. This attribute is optional.</p>
			<p>“IoType” (AttributeDataType=“xs:string”)</p>	<p>The attribute “IoType” specifies the direction INPUT or OUTPUT. The subattribute “IoType” is mandatory if the attribute “Address” exists.</p>
			<p>“BitOffset” (AttributeDataType=“xsint”)</p>	<p>The subattribute “BitOffset” defines the offset within a StartAddress. The subattribute “BitOffset” is optional. The default value is “0”.</p>
<p>“PlantDesignation IEC” (AttributeDataType=“ xs:string” and RefSemantic=“ IEC 81346-1:2009-07#5.3 - Function-oriented structure IEC81346”)</p>		<p>The attribute “PlantDesignation IEC” defines the function designation for this device item according IEC81346. The attribute shall contain in the subattribute refSemantic the value “IEC 81346-1:2009-07#5.3 - Function-oriented structure” to enable its identification if the name is changed. The length of this attribute is bus specific.</p>		

		This attribute is optional.
	"LocationIdentifier IEC" (AttributeDataType=" xs:string" and RefSemantic=" IEC 81346-1:2009-07#5.5 - Location-oriented structure IEC81346")	The attribute "LocationIdentifier IEC" defines the location designation for this device item according IEC81346. The attribute shall contain in the subattribute refSemantic the value "IEC 81346-1:2009-07#5.5 - Location-oriented structure" to enable its identification if the name is changed. The length of this attribute is bus specific. This attribute is optional.
	"ProductDesignation IEC" (AttributeDataType=" xs:string" and RefSemantic=" IEC 81346-1:2009-07#5.4 - Product-oriented structure")	The attribute "ProductDesignation IEC" defines the product designation for this device item according IEC81346. The attribute shall contain in the subattribute refSemantic the value "IEC 81346-1:2009-07#5.4 - Product-oriented structure" to enable its identification if the name is changed. The length of this attribute is bus specific. This attribute is optional.
Interfaces	"InstallationDate" (AttributeDataType=" xs:dateTime	The attribute "InstallationDate" defines the date of the installation of the DeviceItem. This attribute is optional.
	"ModuleAssignment" (RefBaseClassPath=" AutomationProjectConfigurationInterfaceClassLib/ModuleAssignment")	This interface is used to link a module to a CPU in case of multiple CPU configurations. The direction of the link is irrelevant.

Note: The Address attribute is a finite list of variable length. The number of list elements is application case dependent.

Note: The attribute "TypeIdentifier" has a prefix which describes the semantic of the following identifier separated by ":" The following prefixes are allowed: "OrderNumber", "GSD", "System", "CSP+".

Examples:

TypeIdentifier = "OrderNumber:3RK1 200-0CE00-0AA2"

TypeIdentifier = "GSD:SIEM8139.GSD/DAP/DAP 1"

TypeIdentifier = "System:Rack.Generic"

TypeIdentifier = "CSP+:AJ65VBTCE2-8T"

TemplateIdentifier =

"GlobalLib://TemplateLibrary/Master copies/S7-1500/preconfigured/DIs/16x24VDC BA_Advanced"

Note:

If BuildIn=true then the identification shall result from the UUID of the first DeviceItem in the parent hierarchy with BuiltIn=False and the Position-Number of the DeviceItem.

Note:

ECAD and Automation Engineering Systems have different perspectives on the structure of devices and modules. An exchanged AML file must ensure that both perspectives can be derived from its content. Two scenarios have to be distinguished:

- A device or module is composed of two device items. The two device items appear in an automation engineering system at the same level but the second is modelled as part of the first one (main device item). In the ECAD system the composition of the two device items is treated as one module. (hierarchical scenario)

- A device or module is modelled by independent artifacts in the automation system while it is handled as a single physical entity in the ECAD system. This is often used to model various configurations of the same module / device. (flat scenario)

Both scenarios can be handled by using the product administration of the ECAD system. For this aim the ECAD system uses one of the two defined properties on device items, which are handled as built-in by the ECAD system:

- GSD-index
- sub-module order number

At least one property has to be defined. If both are given, sub-module order number takes preference over GSD-index. The actual names of these properties are ECAD system specific. They are reflected in the value of type identifier attribute of the corresponding built-in device item.

For the flat scenario for all device items the value of its type identifier is used. For the hierarchical scenario the main device item uses the value of its type identifier while the built-in device item uses the type identifier of the main device item appended with “#BUILTIN” as value for sub-module order number.

An ECAD export algorithm shall work as follows:

- If no property is defined for the built-in device item, it will be exported from the ECAD system as child of the main device item (hierarchical model)
- If at least one property is defined, the built-in device item will be exported at the level of the main device. Built-in device items shall always be positioned following the main device item. If the order number possesses “#BUILTIN” as part of its order number for the device item, the built-in flag will be set in the AML file.

In case of the flat scenario, PLC engineering systems must export the corresponding type identifier value for each device item. For the hierarchical scenario for the main device item its type identifier value is exported, while for the built-in device item the value of the main device item’s type identifier value appended with “#BUILTIN” is exported.

Remarks:

- Version information of built-in devices will be ignored during import. The importing system may issue a warning.

Depending on the different communication systems and bus systems some DeviceItems may contain additional, unused or restricted attributes. These attributes and the bus specific parameters are described in separate bus specifications. Please refer to these bus specifications for more information.

5.1.6 TagTable

A “**TagTable**” is derived from a “VariableList” according to AutomationML Whitepaper - Communication. Therefore, the VariableList must be mandatory according AutomationML Whitepaper Communication. It is defined as follows.

Table 7 – Definition TagTable

Role class name	TagTable	
Description	The role class "TagTable" shall be used in order to support the structure of tags	
Parent Class	CommunicationRoleClassLib/PhysicalDevice/VariableList	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/TagTable	
Attributes	"Name" (AttributeDataType="xs:string")	The attribute "Name" defines the name of the TagTable. This attribute is mandatory. This attribute is ignored if AssignToDefault is "true". <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>
	"AssignToDefault" (AttributeDataType="xs:boolean")	The Attribute "AssignToDefault" defines if the Tags inside will be imported to an existing default TagTable. This attribute is optional.

Note: While importing if the TagTable has an attribute with 'True' value, then all the Tags inside will be imported to an existing default TagTable. In this case the Name of the TagTable is ignored by the importing tool. By default, False value is assumed for 'AssignToDefault' attribute if it does not exist while importing.

5.1.7 TagUserFolder

A “**TagUserFolder**” is derived from a “VariableList” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 8 – Definition TagUserFolder

Role class name	TagUserFolder	
Description	The role class “TagUserFolder” shall be used in order to support the structure TagTables within a DeviceItem.	
Parent Class	CommunicationRoleClassLib/PhysicalDevice/VariableList	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/TagUserFolder	
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” defines the name of the TagUserFolder. This attribute is mandatory. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>

5.1.8 Node

A “**Node**” is derived from a “logicalDevice” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 9 – Definition Node

Role class name	Node	
Description	The role class “Node” shall be used in order to specify all the interface related networking information of a network node. <i>Note: The name of the Node is modelled by the standard attribute Name of the relevant CAEX object.</i>	
Parent Class	CommunicationRoleClassLib/logicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/Node	
Attributes	“Type” (AttributeDataType=“xs:string”)	The attribute “Type” defines the type of the network. The value of the type is defined in the bus specific specification. This attribute is mandatory.
	“NetworkAddress” (AttributeDataType=“xs:string”)	The attribute “NetworkAddress” defines the network address of this device item. This attribute is mandatory.
Interfaces	“LogicalEndPoint” (RefBaseClassePath=“CommunicationInterfaceClassLib/LogicalEndPoint”)	This interface is used to link the node to a subnet. The direction of the link is irrelevant. In case of connection to more than one subnet the one and only logical endpoint shall contain all connections.

Note: The identification shall result from the identification of the Communication Interface.

Depending on the different communication systems and bus systems a Node may contain additional, unused or restricted attributes. These attributes and the bus specific parameters are described in separate bus specifications. Please refer to these bus specifications for more information.

5.1.9 CommunicationInterface

A “**CommunicationInterface**” is derived from a “DeviceItem” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 10 – Definition CommunicationInterface

Role class name	CommunicationInterface	
Description	<p>The role class "CommunicationInterface" shall be used in order to define the connection point of a device to a network.</p> <p><i>Note: The name of the CommunicationInterface is modelled by the standard attribute Name of the relevant CAEX object.</i></p>	
Parent Class	AutomationProjectConfigurationRoleClassLib/DeviceItem	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/CommunicationInterface	
Attributes	"Label" (AttributeDataType="xs:string")	<p>The attribute "Label" defines the name printed on the item.</p> <p>This attribute is mandatory if "BuiltIn" is "true".</p>
	"Type" (AttributeDataType="xs:string")	<p>The attribute "Type" defines the type of the CommunicationInterface. The value of the type is defined in the bus specific specification.</p> <p>The attribute is optional.</p>
Interfaces	"LogicalEndPoint" (RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint")	<p>This interface is used to link the CommunicationInterface to an IOSystem. The direction of the link is irrelevant. In case of connection to more than one IOSystem the one and only logical endpoint shall contain all connections.</p>

Note: The attribute "Type" is new from AR APC 1.1.0. If the attribute "Type" is missing, the interface is handled as an external bus due to compatibility to AR APC V1.0.0.

Note: The identification shall result from the UUID of the first DeviceItem in the parent hierarchy with BuiltIn=False starting with the interface or port itself, the Bus-System, if present the Label of the Communication-Interface and if present the Label of the Communication-Port. Because if one of these identifying attributes changes the bus-connector itself has changed.

Depending on the different communication systems and bus systems a CommunicationInterface may contain additional, unused or restricted attributes. These attributes and the bus specific parameters are described in separate bus specifications. Please refer to these bus specifications for more information.

5.1.10 IoSystem

An "IoSystem" is derived from LogicalDevice. It is defined as follows.

Table 11 – Definition IoSystem

Role class name	IoSystem	
Description	The role class "IoSystem" shall be used in order to model the master – slave relationship typically found in fieldbus systems. <i>Note: The name of the IoSystem is modelled by the standard attribute Name of the relevant CAEX object.</i>	
Parent Class	CommunicationRoleClassLib/LogicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/IOSystem	
Attributes	"Number" (AttributeDataType="xs:int")	The attribute "Number" defines the unique number of the IoSystem. This attribute is optional.
Interfaces	"LogicalEndPoint" (RefBaseClassePath="CommunicationInterfaceClassLib/LogicalEndPoint")	This interface is used to link the IoSystem to a CommunicationInterface. The direction of the link is irrelevant. In case of connection to more than one CommunicationInterface the one and only logical endpoint shall contain all connections.

Note: The identification shall result from the identification of the Communication Interface.

5.1.11 CommunicationPort

A "CommunicationPort" is derived from a DeviceItem. It is defined as follows.

Table 12 – Definition CommunicationPort

Role class name	CommunicationPort	
Description	The role class "CommunicationPort" shall be used in order to model the device item applied to physically establish the connection to the network. <i>Note: The name of the CommunicationPort is modelled by the standard attribute Name of the relevant CAEX object.</i>	
Parent Class	AutomationProjectConfigurationRoleClassLib/DeviceItem	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/CommunicationPort	
Attributes	"Label" (AttributeDataType="xs:string")	The attribute "Label" defines the name printed on the Port. This attribute is mandatory if "BuiltIn" is "true". The attribute PositionNumber is mandatory if "BuiltIn" is "False"
Interfaces	"CommunicationPortInterface" (RefBaseClassePath="AutomationProjectConfigurationInterfaceClassLib/CommunicationPortInterface")	This interface is used to link the Port to another port. The direction of the link is irrelevant. In case of connection to more than one Port the one and only interface shall contain several logical endpoints for the different connections.

Note: The identification shall result from the identification of the Communication Interface and the label of the port.

Depending on the different communication systems and bus systems a CommunicationPort may contain additional, unused or restricted attributes. These attributes and the bus specific parameters are described in separate bus specifications. Please refer to these bus specifications for more information.

5.1.12 PowerPort

A "PowerPort" is derived from a PhysicalDevice. It is defined as follows.

Table 13 – Definition PowerPort

Role class name	PowerPort	
Description	The role class "PowerPort" shall be used in order to model the physical connection between modules for power transfer. <i>Note: The name of the PowerPort is modelled by the standard attribute Name of the relevant CAEX object.</i>	
Parent Class	AutomationProjectConfigurationRoleClassLib/PhysicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/PowerPort	
Attributes	"Label" (AttributeDataType="xs:string")	The attribute "Label" defines the name printed on the Port. This attribute is mandatory.
Interfaces	"PowerPortInterface" (RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/PowerPortInterface")	This interface is used to link the Port to another port. Only one PowerPortInterface shall be allowed. The direction of the link is not relevant.

5.1.13 SensorPort

A "SensorPort" is derived from a PhysicalDevice. It is defined as follows.

Table 14 – Definition SensorPort

Role class name	SensorPort	
Description	The role class "SensoPort" shall be used in order to model the physical connection between sensor modules for transfer of sensor signals. <i>Note: The name of the SensorPort is modelled by the standard attribute Name of the relevant CAEX object.</i>	
Parent Class	AutomationProjectConfigurationRoleClassLib/PhysicalDevice	
Path for Element reference	AutomationProjectConfigurationRoleClassLib/SensorPort	
Attributes	"Label" (AttributeDataType="xs:string")	The attribute "Label" defines the name printed on the Port. This attribute is mandatory.
Interfaces	"SensorPortInterface" (RefBaseClassPath="AutomationProjectConfigurationInterfaceClassLib/SensorPortInterface")	This interface is used to link the Port to another port. Only one SensorPortInterface shall be allowed. The direction of the link is not relevant.

5.1.14 DeviceItemBusExtension

A "DeviceItemBusExtension" is derived from a "AutomationMLBaseRole". It is defined as follows.

Table 15 – Definition DeviceItemBusExtension

Role class name	DeviceItemBusExtension
Description	The RoleClass "DeviceItemBusExtension" is used to define additionally bus specific attributes for a DeviceItem. <i>Note: DeviceItemBusExtension shall only be used for DeviceItem objects not for derived DeviceItem objects.</i>
Parent Class	AutomationMLBaseRole
Path for Element reference	AutomationProjectConfigurationRoleClassLib/DeviceItemBusExtension

The abstract RoleClass "DeviceItemBusExtension" is used to define additionally bus specific attributes for a DeviceItem. These bus specific attributes shall be defined in a derived class in a separate RoleClassLibrary for deployment as an additional Supported Role Class.

Note: DeviceItemBusExtension shall only be used for DeviceItem objects.

5.1.15 NodeBusExtension

A "NodeBusExtension" is derived from a "AutomationMLBaseRole". It is defined as follows.

Table 16 – Definition NodeBusExtension

Role class name	NodeBusExtension
Description	The RoleClass "NodeBusExtension" is used to define additionally bus specific attributes for a Node. <i>Note: NodeBusExtension shall only be used for Node objects.</i>
Parent Class	AutomationMLBaseRole
Path for Element reference	AutomationProjectConfigurationRoleClassLib/NodeBusExtension

The abstract RoleClass "NodeBusExtension" is used to define additionally bus specific attributes for a Node. These bus specific attributes shall be defined in a derived class in a separate RoleClassLibrary for deployment as an additional Supported Role Class.

Note: NodeBusExtension shall only be used for Node objects.

5.1.16 CommunicationInterfaceBusExtension

A "CommunicationInterfaceBusExtension" is derived from a "AutomationMLBaseRole". It is defined as follows.

Table 17 – Definition CommunicationInterfaceBusExtension

Role class name	CommunicationInterfaceBusExtension
Description	The RoleClass "CommunicationInterfaceBusExtension" is used to define additionally bus specific attributes for a CommunicationInterface. <i>Note: CommunicationInterfaceBusExtension shall only be used for CommunicationInterface objects.</i>
Parent Class	AutomationMLBaseRole
Path for Element reference	AutomationProjectConfigurationRoleClassLib/CommunicationInterfaceBusExtension

The abstract RoleClass "CommunicationInterfaceBusExtension" is used to define additionally bus specific attributes for a CommunicationInterface. These bus specific attributes shall be defined in a derived class in a separate RoleClassLibrary for deployment as an additional Supported Role Class.

Note: CommunicationInterfaceBusExtension shall only be used for CommunicationInterface objects.

5.2 InterfaceClassLibrary

Second main basement of the modelling are the required interface classes. Facing the required model elements there are interface classes especially required for Automation Project Configuration data modelling derived from interface classes used from communication system modelling defined in AutomationML Whitepaper – Communication or derived from AutomationML basic interface classes defined in AutomationML Whitepaper – Architecture and general requirements

The following figures represent the interface class library.

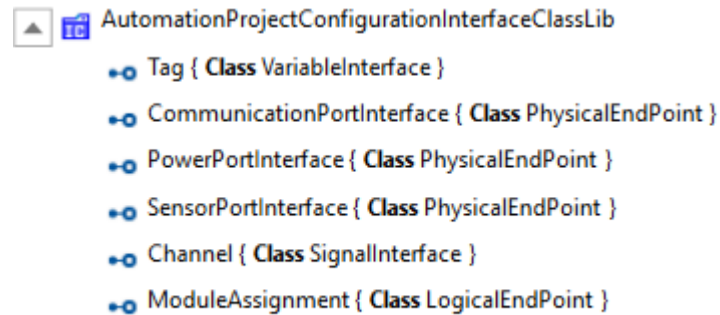


Figure 19 – AutomationProjectConfigurationInterfaceClassLib in AutomationML Editor view

```
<InterfaceClassLib
  Name="AutomationProjectConfigurationInterfaceClassLib">
  <Description>Automation Markup Language Automation Project Configuration InterfaceClass Library</Description>
  <Version>1.2.0</Version>
  <InterfaceClass
    Name="Tag"
    RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCOpenXMLInter
face/VariableInterface">
    <Attribute
      Name="DataType"
      AttributeDataType="xs:string">
    <Attribute
      Name="Customized"
      AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
  </Attribute>
  <Attribute
    Name="IoType"
    AttributeDataType="xs:string"></Attribute>
  <Attribute
    Name="LogicalAddress"
    AttributeDataType="xs:string"/>
  <Attribute
    Name="Comment"
    AttributeDataType="xs:string"/>
  </InterfaceClass>
  <InterfaceClass
    Name="CommunicationPortInterface"
    RefBaseClassPath="CommunicationInterfaceClassLib/PhysicalEndPoint"/>
  <InterfaceClass
    Name="PowerPortInterface"
    RefBaseClassPath="CommunicationInterfaceClassLib/PhysicalEndPoint"/>
  <InterfaceClass
    Name="SensorPortInterface"
    RefBaseClassPath="CommunicationInterfaceClassLib/PhysicalEndPoint"/>
  <InterfaceClass
    Name="Channel"
    RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/Communication/SignalInterface">
    <Attribute
      Name="Type"
      AttributeDataType="xs:string"/>
  </InterfaceClass>
</InterfaceClassLib>
```



```
<Attribute
  Name="IoType"
  AttributeDataType="xs:string"/>
<Attribute
  Name="Number"
  AttributeDataType="xs:int"/>
<Attribute
  Name="Length"
  AttributeDataType="xs:int"/>
</InterfaceClass>
<InterfaceClass
  Name="ModuleAssignment"
  RefBaseClassPath="CommunicationInterfaceClassLib/LogicalEndPoint"/>
</InterfaceClassLib>
```

Figure 20 – AutomationProjectConfigurationInterfaceClassLib as XML representation

5.2.1 Tag

A “**Tag**” is derived from a “VariableInterface” according to AutomationML Whitepaper - Logic. It is defined as follows.

Table 18 – Definition Tag

Role class name	Tag		
Description	The Interface class “Tag” shall be used in order to represent the symbolic name of an I/O date. Tags shall only be used within a TagTable.		
Parent Class	VariableInterface		
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/Tag		
Attributes	“Name” (AttributeDataType=“xs:string”)	The attribute “Name” defines the name of the tag. This attribute is mandatory. <i>Note: This attribute is modelled by the standard attribute Name of the relevant CAEX object.</i>	
	“DataType” (AttributeDataType=“xs:string”)	“Customized” (AttributeDataType=“xs:boolean”)	The attribute “DataType” defines the type of the data. This attribute is mandatory. The subattribute “Customized” indicates if the DataType contains vendor specific data types (Customized = “true”) or not (Customized = “false”). If customized is omitted or set to “false” the DataType contains already defined information according IEC 61131 (e.g. BOOL, BYTE, WORD).
	“IoType” (AttributeDataType=“xs:string”)	The attribute “IoType” specifies the direction of the tag. This attribute has the value “Input” or “Output”. This attribute is optional.	
	“LogicalAddress” (AttributeDataType=“xs:string”)	The attribute “Logical Address” specifies the address of the tag. This attribute shall not contain the direction (see Note below) This attribute is optional.	
	“Comment” (AttributeDataType=“xs:string”)	The attribute “Comment” defines an comment for the device. The attribute “Comment” follows the Best Practice Recommendation Multilingual expressions in AutomationML. This attribute is optional.	

Tags without assigned channels and channels without assigned tags are possible (incomplete engineering). In case of Tags without assigned channels it is recommended to use Tag-name for identification if UUID does not match.

Note: The attribute “IoType” is new from AR APC 1.1.0. Due to upward compatibility with AR APC V 1.0.0 the attribute “LogicalAddress” may contain the direction (e.g. “I” or “O”) and the attribute “IoType” may be missed. In this case the following rule from AR APC 1.0.0 shall be applied furthermore:

The exporting ECAD tool defines the language mnemonic of the attribute. The importing PLC tool may change this mnemonic.

The exporting PLC tool may follow the international mnemonic. The importing ECAD tool doesn’t change this mnemonic.

Therefore, in case of round trip engineering the use of the international or independent mnemonic in all participating tools is recommended.

Use cases see appendix A

5.2.2 Channel

A “**Channel**” is derived from a “SignalInterface” according to AutomationML Whitepaper - Architecture and general requirements. It is defined as follows.

Table 19 – Definition Channel

Role class name	Channel	
Description	The role class “Channel” shall be used in order to define the process interface. A channel shall only be used within a DeviceItem. <i>Note: The name of the Channel is modelled by the standard attribute Name of the relevant CAEX object.</i>	
Parent Class	SignalInterface	
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/Channel	
Attributes	“Type” (AttributeDataType=“xs:string”)	The attribute “Type” defines the analog or digital type of the channel (e.g. “Digital”, “Analog”). This attribute is mandatory.
	“IoType” (AttributeDataType=“xs:string”)	The attribute “IoType” specifies the direction (e.g. “Input”, “Output”). This attribute is mandatory.
	“Number” (AttributeDataType=“xs:int”)	The attribute “Number” specifies the number of the channel starting with 0. This attribute is mandatory.
	“Length” (AttributeDataType=“xs:int”)	The attribute “Length” defines the total width of the channel. This attribute is optional.

A channel references with an Internal Link the associated Tags which are stored at a CPU DeviceItem. The direction of the link is not relevant.

Note: The identification shall result from the UUID of the first DeviceItem in the parent hierarchy with BuiltIn=False, the Number of the Channel, the Type of the Channel and the IoType of the Channel. Because if one of these identifying attributes changes the channel itself has changed.

Use cases see appendix A

5.2.3 CommunicationPortInterface

A “**CommunicationPortInterface**” is derived from a “PhysicalEndPoint” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 20 – Definition CommunicationPortInterface

Role class name	CommunicationPortInterface
Description	The interface class “CommunicationPortInterface” shall be used in order to define the physical connection to the network.
Parent Class	PhysicalEndPoint
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/CommunicationPortInterface

5.2.4 PowerPortInterface

A “**PowerPortInterface**” is derived from a “PhysicalEndPoint” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 21 – Definition *PowerPortInterface*

Role class name	PowerPortInterface
Description	The interface class “PowerPortInterface” shall be used in order to define the physical connection between power modules.
Parent Class	PhysicalEndPoint
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/PowerPortInterface

5.2.5 SensorPortInterface

A “**SensorPortInterface**” is derived from a “PhysicalEndPoint” according to AutomationML Whitepaper - Communication. It is defined as follows.

Table 22 – Definition *SensorPortInterface*

Role class name	SensorPortInterface
Description	The interface class “SensorPortInterface” shall be used in order to define the physical connection between sensor modules.
Parent Class	PhysicalEndPoint
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/SensorrPortInterface

5.2.6 ModuleAssignment

A “**ModuleAssignment**” is derived from a “LogicalEndpoint” according to AutomationML Whitepaper - Architecture and general requirements. It is defined as follows.

Table 23 – Definition *ModuleAssignment*

Role class name	ModuleAssignment
Description	The interface class “ModuleAssignment” shall be used in order to define the assignment of modules to CPU's.
Parent Class	LogicalEndpoint
Path for Element reference	AutomationProjectConfigurationInterfaceClassLib/ModuleAssignment

The interface class "Module Assignment" is used to define the assignment of a module to a CPU in a multi CPU system. By using an internal link between "Module Assignment" interfaces a module is assigned to a CPU. A module can be assigned to several CPU's and a CPU can control several modules. Each DeviceItem shall have maximum one "Module Assignment" interface, which can be used by several internal links. If only one CPU exists, the "Module Assignment" interface can be omitted and is assumed that all modules are controlled by this CPU. The direction of the internal links is not relevant.

5.2.7 Naming and Escaping

For all CAEX-Path Expressions the CAEX naming and escaping rules are defined as follows:

- In a name within a path contains the characters “[” and “]” these characters have to be escaped by replacing them with “[\” and “\]”
(e.g.: „R1/R1.[1]/R1.1.1“ => [R1]/[R1.\[1\]]/[R1.1.1]).

- If one of the characters “@”, “.”, “:” or “/” appears in the value of a path each part of the path must be enclosed by square brackets (“[” and “]”). Example: RefPartnerSideA="[4EA36EB0-8159-4829-8F4B-A829F3320E27]:[My.Tag]"
- Values of name attributes are not affected.
- Semantics of the operators “@” and “.” is not applied and must not be supported.

Appendix A Roundtrip Engineering and Identification of Logical AR APC Objects

This appendix describes various use cases regarding the roundtrip engineering between ECAD and PLC-Tools. Those use cases show the differences regarding the handling of the UUID of tags. In each use case the first “Flow of Events” highlights the case that the PLC tool doesn’t keep the UUID persistent which results in a finished life time of the Tag in each importing/exporting step.

The second “Flow of Events” highlights the case that the UUID of the Tag is kept persistent over the whole bidirectional data exchange process which results that the Tag stays alive in each importing/exporting step.

Use Case	New tag without channel assignment and without further changes
UseCaseID	1
Extends	-
Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name (“rpm”) and a new UUID (“4711”) without channel assignment.
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer does not change the name. 2. The PLC tool exports the tag with the same name (“rpm”) without channel assignment. 3. The ECAD tool imports and identifies the tag by its name.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer does not change the name. 2. The PLC tool exports the tag with the same name (“rpm”) without channel assignment. 3. The ECAD tool imports and identifies the tag by the UUID of the tag. If the UUID cannot be found in the ECAD project, the ECAD tool can identify the tag by its name and imprints the UUID (“4711”) for this tag or can create a new tag with the given UUID.

Use Case	New tag without channel assignment and change of tag
UseCaseID	2
Extends	-
Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name (“rpm”) and a new UUID (“4711”) without channel assignment.
Flow of Events in case of	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the name of the tag (“rpm” -> “voltage”).

finished life time of UUID	<ol style="list-style-type: none"> The PLC tool exports the tag with the new name ("voltage") without channel assignment. The ECAD tool imports the tag with the new name. This results in two tags. The ECAD importer has to decide how to handle the duplication.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> The PLC tool imports the tag. The PLC engineer changes the name of the tag ("rpm" -> "voltage"). The PLC tool must decide (e.g. by user interaction, automatically, etc.), if it's a new tag or an edited tag. If it is a new tag, the old tag would be deleted and a new one with a new UUID ("4712") is created. If the tag is edited, the UUID ("4711") is kept. The PLC tool exports the tag without channel assignment with the new name ("voltage") and the original UUID ("4711") or the new UUID ("4712") depending on PLC tool decision (e.g. the previous decision, by user interaction, etc...). The ECAD tool imports and identifies the tag by the UUID of the tag. If the UUID cannot be found in the ECAD project, the ECAD tool can identify the tag by its name and imprints the UUID ("4711" or "4712") for this tag or can create a new tag with the given UUID.

Use Case	New tag with channel assignment and without further changes
UseCaseID	3
Extends	-
Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name ("rpm") and a new UUID ("4711") with channel assignment ("Channel 5")
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> The PLC tool imports the tag. The PLC engineer does not change anything. The PLC tool exports the tag with the same name ("rpm") and with the same channel assignment ("Channel 5"). The ECAD tool imports and identifies the tag by its channel assignment.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> The PLC tool imports the tag. The PLC engineer does not change anything. The PLC tool exports the tag with the same name ("rpm") and with the same channel assignment ("Channel 5"). The ECAD tool imports and identifies the tag by the UUID and the channel assignment of the tag.

Use Case	New tag with channel assignment and change of channel assignment
UseCaseID	4
Extends	Roundtrip following after UseCaseID 3
Includes	-

Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer changes the channel assignment ("Channel 5"->"Channel 7").
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> 1. The ECAD engineer imports the result of "3". 2. The ECAD engineer changes the channel assignment (from "Channel 5" to "Channel 7") of a tag ("rpm"). 3. The PLC tool imports the tag ("rpm") and changes the channel assignment to the new channel ("Channel 7") 4. The PLC engineer doesn't change anything. 5. The PLC tool exports the tag ("rpm") with the channel assignment ("Channel 7"). 6. The ECAD tool imports and identifies the tag by the channel assignment.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> 1. The ECAD engineer imports the result of "3" 2. The ECAD engineer changes the channel assignment (from "Channel 5" to "Channel 7") of a tag ("rpm"). 3. The PLC tool imports the tag ("rpm") and changes the channel assignment to the new channel ("Channel 7") 4. The PLC engineer doesn't change anything. 5. The PLC tool exports the tag ("rpm") with the channel assignment ("Channel 7") and maintains the UUID. 6. The ECAD tool imports and identifies the tag by the UUID and the channel assignment.

Use Case	New tag with channel assignment and change of tag
UseCaseID	5
Extends	-
Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name ("rpm") and a new UUID ("4711") with channel assignment ("Channel 5").
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the name of the tag ("rpm" -> "voltage") and maintains the channel assignment ("Channel 5"). 2. The PLC tool exports the tag with the new name ("voltage") with channel assignment ("Channel 5"). 3. The ECAD tool imports and identifies the tag by the channel assignment.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the name of the tag ("rpm" -> "voltage"). 2. The PLC tool must decide (e.g. by user interaction, automatically, etc.), if it's a new tag or an edited tag. If it is a new tag, the old tag would be deleted and a new one with a new UUID ("4712") is created. If the tag is edited, the UUID ("4711") is kept. 3. The PLC tool exports the tag without channel assignment with the new name ("voltage") and the original UUID ("4711") or the new UUID

	<p>("4712"), depending on PLC tool decision (e.g. the previous decision, by user interaction, etc...).</p> <p>4. The ECAD tool imports and identifies the tag by the UUID of the tag. If the UUID cannot be found in the ECAD project, the ECAD tool can identify the tag by its name and channel assignment and imprints the UUID ("4711" or "4712") for this tag or can create a new tag with the given UUID.</p>
--	---

Use Case	New tag with channel assignment and change of channel assignment
UseCaseID	6
Extends	-
Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name ("rpm") and a new UUID ("4711") with channel assignment ("Channel 5").
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the channel assignment of the tag ("Channel 5"->"Channel 7") by changing the logical address (e.g. "I1.1"->"I1.2", "X4"->"X6"). 2. The PLC tool exports the tag with the same name ("rpm") and with the new channel assignment ("Channel 7"). 3. The ECAD tool imports and identifies the tag by the name and changes the channel assignment.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the channel assignment of the tag ("Channel 5"->"Channel 7") by changing the logical address (e.g. "I1.1"->"I1.2", "X4"->"X6"). 2. The PLC tool must decide (e.g. by user interaction, automatically, etc.), if it's a new tag or an edited tag. If it is a new tag, the old tag would be deleted and a new one with a new UUID ("4712") is created. If the tag is edited, the UUID ("4711") is kept. 3. The PLC tool exports the tag with the same name ("rpm"), the new channel assignment ("Channel 7") and the original UUID ("4711") or the new UUID ("4712"), depending on PLC tool decision (e.g. the previous decision, by user interaction, etc...). 4. The ECAD tool imports and identifies the tag by the UUID of the tag. If the UUID cannot be found in the ECAD project, the ECAD tool can identify the tag by its name and channel assignment and imprints the UUID ("4711" or "4712") for this tag or can create a new tag with the given UUID.

Use Case	New tag with channel assignment and change of tag and channel assignment
UseCaseID	7
Extends	-

Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name ("rpm") and a new UUID ("4711") with channel assignment ("Channel 5").
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the name of the tag ("rpm"->"voltage") and the channel assignment of the tag ("Channel 5"->"Channel 7"). 2. The PLC tool exports the tag with the new name ("voltage") and with the new channel assignment ("Channel 7"). 3. The ECAD tool imports and identifies the tag by the name and changes the channel assignment. The ECAD importer has to decide how to handle the duplication.
Flow of Events in case of persistent life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the name of the tag ("rpm"->"voltage") and the channel assignment of the tag ("Channel 5"->"Channel 7"). 2. The PLC tool must decide (e.g. by user interaction, automatically, etc.), if it's a new tag or an edited tag. If it is a new tag, the old tag would be deleted and a new one with a new UUID ("4712") is created. If the tag is edited, the UUID ("4711") is kept. 3. The PLC tool exports the tag with the new name ("voltage"), the new channel assignment ("Channel 7") and the original UUID ("4711") or the new UUID ("4712") depending on PLC tool decision (e.g. the previous decision, by user interaction, etc...). 4. The ECAD tool imports and identifies the tag by the UUID of the tag. If the UUID cannot be found in the ECAD project, the ECAD tool can identify the tag by its name and channel assignment and imprints the UUID ("4711" or "4712") for this tag or can create a new tag with the given UUID.

Use Case	New tag with channel assignment and change of start address
UseCaseID	8
Extends	-
Includes	-
Actor	ECAD Engineer, PLC Engineer
Brief Description	The ECAD engineer creates a new tag with a new name ("rpm") and a new UUID ("4711") with channel assignment ("Channel 5").
Flow of Events in case of finished life time of UUID	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the start address of the tag. 2. The PLC tool exports the tag with the new start address. 3. The ECAD tool imports and identifies the tag by its name and changes the start address.
Flow of Events in case of	<ol style="list-style-type: none"> 1. The PLC tool imports the tag. The PLC engineer changes the start address of the tag. 2. The PLC tool must decide (e.g. by user interaction, automatically, etc.), if it's a new tag or an edited tag. If it is a new tag, the old tag would be

persistent life time of UUID	<p>deleted and a new one with a new UUID ("4712") is created. If the tag is edited, the UUID ("4711") is kept.</p> <ol style="list-style-type: none">3. The PLC tool exports the tag with the new start address and the original UUID ("4711") or the new UUID ("4712") depending on PLC tool decision (e.g. the previous decision, by user interaction, etc...).4. The ECAD tool imports and identifies the tag by the UUID of the tag. If the UUID cannot be found in the ECAD project, the ECAD tool can identify the tag by its name and channel assignment and imprints the UUID ("4711" or "4712") for this tag or can create a new tag with the given UUID.
------------------------------	---