IEC 62769-8

Edition 1.0 2023-04

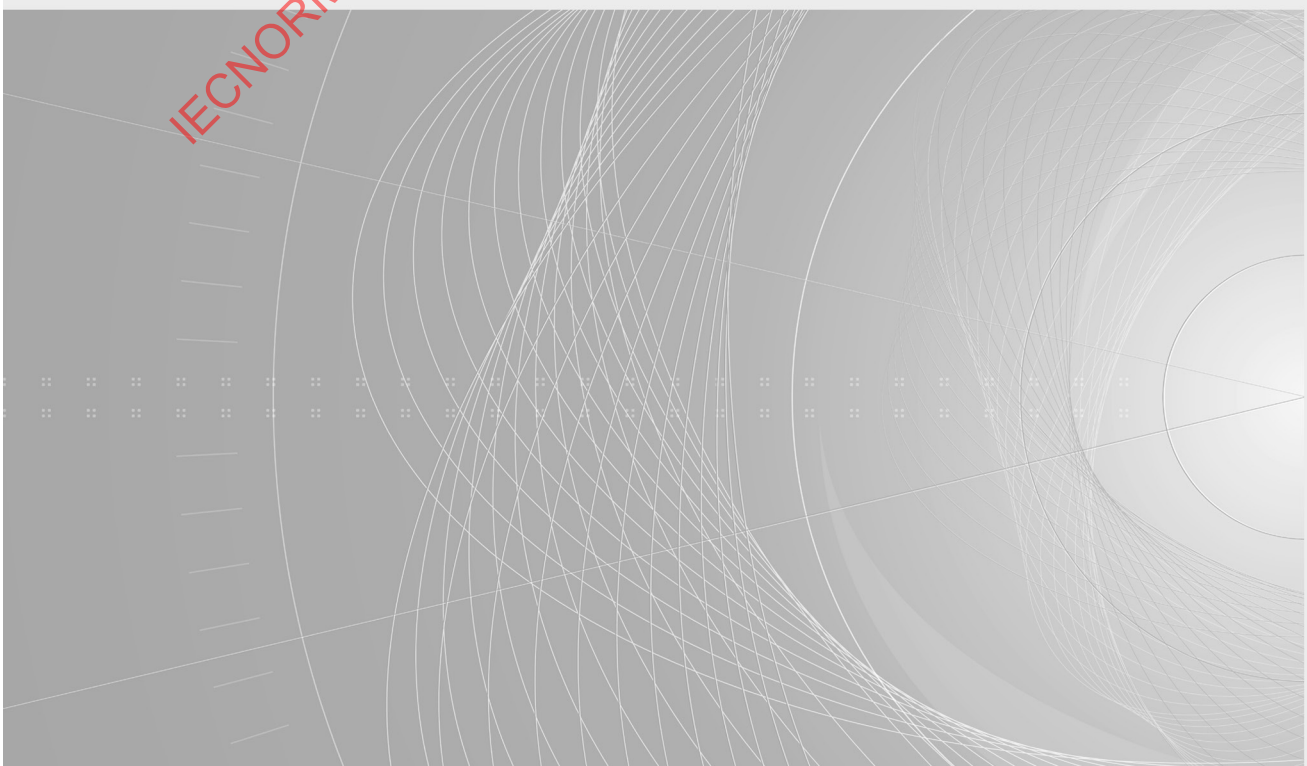
INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Field device integration (FDI®) – Part 8: EDD to OPC-UA Mapping

Intégration des appareils de terrain (FDI®) – Partie 8: Mapping de l'EDD avec l'OPC-UA



IEC 62769-8:2023-04(en-fr)

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

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

IEC 62769-8

Edition 1.0 2023-04

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Field device integration (FDI®) – Part 8: EDD to OPC-UA Mapping

Intégration des appareils de terrain (FDI®) – Partie 8: Mapping de l'EDD avec l'OPC-UA

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 33.040.40

ISBN 978-2-8322-6792-9

CONTENTS

FOREWORD 5
1 Scope 7
2 Normative references 7
3 Terms, definitions, abbreviated terms, acronyms and conventions 8
3.1 Terms and definitions 8
3.2 Abbreviated terms and acronyms 8
3.3 Conventions 8
3.3.1 Capitalization 8
3.3.2 Graphical notation 8
4 Overview 10
5 Basic principles of explicit mapping 11
5.1 Semantic maps to tag EDD constructs 11
5.2 Alias collections 12
5.2.1 General 12
5.2.2 Syntax of semantic id for alias mappings 12
5.3 Namespace Alias Collection 12
5.4 Reference Type Alias Collection 14
5.5 Semantic maps for OPC-UA type mapping 16
5.5.1 General 16
5.5.2 Syntax of semantic Id for OPC-UA 16
5.6 Semantic maps for unit mapping 17
5.6.1 General 17
5.6.2 Syntax of semantic Id for Units 17
5.7 Explicit mapping of OPC-UA variable types 17
5.8 Explicit mapping of complex OPC-UA types 19
5.9 Explicit mapping of nested object and variable types 21
5.9.1 General 21
5.9.2 When to use collections 21
5.9.3 When to use menus 21
5.9.4 OPC-UA diagram of nested mapping example 21
5.9.5 EDD snippet of nested mapping example 22
5.10 Explicit mapping of methods 26
5.10.1 Mapping EDD methods to OPC-UA objects, variables or properties 26
5.10.2 Mapping EDD methods to OPC-UA methods 26
5.11 Explicit mapping of alarms 28
5.12 Explicit mapping of units 35
5.13 Explicit mapping of aggregated data by methods 36
5.14 Explicit mapping with reference types 38
5.14.1 General 38
5.14.2 Example: Adding an additional property to an instance of variable 39
5.14.3 Example: Adding an additional variable to an instance of a variable or an object 40
5.14.4 Example: Adding an OPC-UA alias to a variable 42
6 Implicit rules 44
6.1 BrowseName of OPC-UA object 44
6.1.1 General 44
6.1.2 Overruling of BrowseName implicit rule 44

6.2 DisplayName of OPC-UA object 44
6.2.1 General 44
6.2.2 Overruling of DisplayName implicit rule 44
6.3 HANDLING and AccessLevel 44
6.4 VALIDITY and availability 44
6.5 Return values of EDD methods 45
6.5.1 EDD methods mapped to OPC-UA objects, variables, properties or attributes 45
6.5.2 EDD methods mapped to OPC-UA methods 45
6.6 Units 45
6.7 Range 45
6.8 Forward cast 45
6.9 Backward cast 45
6.10 Abstract OPC-UA types 45
6.11 Unmapped mandatory OPC-UA properties, components and folders 46
6.12 Semantic Identifiers and Dictionary References 46
6.13 Implicit Type Casts for OPC-UA DataTypes 47
7 Mapping the EDD device model into PA-DIM (OPC 30081) 48
7.1 General 48
7.2 Explicit mapping of sub-devices 48
7.3 Adding sub-devices 48
Bibliography 52

Figure 1 – OPC UA Graphical notation for NodeClasses 8
Figure 2 – OPC UA Graphical notation for References 9
Figure 3 – OPC UA Graphical notation example 9
Figure 4 – Optimized Type Reference 10
Figure 5 – Similarity of OPC-UA objects and EDD collections 10
Figure 6 – Syntax of sematic ids for EDD entry points 12
Figure 7 – Syntax of sematic id for alias mapping 12
Figure 8 – Namespace Collection 14
Figure 9 – Reference Type Collection 15
Figure 10 – Use of SEMANTIC_MAP for OPC-UA types 16
Figure 11 – Syntax of OPC-UA type identifier 16
Figure 12 – Semantic map example 16
Figure 13 – Syntax of Unit Identifier 17
Figure 14 – Most simple mapping example 17
Figure 15 – EDD variable mapped to an OPC-UA BaseDataVariableType 18
Figure 16 – Simple mapping example with range and unit 19
Figure 17 – Mapping of a collection to an OPC-UA variable 20
Figure 18 – Nested objects and variables 22
Figure 19 – Example of nested objects and variables 25
Figure 20 – Simple method 26
Figure 21 – Example of simple method mapping 28
Figure 22 – Supported Alarms 32
Figure 23 – Example of alarm mapping 35

Figure 24 – EDD example of explicit unit mapping 36
Figure 25 – Instance of PADIMType 37
Figure 26 – Method of type DD_STRING mapped to a string variable 38
Figure 27 – Adding a property which is not defined in mapped type 39
Figure 28 – EDD example of adding a property 40
Figure 29 – Adding a component 41
Figure 30 – EDD example adding a component 42
Figure 31 – Adding an alias 43
Figure 32 – EDD example adding an alias 43
Figure 33 – Explicit mapping of dictionary entries 46
Figure 34 – Combination with an implicitly mapped dictionary entry 47
Figure 35 – Subdevices 49
Figure 36 – Subdevices example 51

Table 1 – Alarm properties mapping 29
Table 2 – Implicit Type Casts 47

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

FIELD DEVICE INTEGRATION (FDI®) –

Part 8: EDD to OPC-UA Mapping

FOREWORD

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

IEC 62769-8 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation. It is an International Standard.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65E/851/CDV	65E/909/RVC

Full information on the voting for its approval can be found in the report on voting indicated in the above table.

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

A list of all parts in the IEC 62769 series, published under the general title *Field device integration (FDI®)*, can be found on the IEC website.

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

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

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

FIELD DEVICE INTEGRATION (FDI®) –

Part 8: EDD to OPC-UA Mapping

1 Scope

This part of IEC 62769 specifies how the internal view of a device model represented by the EDD can be transferred into an external view as an OPC-UA information model by mapping EDD constructs to OPC-UA objects.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE For undated references, the edition of the referenced document (including any amendments), which applies for a specific FDI®[1] Technology Version is defined within the FDI® Technology Management Document and on the support portals of FieldComm Group and PI International.

IEC 61804-3, *Devices and integration in enterprise systems − Function blocks (FB) for process control and electronic device description language (EDDL) − Part 3: EDDL syntax and semantics*

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

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

IEC 62541-5, *OPC Unified Architecture – Part 5: Information Model*

IEC 62541-8, *OPC Unified Architecture – Part 8: Data Access*

IEC 62541-9:2020, *OPC Unified Architecture – Part 9: Alarms and Conditions*

OPC 10000-17, *OPC Unified Architecture – Part 17: Alias Names*

OPC 10000-19, *OPC Unified Architecture – Part 19: Dictionary Reference*

IEC 62541-100, *OPC unified architecture – Part 100: Device Interface*

IEC 62769-1, *Field Device Integration (FDI®) – Part 1: Overview*

IEC 62769-5, *Field Device Integration (FDI®) – Part 5: FDI® Information Model*

IEC 62769-6, *Field Device Integration (FDI®) – Part 6: FDI® Technology Mappings*

---

[1] FDI® is a registered trademark of the non-profit organization Fieldbus Foundation, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trade name. Use of the trade name requires permission of the trade name holder.

ISO/IEC 11179-6, *Information technology – Metadata registries (MDR) – Part 6: Registration*

OPC 30081, *Process Automation Devices – PADIM*

UN/CEFACT, UNECE Recommendation 20, Codes for Units of Measure Used in International Trade
available at https://www.unece.org/cefact/codesfortrade/codes_index.html [viewed 2023-02-07]

3 Terms, definitions, abbreviated terms, acronyms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62769-1, IEC 62769-5, IEC 62769-6, IEC 62541-3, IEC 62541-4, IEC 62541-5, IEC 62541-8, IEC 62541-9, OPC 10000-17, IEC 62541-100, and OPC 30081 apply.

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

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

3.2 Abbreviated terms and acronyms

For the purposes of this document, the abbreviated terms and acronyms given in IEC 62769-1 as well as the following apply.

PA-DIM Process Automation Device Information Model

3.3 Conventions

3.3.1 Capitalization

Capitalization of the first letter of words is used in the IEC 62769 series to emphasize an FDI® defined term.

3.3.2 Graphical notation

OPC UA defines a graphical notation for an OPC UA AddressSpace. It defines graphical symbols for all NodeClasses and how different types of References between Nodes can be visualized. Figure 1 shows the symbols for the NodeClasses used in this document. NodeClasses representing types always have a shadow.

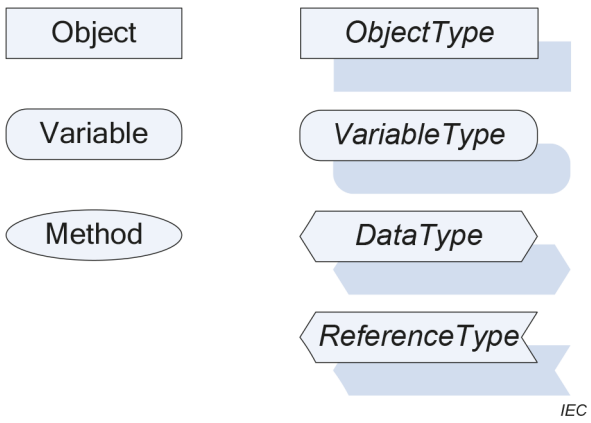


Figure 1 – OPC UA Graphical notation for NodeClasses

Figure 2 shows the symbols for the ReferenceTypes used in this document. The Reference symbol is normally pointing from the source Node to the target Node. The only exception is the HasSubType Reference. The most important References such as HasComponent, HasProperty, HasTypeDefinition and HasSubType have special symbols avoiding the name of the Reference. For other ReferenceTypes or derived ReferenceTypes, the name of the ReferenceType is used together with the symbol.

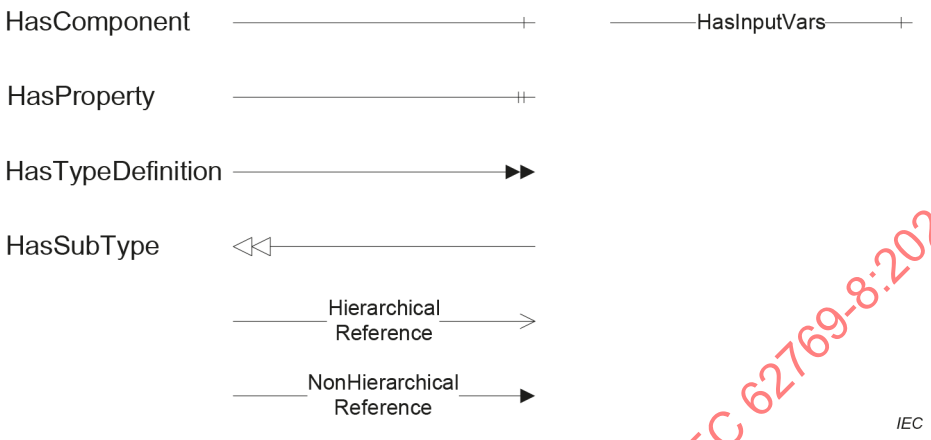


Figure 2 – OPC UA Graphical notation for References

Figure 3 shows a typical example for the use of the graphical notation. Object_A and Object_B are instances of the ObjectType_Y indicated by the HasTypeDefinition References. The ObjectType_Y is derived from ObjectType_X indicated by the HasSubType Reference. The Object_A has the components Variable_1, Variable_2 and Method_1.

To describe the components of an Object or the ObjectType, the same NodeClasses and References are used on the Object and on the ObjectType such as for ObjectType_Y in the example. The Nodes used to describe an ObjectType are instance declaration Nodes.

To provide more detailed information for a Node, a subset or all Attributes and their values can be added to a graphical symbol (see for example Variable_1, the component of Object_A in Figure 3).

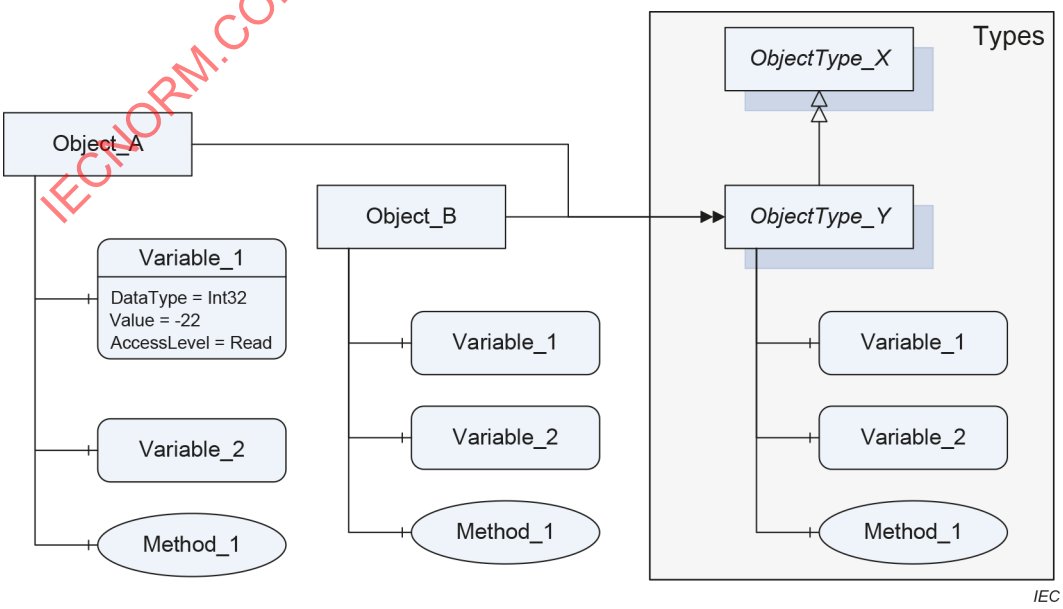


Figure 3 – OPC UA Graphical notation example

To improve readability, this document frequently includes the type name inside the instance box rather than displaying both boxes and a reference between them. This optimization is shown in Figure 4.

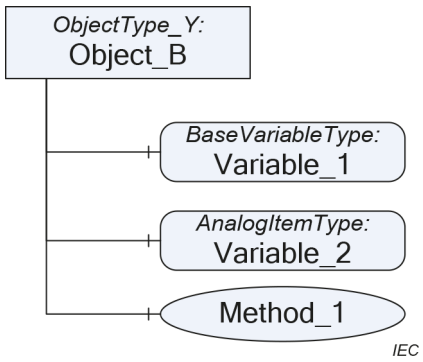


Figure 4 – Optimized Type Reference

4 Overview

There are two types of mapping mechanisms – explicit and implicit mapping. Explicit mapping is provided by the EDD constructs SEMANTIC_MAP in combination with COLLECTIONS, METHODS and VARIABLES including the mapping of enumerations and is done by the EDD-developer. Implicit mapping is provided by the implementation of the OPC-UA server in conjunction with a FDI® Server and covers definitions like default casts (e.g. any number value to float64) or even the mapping of complete lists of unit codes from a fieldbus domain (HART HART®[2], PROFIBUS®[3], ...) into an OPC-UA domain (e.g. UNECE, CDD, ...).

Looking at OPC-UA objects containing named data items like attributes, properties, variables and other objects, the most similar EDD objects are collections containing named data items like variables, menus and other collections (see Figure 5).

Figure 5 – Similarity of OPC-UA objects and EDD collections

In fact, the EDD construct collection is the key element for the basic principle of how EDD device model data shall be mapped into an OPC-UA information model. The following clauses

---

2 HART® is a registered trademark of the non-profit organization FieldComm Group, Inc. This information is given for convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trademark. Use of the trademark requires permission of the trademark holder.

3 PROFIBUS® is the registered trademark of the non-profit organization PROFIBUS Nutzerorganisation e.V. (PNO). This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trademark. Use of the trademark requires permission of the trademark holder.

describe and define how this shall be done. Therefore, this document has normative character for EDD developers and developers of OPC-servers accessing an FDI® Server to publish an OPC-UA information model according to a specific OPC-UA namespace and the mapping information provided by the EDD.

Clause 5 covers in detail how the mapping works in principle and for any OPC-UA information model based on any namespace.

Based on Clause 5, Clause 7 describes some details how to map the EDD into PA-DIM. Additional normative definitions how to map fieldbus specific data (e.g. identification) into PA-DIM is provided by the FDI-Profile specifications for HART, FF, PROFINET®[4], PROFIBUS, Modbus®[5] and ISA100 Wireless®[6].

Before starting with doing some EDD mapping, it is strongly recommended to get a basic understanding of OPC-UA concerning how object types, variable types and reference types are defined.

5 Basic principles of explicit mapping

5.1 Semantic maps to tag EDD constructs

For not having to use naming conventions for EDD constructs to link a specific purpose to an EDD construct, semantic maps shall be used to kind of tag an EDD construct by a specifically defined semantic id. For the time being, three syntax definitions exist for three specific purposes. The details of how to use them will be explained in the corresponding context.

The syntax of semantic ids for EDD entry points is illustrated in Figure 6.

---

4 PROFINET® is the registered trademark of the non-profit organization PROFIBUS Nutzerorganisation e.V. (PNO). This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trademark. Use of the trademark requires permission of the trademark holder.

5 Modbus® is a registered trademark of Schneider Electric USA, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trademark. Use of the trademark requires permission of the trademark holder.

6 ISA100 Wireless® is the registered trademark of the non-profit organization Automation Standards Compliance Institute. This information is given for convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trademark. Use of the trademark requires permission of the trademark holder.

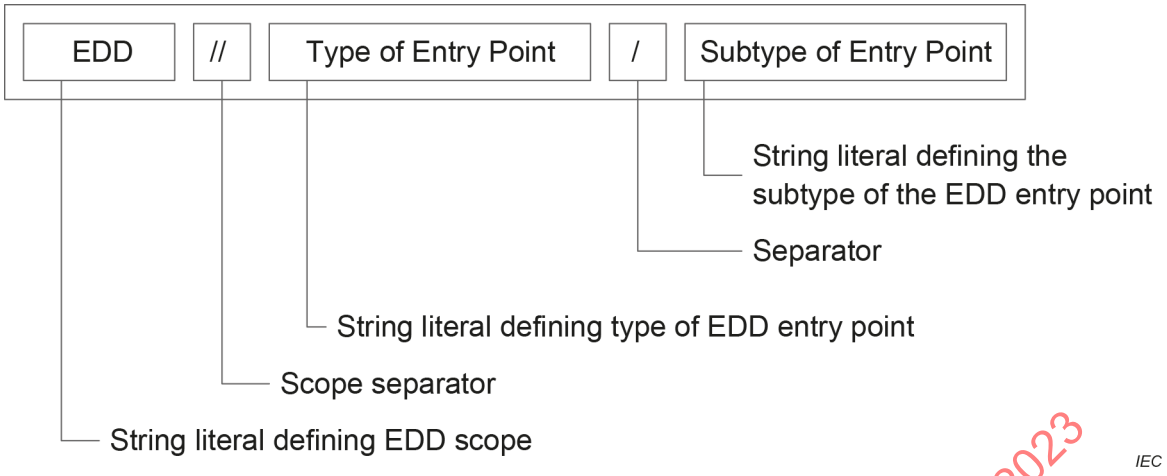


Figure 6 – Syntax of sematic ids for EDD entry points

5.2 Alias collections

5.2.1 General

Alias collections are used to define aliases for lengthy strings having a specific meaning. The alias should be short and shall be unique across the complete EDD. Aliases will be used as member identifiers or as a part of member identifiers or in SEMANTIC_MAPs for OPC-UA type mapping. Alias collections are the main entry points to resolve EDD to OPC-UA mapping and shall not depend on each other (see 5.3 and 5.4).

5.2.2 Syntax of semantic id for alias mappings

The syntax of semantic id for alias mapping is illustrated in Figure 7.

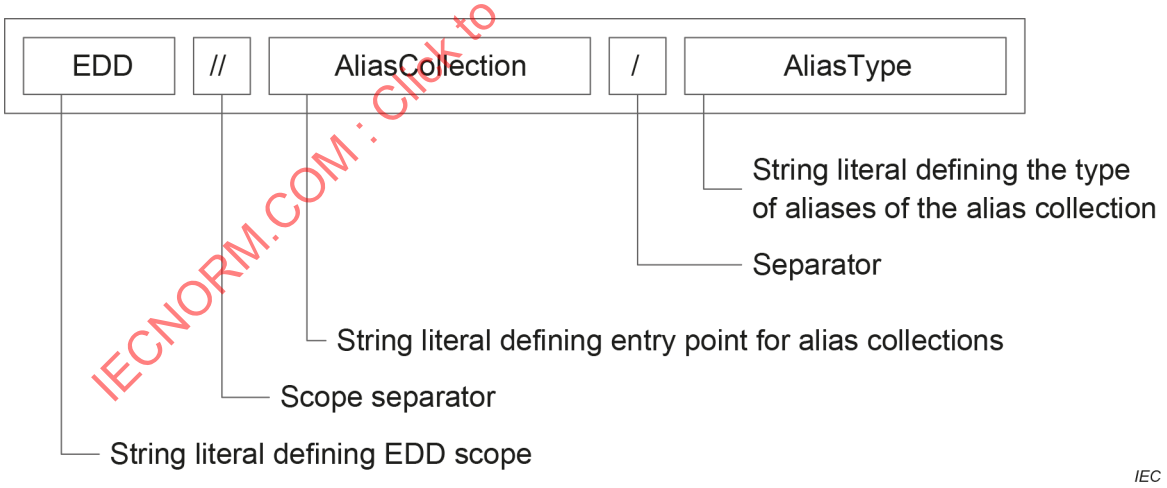


Figure 7 – Syntax of sematic id for alias mapping

5.3 Namespace Alias Collection

To prevent name conflicts, any OPC-UA type is defined in scope of a specific namespace. A namespace collection and appropriate variables shall be defined to provide suitable names for the namespace URIs. The member identifier of the collection item shall be used as an alias wherever a namespace identifier is needed. In the following example "__UA_" represents the namespace http://opcfoundation.org/UA/ which is defined by the default value of the referenced variable "UA_Namespace".

It is mandatory to define at least one namespace collection for an EDD defining EDD to OPC-UA mapping.

Definition:
The name of the namespace collection shall be tagged by a SEMANTIC_MAP with the following semantic ID:

"EDD//AliasCollection/Namespaces"

The name of the namespace COLLECTION, the name of the SEMANTIC_MAP and the names of the referenced string variables can be freely chosen. The default value of the referenced string variable shall contain a valid namespace.

Figure 8 shows an EDD namespace Collection example.

```
SEMANTIC_MAP Namespace_Map
{
    "EDD//AliasCollection/Namespaces": Namespace_Collection
}

COLLECTION OF VARIABLE Namespace_Collection
{
    LABEL "OPC-UA Namespaces";
    MEMBERS
    {
       __UA_,        UA_Namespace;
       __DI_,        DI_Namespace;
       __PADIM_,     PADIM_Namespace;
       UNECE,        UNECE_Namespace;
    }
}

VARIABLE UA_Namespace
{
    LABEL    "UA namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/";
}

VARIABLE DI_Namespace
{
    LABEL    "DI namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/DI/";
}

VARIABLE PADIM_Namespace
{
    LABEL    "PADIM namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/PADIM/";
}

VARIABLE UNECE_Namespace
{
    LABEL    "UNECE namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://www.opcfoundation.org/UA/units/un/cefact/";
}
```


Figure 8 – Namespace Collection

5.4 Reference Type Alias Collection

For the use of OPC-UA references where the specific reference type cannot be unambiguously derived from the two objects the reference is connecting, the reference type shall be explicitly defined.

This collection is only mandatory if mappings as described in 5.14 are used.

The name of the reference type collection shall be tagged by a SEMANTIC_MAP with the following semantic id:

"EDD//AliasCollection/ReferenceTypes"

The name of the reference type COLLECTION, the name of the SEMANTIC_MAP and the names of the referenced string variables can be freely chosen. The default value of the referenced string variable shall contain a valid nodeid of a reference type.

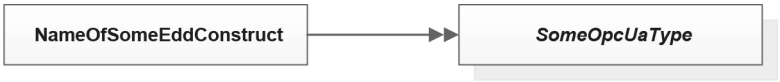
```
SEMANTIC_MAP ReferenceType_Map

{
    "EDD//AliasCollection//ReferenceTypes": ReferenceType_Collection
}

COLLECTION OF VARIABLE ReferenceType_Collection
{
     LABEL "OPC-UA reference types"
     MEMBERS
     {
          __HasAlias,       HasAlias_Definition;
          __HasComponent,   HasComponent_Def;
          __HasProperty,    HasProperty_dfn;
     }
}

VARIABLE HasAlias_Definition
{
    LABEL     "HasAlias reference type";
    HELP      [blank];
    CLASS     LOCAL;
    HANDLING READ;
    TYPE      ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/HasAlias";
}

VARIABLE HasProperty_dfn
{
    LABEL     "HasProperty reference type";
    HELP      [blank];
    CLASS     LOCAL;
    HANDLING READ;
    TYPE      ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/HasProperty";
}


VARIABLE HasComponent_Def
{
    LABEL     "HasComponent reference type";
    HELP      [blank];
    CLASS     LOCAL;
    HANDLING READ;
    TYPE      ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/HasComponent";
}
```


Figure 9 – Reference Type Collection

5.5 Semantic maps for OPC-UA type mapping

5.5.1 General

The EDD construct SEMANTIC_MAP shall be used to assign an OPC-UA object type or variable type to an EDD construct (collections in most cases) (see Figure 10). Based on this assignment, the OPC-UA server instantiates an object of the specified object/variable type and fills it with the data of the EDD construct.



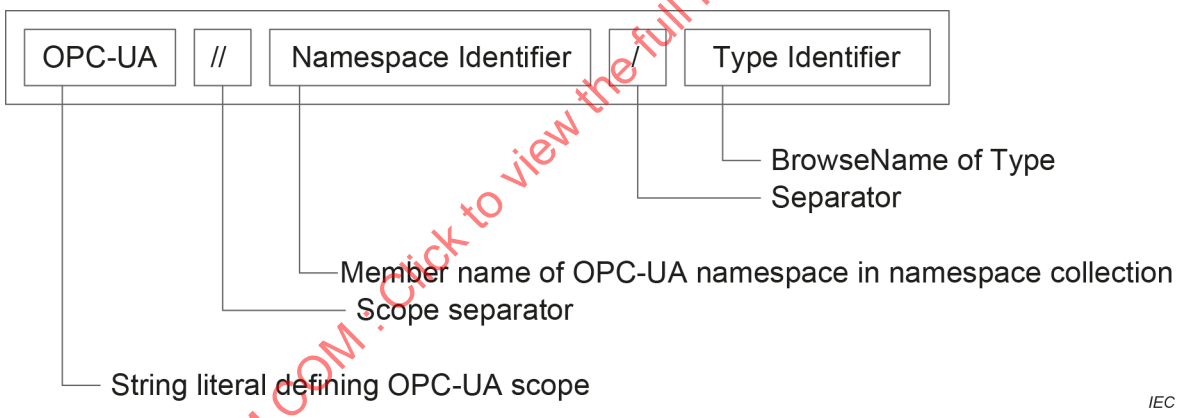

```
SEMANTIC_MAP SomeExplicitTypeMap
{
      "OPC-UA//SomeNamespace/SomeOpcUaType" : NameOfSomeEddConstruct
}
```

IEC

Figure 10 – Use of SEMANTIC_MAP for OPC-UA types

5.5.2 Syntax of semantic Id for OPC-UA

The semantic id referencing the OPC-UA object type shall comply to the syntax illustrated in Figure 11.



IEC

Figure 11 – Syntax of OPC-UA type identifier

In respect of the example from 5.2, a semantic map could look like the example shown in Figure 12.

```
SEMANTIC_MAP PressureVariableMap
{
   "OPC-UA//__PADIM_/PressureMeasurementVariableType" : PressureVariableColl
}
```

IEC

Figure 12 – Semantic map example

In Figure 12, the object type 'PressureMeasurementVariableType' from the namespace 'http://opcfoundation.org/UA/PADIM/' is mapped to an EDD construct with the name 'PressureVariableColl'.

5.6 Semantic maps for unit mapping

5.6.1 General

According to the IEC 61804-3, SEMANTIC_MAP can be used to map semantic ids to unit enumerations of EDD unit variables. The unit ids shall be used in the enumeration of the OPC-UA type that is mapped to EDD unit variable and as unit codes in the EUInformation structure of OPC-UA variables. This definition applies only to unit identifiers not compliant with ISO/IEC 11179-6 naming convention.

5.6.2 Syntax of semantic Id for Units

The semantic id referencing a unit not compliant with ISO/IEC 11179-6 shall comply to the following syntax illustrated in Figure 13.

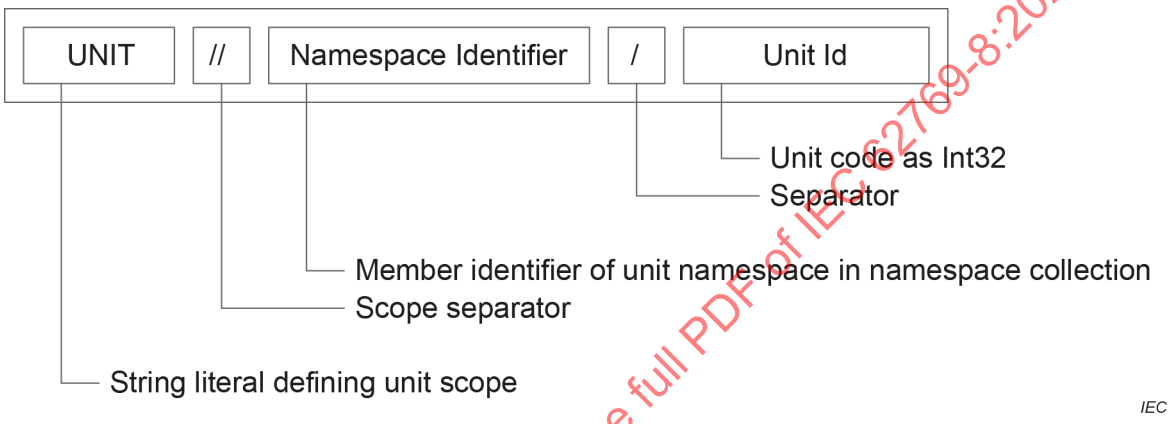


Figure 13 – Syntax of Unit Identifier

5.7 Explicit mapping of OPC-UA variable types

The most basic case maps an EDD variable to an instance of an OPC-UA variable type is illustrated in Figure 14.

Figure 14 – Most simple mapping example

In this case, the value of the EDD variable SMR_HighBlockDistance_2 has been mapped to the value of an OPC-UA variable with the name SMR_HighBlockDistance_2 as an instance of BaseDataVariableType (see Figure 15).

```
SEMANTIC_MAP BlockingDistance_mp
{
    "OPC-UA//__UA_/BaseDataVariableType": SMR_HighBlockDistance_2
}

UNIT SU_Length_UR
{
    SU_Length_1: SMR_HighBlockDistance_2
}

VARIABLE SMR_HighBlockDistance_2
{
       LABEL T_19_BlockingDistance;
       HELP  T_BlockDistance_TT;
       CLASS DEVICE;

       HANDLING IF (HWLock||SILLock||SWLock||WHGLock) {READ;} ELSE {READ & WRITE;}

       TYPE  FLOAT
       {

              MAX_VALUE
                     SELECT (SU_Length_1)
                     {
                            CASE e_Length_milliMeter_LONG:/*Max*/200000.0;
                            CASE e_Length_Meter_LONG:/*Max*/200.0;
                            CASE e_Length_Foot_LONG:/*Max*/656.167979002625;
                            CASE e_Length_Inch_LONG:/*Max*/7874.0157480315;
                     }


              MIN_VALUE
                     SELECT (SU_Length_1)
                     {
                            CASE e_Length_milliMeter_LONG:/*Min*/0.0;
                            CASE e_Length_Meter_LONG:/*Min*/0.0;
                            CASE e_Length_Foot_LONG:/*Min*/0.0;
                            CASE e_Length_Inch_LONG:/*Min*/0.0;
                     }

              EDIT_FORMAT
                     SELECT (SU_DecimalPlacesLength_1)
                     {
                            CASE e_DecimalPlacesSelector_X_LONG: ".0f";
                            CASE e_DecimalPlacesSelector_X_X_LONG: ".1f";
                            CASE e_DecimalPlacesSelector_X_XX_LONG: ".2f";
                            CASE e_DecimalPlacesSelector_X_XXX_LONG: ".3f";
                            CASE e_DecimalPlacesSelector_X_XXXX_LONG: ".4f";
                     }

              DISPLAY_FORMAT
                     SELECT (SU_DecimalPlacesLength_1)
                     {
                            CASE e_DecimalPlacesSelector_X_LONG: ".0f";
                            CASE e_DecimalPlacesSelector_X_X_LONG: ".1f";
                            CASE e_DecimalPlacesSelector_X_XX_LONG: ".2f";
                            CASE e_DecimalPlacesSelector_X_XXX_LONG: ".3f";
                            CASE e_DecimalPlacesSelector_X_XXXX_LONG: ".4f";
                     }

       }
       DEFAULT_VALUE 0;
}
```


Figure 15 – EDD variable mapped to an OPC-UA BaseDataVariableType

Because of an implicit rule (see 6.1), the name of the EDD variable became the BrowseName of the OPC-UA variable. Another implicit rule has been applied for the HANDLING attribute (see 6.3) and an implicit typecast has been applied to map the type of the value attribute of the EDD variable to an appropriate datatype in the OPC-UA object (see 6.13). Besides name, label, value and value datatype, none of the other attributes of the EDD variable could be implicitly mapped to this OPC-UA object – not even the unit. This is because a BaseVariableType does not support units and it also does not support a range with something like a MIN_VALUE or a MAX_VALUE. For this purpose, a more convenient datatype like AnalogUnitRangeType could be used which has additional properties for an engineering range and an engineering unit.

```
SEMANTIC_MAP BlockingDistance_mp
{
    "OPC-UA//__UA_/AnalogUnitRangeType" : SMR_HighBlockDistance_2
}
```

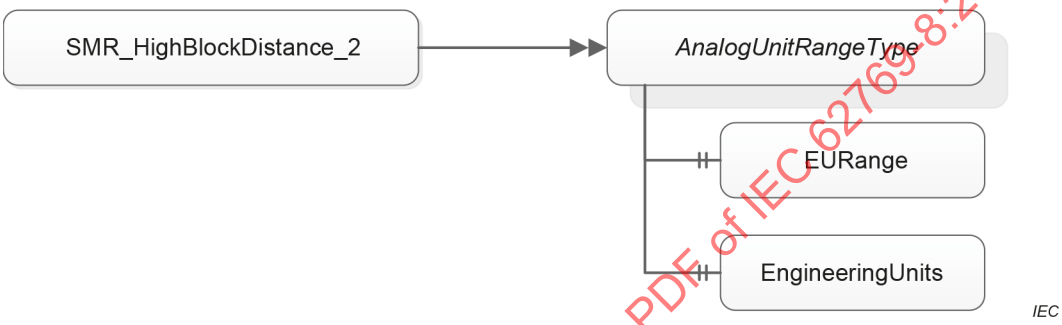


Figure 16 – Simple mapping example with range and unit

In this case, two additional implicit rules were applied, one for the unit (see 6.5) and one for the range (see 6.7). If the default mapping of the implicit rules does not suit the needs, these rules could be overwritten by some more explicit mapping (see 5.8 and 5.9).

5.8 Explicit mapping of complex OPC-UA types

For OPC-UA types which contain more than one attribute, object, variable or property which cannot be implicitly mapped by some implicit mapping rule or if the implicit mapping needs to be overruled, an EDD collection is mapped to an instance of an OPC-UA object or variable type.

In the example shown in Figure 17, an instance of PressureMeasurementVariableType from the namespace http://opcfoundation.org/UA/PADIM/ is mapped to the collection ‘Pressure_1’.

```
SEMANTIC_MAP PressureVariableMap
{
   "OPC-UA//__PADIM_/PressureMeasurementVariableType" : Pressure_1
}

COLLECTION Pressure_1
{
   LABEL "PV";
   HELP [pressure_help];
   MEMBERS
   {
        __UA_Value,                press.DIGITAL_VALUE;
        __PADIM_ActualValue,       pressureInternalValue;
        __PADIM_SimulationValue,   pressureOverrideValue;
        __PADIM_SimulationState,   pressureSimulation;
        __PADIM_Damping,           press.DAMPING;
   }
}
```

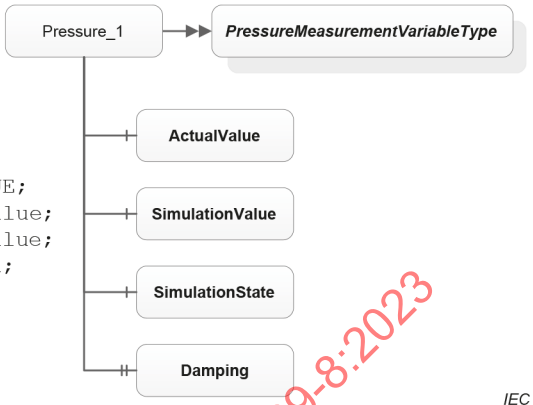


Figure 17 – Mapping of a collection to an OPC-UA variable

Prefixed member identifiers (e.g. __PADIM_Value, __PADIM_ActualValue, ...) of the collection members are used to map the referenced EDD objects (e.g. press.DIGITAL_VALUE, pressureInternalValue, etc.) to the properties, variables or components of the OPC-UA object by their corresponding BrowseNames. For 'ActualValue', 'SimulationValue', 'SimulationState' and 'Damping', the mapping is very self-explaining.

Although a value is not explicitly shown in the OPC-UA type on the right side of the picture, it exists as an attribute with the name 'Value' being an integral part of every variable type. This attribute is specified in the OPC-UA base specification and therefore the prefix __UA_ is used according to the definition of the namespace collection (see 5.3).

See IEC 62541-3 for the complete definition of attributes for variable types.

Prefixing the member name with the namespace alias guarantees that the EDD objects are unambiguously assigned to the corresponding item on the OPC-UA side even in case of identical BrowseNames (but belonging to different namespaces). As a side effect, for HART-EDDs the namespace identifier can be chosen in a way not conflicting with already existing member names (member identifier).

The order of the EDD collection members does not need to be the same as the order of the instance declarations (properties, variables, attributes, ...) of the OPC-UA type.

This mapping mechanism implies that the mapped data items belong to an OPC-UA object or variable type as attributes or are connected to the type by a hasComponent or a hasProperty reference type. Other object relations based on other reference types shall be declared explicitly. This is also true for additional components or properties which do not belong to the original object type mapped to the collection. See 5.14 for further details.

Furthermore, implicit mapping rules do apply the same way as if the mapping had been done directly as shown in 5.7. For this specific example, this means that the unit and the MIN_VALUE/MAX_VALUE of the variable behind 'press.DIGITAL_VALUE' is implicitly mapped to the 'EngineeringUnit' respectively 'EURange' of 'PressureMeasurementVariableType' which indirectly derives from AnalogUnitRangeType.

5.9 Explicit mapping of nested object and variable types

5.9.1 General

Defining only one collection might not be enough to map all information available by an EDD object to a complex object or variable type. By defining additional, nested EDD collections and menus, where each collection/menu refers to an OPC-UA instance declaration of a complex type, the nesting of the EDD collections/menu reflects the nesting relations of the OPC-UA object respectively OPC-UA variable types.

If the mapped OPC-UA object type represents a complete information model (e.g. PADIMType), the nesting might stretch across several levels. The following example shall demonstrate how this can be done – when to use EDD collections and when to use EDD menus (see Figure 17).

While setting up the collections and menus, care shall be taken to distinguish between composition and derivation. While composition always leads to an additional nesting level, derivation leads to additional member identifiers on the same level as the rest of the mapped type; these reference instance declarations are not obviously seen just looking at the derived type.

5.9.2 When to use collections

Whenever a clearly defined one to one relation between instance declarations and data items or data containers exists and where the names of the instance declarations can be unambiguously assigned to exactly one EDD construct, a collection shall be used to represent the parent type of these instance declarations. See 5.8 for an example.

5.9.3 When to use menus

Whenever it is not possible to unambiguously assign the name of an instance declaration to exactly one data item or data container, an EDD menu shall be mapped to the instance declaration to serve as container for a multitude of data items or data containers which are all of the same type as defined by the instance declaration the menu is mapped to.

For example, this is true for instance declarations of type ‘FolderType’ or ‘SignalSetType’. In the following example, this has been applied for the ‘SignalSet’ of ‘PADIMType’ (for details about PADIMType, see OPC 30081).

5.9.4 OPC-UA diagram of nested mapping example

Figure 18 shows an example of nested mapping of objects and variables.

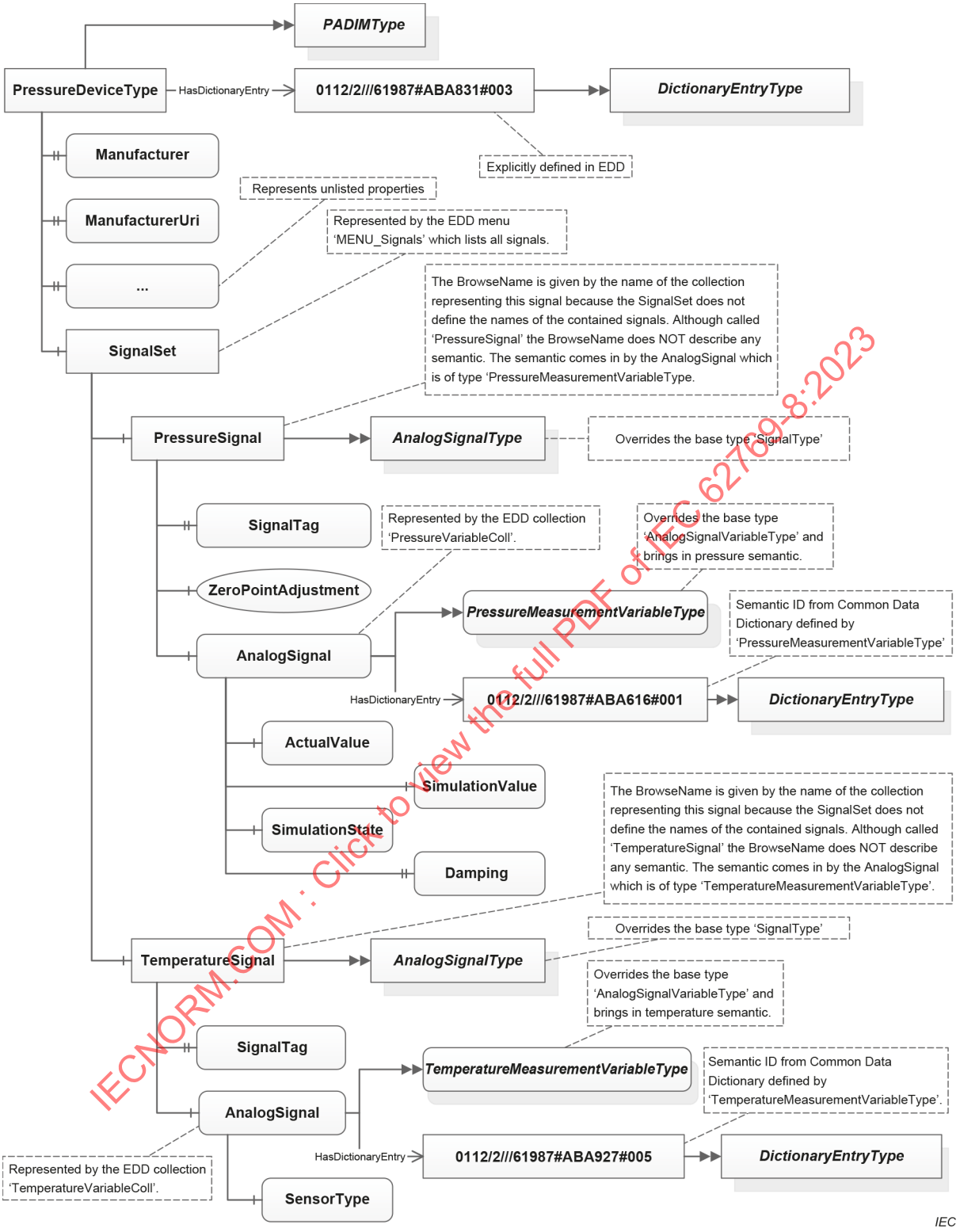


Figure 18 – Nested objects and variables

5.9.5 EDD snippet of nested mapping example

5.9.5.1 General

This example illustrates how EDD constructs shall be used to map parts of the internal device model of an EDD to a complex information model in OPC-UA (e.g. PA-DIM). Although the same principles apply to map OPC-UA events and alarms, they are described in a separate clause for not bloating this example.

5.9.5.2 Explicit mapping of ranges

Although ‘EngineeringUnits’ of the ‘AnalogUnitRangeType’ is implicitly mapped to the unit of the mapped EDD variable, the ‘EURange’ respectively ‘InstrumentRange’ default to an inadequate MIN_FLOAT respectively MAX_FLOAT if no MIN_VALUE/MAX_VALUE has been defined in the EDD variable. In the following examples, two additional collections are defined to explicitly map the ranges ‘EURange’ and ‘InstrumentRange’. These two ranges are defined by ‘AnalogUnitRangeType’ which is an indirect base type of ‘PressureMeasurementVariableType’.

The example assumes that some local variables have been defined for the upper respectively lower limits of the referenced ranges.

5.9.5.3 Overriding the type of an instance declaration

The SEMANTIC_MAP ‘AnalogSignalMap’ is necessary because the instance declaration ‘SignalSet’ is type of ‘SignalType’ and the two signals for pressure and temperature are types of ‘AnalogSignalType’.

Because the ‘AnalogSignal’ of the ‘AnalogSignalType’ is a generic base type (AnalogSignalVariableType) which does not reflect any measurement principle, the additional SEMANTIC_MAP ‘PressureVariableMap’ is used to override the base type with the ‘PressureMeasurementVariableType’ to reflect the pressure measurement principles. This only works if the overriding type derives directly or indirectly from the base type. The same is true for ‘TemperatureVariableMap’.

5.9.5.4 Further hints

Enclosing ‘AnalogSignaVariableType’ with ‘AnalogSignalType’ is necessary to bring the methods of the object type ‘AnalogSignalType’ into the context of the variable type ‘AnalogSignalVariableType’ because in OPC-UA it is not allowed to add methods directly to variable types (for historical reasons).

```
/* Assigns pressure device semantic and OPC-UA PADIMType */
SEMANTIC_MAP DeviceTypeMap
{
    "OPC-UA//__PADIM_/PADIMType", "0112/2///61987#ABA831#003": PressureDeviceType
}

/* Although called 'PressureDeviceType' the collection name does NOT add any semantic */
/* The semantic comes in by the semantic map above.                                    */
COLLECTION PressureDeviceType
{
    LABEL "Pressure devicetype";
    MEMBERS
    {
       __DI_Manufacturer,            DeviceManufacturer_local;
       __DI_ManufacturerUri,  DeviceManufacturerUrl;
       __DI_Model,                   DeviceNameString;
       __DI_SerialNumber,            DeviceSerialNumber;
       __DI_ProductCode,             DeviceOrderCode;
       __DI_HardwareRevision, PaPbHardwareRevision;
       __DI_SoftwareRevision, PaPbSoftwareRevision;
       __DI_RevisionCounter,  ConfigurationCounter;
       __DI_ProductInstanceUri,      METH_CalcInstanceUri;
       __DI_AssetId,                 PaPbDescriptor;
       __DI_DeviceHealth,            METH_CalcDeviceHealthStatus;
       __DI_DeviceHealthAlarms,      MENU_DeviceHealthAlarms;
       __PADIM_SignalSet,            MENU_Signals;
    }
}

/* Signals from PA-DIM SignalSet    */
MENU MENU_Signals
```

```
{
    LABEL "SignalSet";
    ITEMS
    {
        PressureSignal,
        TemperatureSignal
    }
}

/* Defines OPC-UA type for both signals */
SEMANTIC_MAP AnalogSignalMap
{
    "OPC-UA//__PADIM_/AnalogSignalType": PressureSignal, TemperatureSignal
}

/* Eplicitly mapped to the __PADIM_ type 'AnalogSignalType'*/
COLLECTION PressureSignal
{
   VALIDITY IF (OperatingMode==0) {TRUE;} ELSE { FALSE;}
   LABEL "Pressure signal";
   MEMBERS
   {
        __PADIM_SignalTag,                  PrimValDesc;
        __PADIM_AnalogSignal,         PressureVariableColl;
        __PADIM_ZeroPointAdjustment, SetPressPv2Zero;
   }
}

SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/PressureMeasurementVariableType": PressureVariableColl
}

/* Eplicitly mapped to the __PADIM_ type 'PressureMeasurementVariableType'*/
COLLECTION PressureVariableColl
{
   LABEL "Pressure PV";
   MEMBERS
   {
        __UA_Value,                 PaTbPrimaryValueValue;
        __PADIM_ActualValue,   PaTbCurrentValue1Value;
        __PADIM_SimulationValue,      Pressure1SimulationValue;
        __PADIM_SimulationState,      SimulationMode;
        __PADIM_Damping,              ActivePressure1Damping;
        __UA_InstrumentRange,  COL_InstRange_PressVal;
        __UA_EURange,                 COL_ProcRange_PressVal;
   }
}

/* Implicitly mapped to the __UA_ type 'AnalogUnitRangeType' */
COLLECTION COL_PressureValue
{
LABEL "Pressure value";
   HELP [blank];
   MEMBERS
   {
          __UA_Value,                 PaTbPrimaryValueValue;
          __UA_InstrumentRange,       COL_InstRange_PressVal;
          __UA_EURange,               COL_ProcRange_PressVal;
   }
}

/* Implicitly mapped to the __UA_ structure 'Range' */
COLLECTION COL_InstRange_PressVal
{
LABEL "Pressure value instrument range";
   HELP [blank];
   MEMBERS
   {
        __UA_low,       Pressure1SensorLowLimit;
```

```
        __UA_high,      Pressure1SensorHighLimit;
    }
}

/* Implicitly mapped to the __UA_ structure 'Range' */
COLLECTION COL_ProcRange_PressVal
{
   LABEL "Pressure value process range";
   HELP [blank];
   MEMBERS
   {
        __UA_low,       PaTbScaleIn0;
        __UA_high,      PaTbScaleIn100;
   }
}

/* Eplicitly mapped to the __PADIM_ type 'AnalogSignalType'*/
COLLECTION TemperatureSignal
{
   VALIDITY IF (OperatingMode==1) {TRUE;} ELSE { FALSE;}
   LABEL "Temperature signal";
   MEMBERS
   {
        __PADIM_SignalTag,           SecValDesc;
        __PADIM_AnalogSignal, TemperatureVariableColl;
   }
}

SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/TemperatureMeasurementVariableType": TemperatureVariableColl
}

/* Eplicitly mapped to the __PADIM_ type 'TemperatureMeasurementVariableType'        */
/* Optional components were not available and therefore not mapped                   */
COLLECTION TemperatureVariableColl
{
   LABEL "Temperature SV";
   MEMBERS
   {
      __UA_Value,            VarTemperatureValue;
      __PADIM_SensorType,    VarSensType;
   }
}
/* This method returns a DD_STRING that contains the ProductInstanceUri which       */
/* (obviously) does not exist in the device as such and therefore must calculated    */
/* based on three variables where at least the 'DeviceSerialNumber' must be read from */
/* the device to make it unique.                                                     */
/* The other two variables could be local variables with adequate default values.    */
/*                                                                                   */
/* Note:                                                                             */
/* Including a device name is not really necessary if the serial number is guaranteed */
/* to be unique for the ManufactureUrl                                               */
METHOD METH_CalcInstanceUri
{
    LABEL"Calculate product instance URI";
    TYPE DD_STRING;
    DEFINITION
    {
         DD_STRING uri;

         uri = DeviceManufacturerUrl + "/" + DeviceNameString + "/" +
DeviceSerialNumber;
         return uri;
    }
}
```

Figure 19 – Example of nested objects and variables

5.10 Explicit mapping of methods

5.10.1 Mapping EDD methods to OPC-UA objects, variables or properties

EDD methods can be used to calculate values on demand. This is also useful if a certain data item does not exist in the EDD as expected by a the OPC-UA information model. By mapping an EDD method to an OPC-UA data item, the return value will be filled into the value of the data item.

See in Figure 19 for 'ProductInstanceUri' which does not exist as proper value neither in the EDD nor in the device (see 6.5).

5.10.2 Mapping EDD methods to OPC-UA methods

Basically, the mapping of methods works in a similar way as the mapping of complex types. Because a method is always belonging to an instance of an object type, a simple method without any arguments and without any return values as in Figure 20 is mapped the same way as described in 5.8.

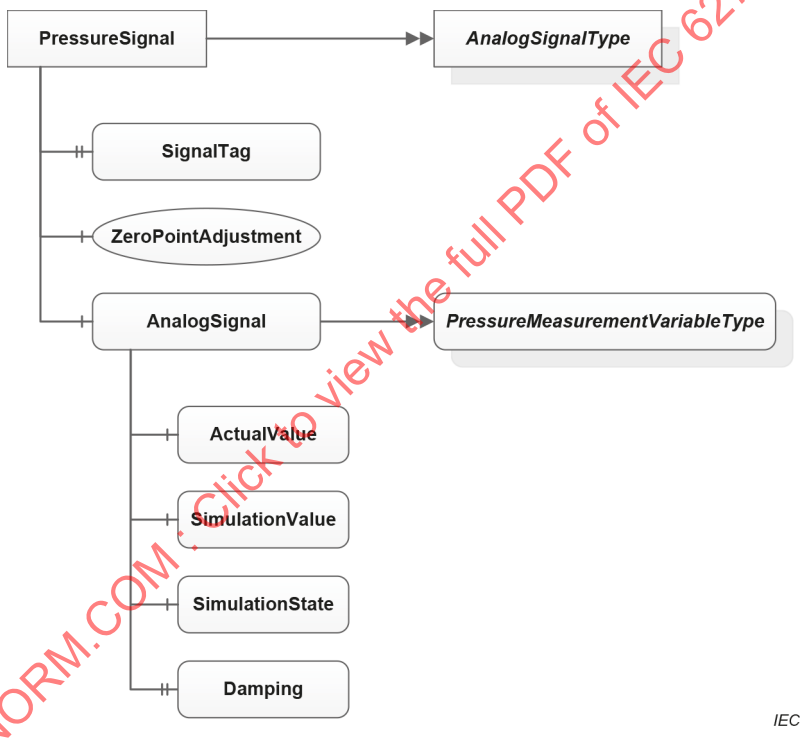


Figure 20 – Simple method

Nevertheless, 'Common Service Result Codes' shall always be returned (e.g. Good, Bad_InternalError Bad_CertificateInvalid, Bad_DataTypeIdUnknown, Bad_Shutdown, Bad_Timeout, etc.) as status code of the response header. Service result codes in the response header are provided by the OPC-UA server and therefore do not relate to the inner EDD-method operation/execution. Any operational status produced by the method execution shall be returned as return value of the method and though shall be set in the method definition accordingly (see 6.5).

Figure 19 shows an EDD example of simple method mapping.

```
/* Defines OPC-UA type for the PressureSignal*/
SEMANTIC_MAP PressureSignalMap
{
    "OPC-UA//__PADIM_/AnalogSignalType": PressureSignal
}

/* Eplicitly mapped to the __PADIM_ type 'AnalogSignalType'*/
COLLECTION PressureSignal
{
   VALIDITY IF (OperatingMode==0) {TRUE;} ELSE { FALSE;}
   LABEL "Pressure signal";
   MEMBERS
   {
       __PADIM_SignalTag,                PrimValDesc;
       __PADIM_AnalogSignal,             PressureVariableColl;
       __PADIM_ZeroPointAdjustment,      SetPressPv2Zero;
   }
}
METHOD SetPressPv2Zero
{
    LABEL     "Sets the current PV to zero";
    TYPE      UNSIGNED_INTEGER(2);
    DEFINITION
    {
              UNSIGNED_INTEGER(2) result;
              BOOLEAN timeout, succeeded;

              /* initialize result with 'BadUnexpectedError' */
              result =  = 0x80010000;

              timeout   = TRUE;
              succeeded = FALSE;

              /*********************************************************/
              /*                                                       */
              /* do something and set booleans accordingly             */
              /*                                                       */
              /*********************************************************/


              if (timeout == true)
              {
                  /* set result to 'BadTimeout' */
                  result =  = 0x800A0000;
              }
              else if (succeeded)
              {
                  /* set result to 'Good' */
                  result =  = 0x00000000;
              }

              return result;
    }
}


SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/PressureMeasurementVariableType": PressureVariableColl
}

/* Eplicitly mapped to the __PADIM_ type 'PressureMeasurementVariableType'*/
COLLECTION PressureVariableColl
{
   LABEL "Pressure PV";
   MEMBERS
   {
       __UA_Value,                PaTbPrimaryValueValue;
       __PADIM_ActualValue,       PaTbCurrentValue1Value;
```

```
        __PADIM_SimulationValue,   Pressure1SimulationValue;
        __PADIM_SimulationState,   SimulationMode;
        __PADIM_Damping,           ActivePressure1Damping;
        __UA_InstrumentRange,      COL_InstRange_PressVal;
        __UA_EURange,              COL_ProcRange_PressVal;
    }
}
```

Figure 21 – Example of simple method mapping

5.11 Explicit mapping of alarms

Only the four alarm types derived from *DeviceHealthDiagnosticAlarmType* defined by IEC 62541-100 shall be supported with the first release of this document (see Figure 22 and Figure 23). More alarm types will be added as required with future releases.

- FailureAlarmType,
- CheckFunctionAlarmType,
- OffSpecAlarmType,
- MaintenanceRequiredAlarmType.

These alarm types reflect the namur status according to NE 107 [1][7].

Alarm types are mapped the same way as object and variable types (see 5.9).

Although all four alarms can be enabled, only one alarm shall be active the same time.

Alarms and their events shall not be set or fired by the configuration of an offline dataset.

NOTE All four device health alarms derive directly from *DeviceHealthDiagnosticAlarmType* without adding any additional properties. The same is true for *DeviceHealthDiagnosticAlarmType* which derives directly from *InstrumentDiagnosticAlarmType*.

Table 1 describes how to handle all the mandatory properties and some of the optional properties. The concept behind the definitions how to handle these properties is based on the approach that ActiveState is the most important state that needs to be updated according to the current state of the device while at the same time all alarm objects are visible in the address space.

7 Numbers in square brackets refer to the Bibliography.

Table 1 – Alarm properties mapping

	DeviceHealthDiagnosticAlarmType		
	InstrumentDiagnosticAlarmType		
	OffNormalAlarmType		
	BrowseName	Value	Set by
	NormalState	NULL	Host
	DiscreteAlarmType		
	AlarmConditionType		
	BrowseName	Value	Set by
	ActiveState	Shall be explicitly mapped to a boolean variable reflecting the state of this alarm. Only one of the four supported DeviceHealthDiagnosticAlarms shall be active at the same time. True = active False = inactive The localized text of TrueState respectively FalseState shall be set by the host according to the recommendation in IEC 62541-9:2020, Annex A. This is the only state variable that shall be set by the EDD to properly represent the state of the device. All other state variables interact with the user of the alarming system of the host. Informational: The server fires events on state transitions. From an EDD perspective, this is the most important property that shall be managed by the EDD to provide proper alarm states.	EDD
	InputNode	NULL	Host
	SuppressedState	Part of the alarming system of the host. Shall not be set by the EDD. Any explicit mapping of this variable shall be ignored by the host and/or shall be considered as error during tokenization of the EDD.	Host
	OutofServiceState	Part of the alarming system of the host. Shall not be set by the EDD. Any explicit mapping of this variable shall be ignored by the host and/or shall be considered as error during tokenization of the EDD.	Host
	ShelvingState	Part of the alarming system of the host. Shall not be set by the EDD. Any explicit mapping of this variable shall be ignored by the host and/or shall be considered as error during tokenization of the EDD.	Host
	SuppressedOrShelved	Part of the alarming system of the host. Shall not be set by the EDD. Any explicit mapping of this variable shall be ignored by the host and/or shall be considered as error during tokenization of the EDD. If the host does not support any of the states described above, the value defaults to false. See IEC 62541-9 for further details.	Host

	AcknowledgeableConditionType		
	BrowseName	Value	Set by
	AckedState	Part of the alarming system of the host. Shall not be set by the EDD. Any explicit mapping of this variable shall be ignored by the host and/or shall be considered as error during tokenization of the EDD. If the host does not support acknowledgement, the value defaults to true. See IEC 62541-9 for further details.	Host
	ConfirmedState	Part of the alarming system of the host. Shall not be set by the EDD. Any explicit mapping of this variable shall be ignored by the host and/or shall be considered as error during tokenization of the EDD. If the host does not support confirmation the value defaults to true. See IEC 62541-9 for further details.	Host
	ConditionType		
	BrowseName	Value	Set by
	ConditionClassId	Shall be set to the node id of MaintenanceConditionClassType	Host
	ConditionClassName	Shall be set to the string 'MaintenanceConditionClassType'	Host
	ConditionName	The NE 107 [1] NAMUR status name as string which is associated with the concrete device health alarm.	Host
	BranchId	NULL	Host
	Retain	If the alarm is active retain shall be set to true otherwise to false. The host synchronizes this state according to the alarm state.	Host
	EnabledState	As soon as the alarm object is available in the address space, this state shall be set to true and as soon as the alarm object is going to disappear it shall be set to false. This is important to clients to get informed by the corresponding event notifications that there are new alarm objects they might want to subscribe to or shall unsubscribe from because they do not exist anymore. If the validity of the EDD object the alarm is mapped to is not conditioned or is explicitly set to true, this flag shall be set to true on loading the EDD respectively set to false on unloading the EDD to trigger the appropriate event notfifications. The validity of the EDD object can have conditionals to control the availability of the alarm by the state of the device (e.g. during download to the device). See IEC 62541-9 for further details.	Host/EDD
	Quality	Shall be set to 'Good'	Host
	LastSeverity	Set by the host on change of the current severity. See below under *BaseEventType* how to set the current severity	Host
	Comment	Default is NULL but might be set by the host. Shall not be set by the EDD.	Host
	ClientUserId	Set by host if comment not NULL	Host

	BaseEventType		
	BrowseName	Value	Set by
	EventId	See IEC 62541-5	Host
	EventType	See IEC 62541-5	Host
	SourceNode	NodeId of PADIMType instance	Host
	SourceName	DisplayName of PADIMType instance	Host
	Time	See IEC 62541-5	Host
	ReceiveTime	Same value as 'Time', see above	Host
	Message	Shall be mapped explicitly by the EDD to contain cause and remedy information.	EDD
	Severity	If not mapped by the EDD, the server sets this value to an appropriate level that makes sense in context of the system it is part of. If mapped by the EDD, it shall be set to a value between 1 to 1 000. The server is allowed override the value set by the EDD. See IEC 62541-5 for detailed information. NOTE Changing the severity is triggering an event notification.	Host / EDD

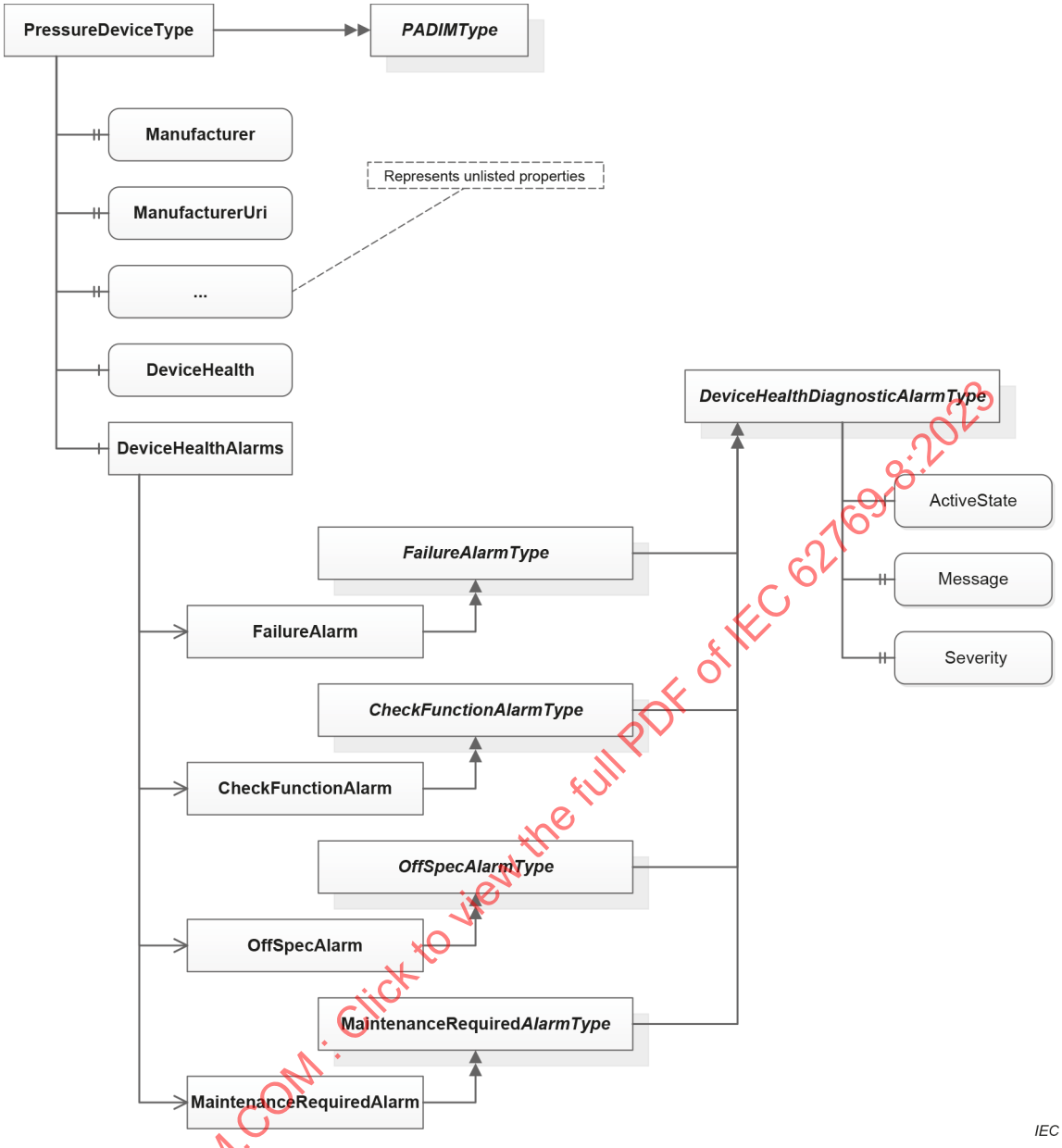


Figure 22 – Supported Alarms

```
SEMANTIC_MAP DeviceTypeMap
{
    "OPC-UA//__PADIM_/PADIMType": PressureDeviceType
}

COLLECTION PressureDeviceType
{
  LABEL "Pressure devicetype";
  MEMBERS
  {
       __DI_Manufacturer,          __resource_block_2[0].PARAM.MANUFAC_ID;
       __DI_ManufacturerUri,       __resource_block_2[0].PARAM.MANUFACTURER_URI;
       __DI_Model,                 __resource_block_2[0].PARAM.DEV_TYPE;
       __DI_SerialNumber,          __resource_block_2[0].PARAM.DEVICE_SN;
       __DI_ProductCode,           __resource_block_2[0].PARAM.DEVICE_PRODUCT_CODE;
       __DI_HardwareRevision,      __resource_block_2[0].PARAM.HARDWARE_REVISION;
       __DI_SoftwareRevision,      __resource_block_2[0].PARAM.DEV_REV;
       __DI_RevisionCounter,       __resource_block_2[0].LOCAL_PARAM.GLOBAL_ST_REV;
       __DI_ProductInstanceUri,    __resource_block_2[0].METHOD.RB_METH_CALCINSTANCE_URI;
       __DI_AssetId,               __resource_block_2[0].PARAM.ASSET_ID;

//Helper methods to fill in information NE107 status
       __DI_DeviceHealth,          __resource_block_2[0].METHOD.RB_METH_CALC_DEVICE_HEALTH;

       //optional but strongly recommended to provide this information
       __DI_DeviceHealthAlarms,    DEVICE_HEALTH_ALARMS;
  }
}

METHOD RB_METH_CALC_DEVICE_HEALTH
{
    LABEL "Calculate NE107";
    TYPE unsigned char;
    DEFINITION
    {
       unsigned char status;
       unsigned char NE107_status;
       DD_STRING strEnum;

   int iFailure, iMaintReq, iFunCheck, iOffSpec;

   NE107_status = 0;
   iFailure = 0;
   iMaintReq = 0;
   iFunCheck = 0;
        iOffSpec = 0;

        ReadCommand(read_DeviceStatus);
        status = DeviceStatus;

        switch (status)
        {
       case e_DeviceStatus_Status_Good:
       NE107_status = 0;
       break;
       case e_DeviceStatus_Status_Failure:
       NE107_status = 1;
       iFailure = 1;
       break;
       case e_DeviceStatus_Status_MaintenanceReq:
       NE107_status = 4;
       iMaintReq = 1;
       break;
       case e_DeviceStatus_Status_FunctionCheck:
       NE107_status = 2;
       iFunCheck = 1;
       break;
       case e_DeviceStatus_Status_OutOfSpec:
       NE107_status = 3;
```

```
        iOffSpec = 1;
        break;
        default:
        NE107_status = 0;
        break;
        }

        ReadCommand(read_ActualAlarmInfo_io);
        iActualAlarmInfo = int_value(ActualAlarmInfo_io);
    // Copy Enum String from ActualAlarmInfo_io into LOC_ActualAlarmInfo
    get_enum_string( ActualAlarmInfo_io, iActualAlarmInfo , strEnum);
    LOC_ActualAlarmInfo = strEnum;

        iassign(DeviceHealthStatus, NE107_status);
        iassign(LOC_FailureVar, iFailure);
        iassign(LOC_MaintReqVar, iMaintReq);
        iassign(LOC_FunCheckVar, iFunCheck);
        iassign(LOC_OffSpecVar, iOffSpec);
        save_values();

        return NE107_status;
        }
}



MENU DEVICE_HEALTH_ALARMS
{
  ITEMS
  {
    FailureAlarmCol,
    CheckFunctionAlarmCol,
    OffSpecAlarmCol,
    MaintenanceRequiredAlarmCol
  }
}


SEMANTIC_MAP FailureAlarmMap
{
    "OPC-UA//__DI_/FailureAlarmType": FailureAlarmCol
}

COLLECTION FailureAlarmCol
{
  LABEL "Failure Alarm";
  MEMBERS
  {
    __UA_ActiveState, LOC_FailureVar;      //True, False - 0 is False, Non Zero is True
    __UA_EnabledState, LOC_AlwaysTrueVar;  //True, False - 0 is False, Non Zero is True
    __UA_Message, LOC_ActualAlarmInfo;
  }
}

SEMANTIC_MAP CheckFunctionAlarmMap
{
    "OPC-UA//__DI_/CheckFunctionAlarmType": CheckFunctionAlarmCol
}

COLLECTION CheckFunctionAlarmCol
{
  LABEL "Check Function Alarm";
  MEMBERS
  {
    __UA_ActiveState, LOC_FunCheckVar;     //True, False - 0 is False, Non Zero is True
    __UA_EnabledState, LOC_AlwaysTrueVar;  //True, False - 0 is False, Non Zero is True
    __UA_Message, LOC_ActualAlarmInfo;
  }
}
```

```
SEMANTIC_MAP OffSpecAlarmMap
{
    "OPC-UA//__DI_/OffSpecAlarmType": OffSpecAlarmCol
}

COLLECTION OffSpecAlarmCol
{
  LABEL "Out of Specification Alarm";
  MEMBERS
  {
    __UA_ActiveState, LOC_OffSpecVar;      //True, False - 0 is False, Non Zero is True
    __UA_EnabledState, LOC_AlwaysTrueVar;  //True, False - 0 is False, Non Zero is True
    __UA_Message, LOC_ActualAlarmInfo;
  }
}

SEMANTIC_MAP MaintenanceRequiredAlarmMap
{
    "OPC-UA//__DI_/MaintenanceRequiredAlarmType": MaintenanceRequiredAlarmCol
}

COLLECTION MaintenanceRequiredAlarmCol
{
  LABEL "Maintenance Required Alarm";
  MEMBERS
  {
    __UA_ActiveState, LOC_MaintReqVar;     //True, False - 0 is False, Non Zero is True
    __UA_EnabledState, LOC_AlwaysTrueVar;  //True, False - 0 is False, Non Zero is True
    __UA_Message, LOC_ActualAlarmInfo;
  }
}
```

Figure 23 – Example of alarm mapping

5.12 Explicit mapping of units

To be able to support units which are not defined in the scope of a bus protocol or which unit enumerations do not comply to the standard definition, units shall be mapped explicitly (see Figure 24).

In case the UNECE namespace is used for the mapping, the display name and the description of the unit shall always be taken from the UNECE definition to be used on the OPC-UA side (e.g. EUInformation, EnumValueType, etc.). The conversion of the UNECE common code to Int32 shall be calculated as recommended in IEC 62541-8 in context of EUInformation. The UNECE common code definitions can be found at 'UN/CEFACT'.

```
SEMANTIC_MAP NonCompliantLenghtUnit_Map
{
    "OPC-UA//__UA_/MultiStateValueDiscreteType": NonCompliantLengthUnitVar
    {
        { 0, "UNIT//UNECE/5066068"},
        { 1, "UNIT//UNECE/5067858"},
        { 2, "UNIT//UNECE/4804168"},
        { 3, "UNIT//UNECE/4607828"}
    }
}

VARIABLE NonCompliantLengthUnitVar
{
    LABEL "Non compliant unit variable";
    CLASS DEVICE;
    TYPE  ENUMERATED
    {
        { 0, "mm"},
        { 1, "m"},
        { 2, "in"},
        { 3, "ft"}
    }
}

UNIT NonCompliantLengthUnit_UR
{
    NonCompliantLengthUnitVar: SMR_HighBlockDistance_2
}

VARIABLE SMR_HighBlockDistance_2
{
      LABEL T_19_BlockingDistance;
      HELP  T_BlockDistance_TT;
      CLASS DEVICE;

      HANDLING IF (HWLock||SILLock||SWLock||WHGLock) {READ;} ELSE {READ & WRITE;}

      TYPE  FLOAT
      {

            MAX_VALUE
                  SELECT (NonCompliantLengthUnitVar)
                  {
                        CASE 0:/*Max*/200000.0;
                        CASE 1:/*Max*/200.0;
                        CASE 2:/*Max*/7874.0157480315;
                        CASE 3:/*Max*/656.167979002625;
                  }
            MIN_VALUE 0.0;
      }
      DEFAULT_VALUE 0;
}
```


Figure 24 – EDD example of explicit unit mapping

5.13 Explicit mapping of aggregated data by methods

In case the data required on the OPC-UA-side does not exist in the required format but might be calculated or aggregated by an EDD method which then returns the data with the required format and type, this method can be mapped directly to the data item and shall always be executed before retrieving the data from the OPC-UA server.

The type of the return value of the method shall be compliant with the datatype of the data item it is mapped to. The data item on the OPC-UA side shall be set to read only (see 6.5).

The following example (see Figure 25 and Figure 26) shows an instance of a PADIMType containing the variable ‘ProductInstanceUri’ which does not exist as plain value in the device but could be calculated by concatenation of ‘ManufacturerUri’, ‘Model’ and ‘SerialNumber’.

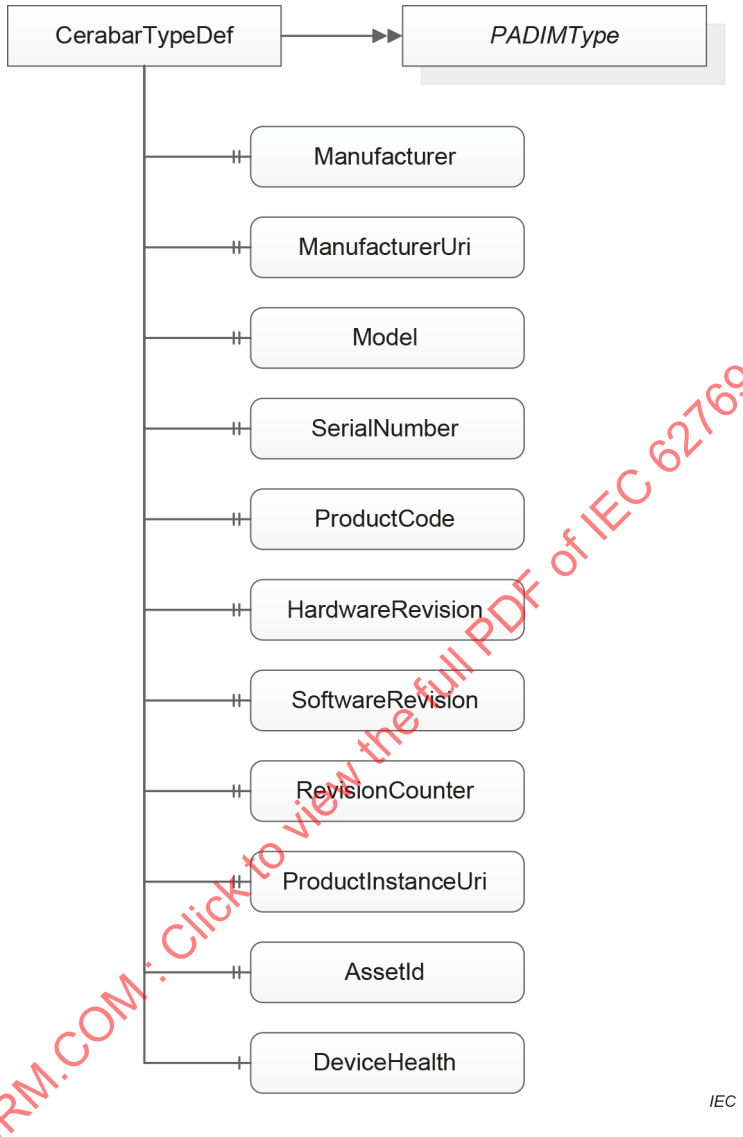


Figure 25 – Instance of PADIMType

```
SEMANTIC_MAP PADIMTypeDefinitionMap
{
    "OPC-UA//__PADIM_/PADIMType", "0112/2///61987#ABA831#003": CerabarTypeDef
}

/* The method 'METH_CalcInstanceUri' is mapped to the data item
'ProductInstanceUri' */
COLLECTION CerabarTypeDef
{
   LABEL "Cerabar S";
   HELP    [blank];
   MEMBERS
   {
            __DI_Manufacturer,         DeviceManufacturer_local;
            __DI_ManufacturerUri,      DeviceManufacturerUrl;
            __DI_Model,                DeviceNameString;
            __DI_SerialNumber,         DeviceSerialNumber;
            __DI_ProductCode,          DeviceOrderCode;
            __DI_HardwareRevision,     PaPbHardwareRevision;
            __DI_SoftwareRevision,     PaPbSoftwareRevision;
            __DI_RevisionCounter,      ConfigurationCounter;
            __DI_AssetId,              PaPbDescriptor;
            __DI_DeviceHealth,         DeviceHealthStatus;
            __DI_ProductInstanceUri,   METH_CalcInstanceUri;
   }
}

/* Return value will be assigned to 'ProductInstanceUri' */
METHOD METH_CalcInstanceUri
{
    LABEL     "Calculate product instance URI";
    TYPE      DD_STRING;
    DEFINITION
    {
            DD_STRING uri;

            uri = DeviceManufacturerUrl + "/" + DeviceNameString + "/" +
DeviceSerialNumber;
            return uri;
    }
}
```

Figure 26 – Method of type DD_STRING mapped to a string variable

5.14 Explicit mapping with reference types

5.14.1 General

In case an instance of an OPC-UA object type needs to be expanded by additional data items not belonging to the object type (e.g. AliasName), the reference type describing the relation to this additional item shall be declared explicitly. This is where the reference type collection comes into play to define aliases (placeholders) for reference types (see 5.4). These aliases shall be used in the EDD collection to define the relation (reference type) to the referenced EDD object.

Because there is no instance declaration for the additional data item, the EDD construct representing it shall be explicitly mapped to an OPC-UA type, with one exception: If the referenced EDD construct is an EDD menu which is not mapped to an OPC-UA type, the reference alias referencing the EDD menu shall be used directly for all EDD objects contained in the menu. The EDD objects of the menu appear as direct children of the parent object of the menu and the menu itself does not appear in the object model (see 7.2).

5.14.2 Example: Adding an additional property to an instance of variable

There is no equivalent instance declaration for something like ‘FillPercentage’ in the type definition of ‘LevelMeasurementVariableType’. In the following example illustrated in Figure 27 and Figure 28, an additional property ‘FillPercentage’ has been added to an instance of a ‘LevelMeasurementVariableType’. Furthermore, a DictionaryEntry, defined in OPC 10000-19, has been linked to the property to specify its semantics.

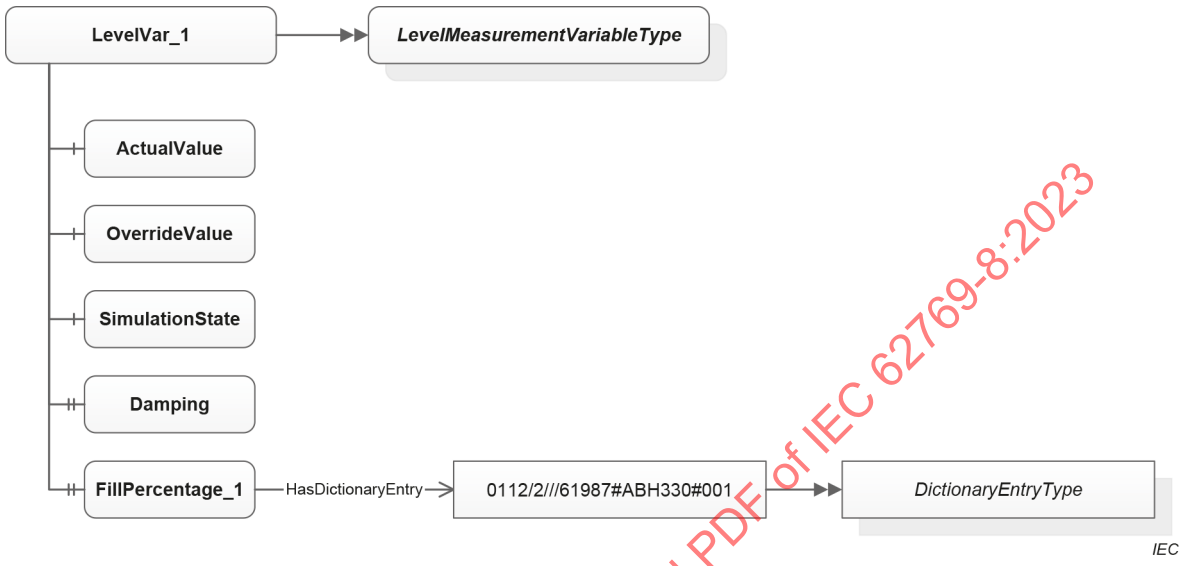


Figure 27 – Adding a property which is not defined in mapped type

```
SEMANTIC_MAP Level1Map
{
    "OPC-UA//__PADIM_/LevelMeasurementVariableType": LevelVar_1
}

COLLECTION LevelVar_1
{
    LABEL "Level value with fill percentage";
    HELP [blank];
    MEMBERS
    {
        __UA_Value,               LevelOut_1;
        __PADIM_Damping,          LevelDamp_1;
        __PADIM_ActualValue,      InternalLevel_1;
        __PADIM_SimulationValue,  LevelSimVal_1;
        __PADIM_SimulationState,  LevelSimOn_1;
        __HasProperty,            FillPercentage_1;
    }
}

SEMANTIC_MAP
{
    "OPC-UA//__UA_/Float", "0112/2///61987#ABH330#001": FillPercentage_1
}

VARIABLE FillPercentage_1
{
    LABEL   "Fill percentage";
    HELP    [blank];
    CLASS   DYNAMIC;
    HANDLING READ;
    TYPE    FLOAT;
}
```

Figure 28 – EDD example of adding a property

To make this example work, it is mandatory to define a reference type collection which contains a definition for '__HasProperty' as alias for the 'HasProperty' reference type (see 5.4). Mapping the datatype 'Float' to the variable is shown just for completeness but is not really needed because the implicit typecast would do the same in this case (see 6.13).

5.14.3 Example: Adding an additional variable to an instance of a variable or an object

There is no equivalent instance declaration for something like 'MeasuredTemperature3' in the type definition of 'PressureMeasurementVariableType'. In the following example illustrated in Figure 29 and Figure 30, an additional component 'MeasuredTemperature3' has been added to an instance of a 'PressureMeasurementVariableType'. Furthermore, a DictionaryEntry, defined in OPC 10000-19, has been linked to the component to specify its semantic.

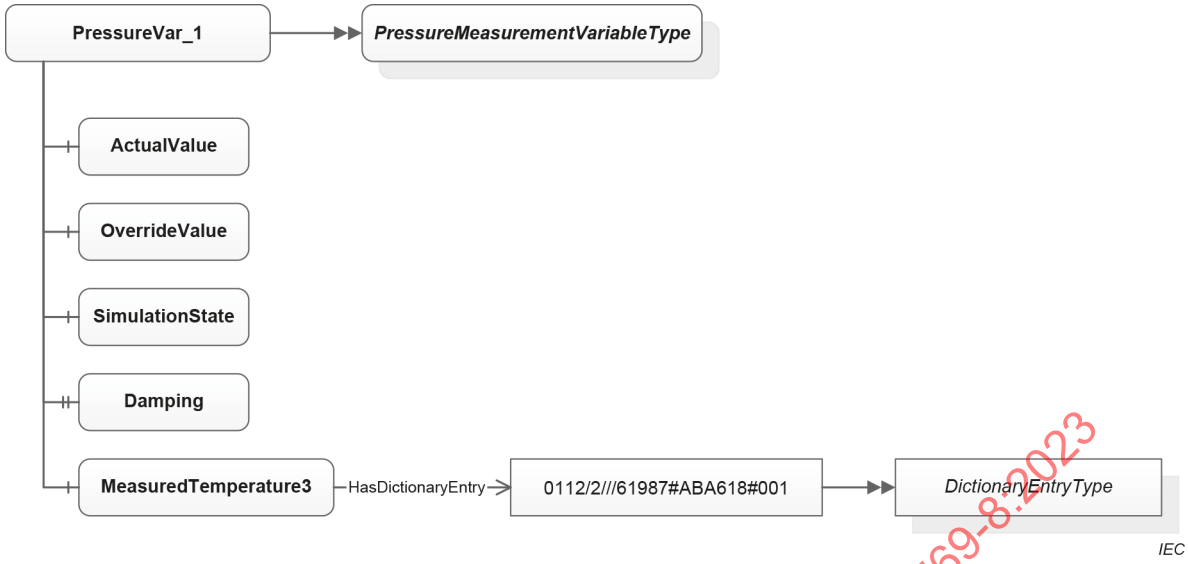


Figure 29 – Adding a component

```
SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/PressureMeasurementVariableType": PressureVar_1
}

COLLECTION PressureVar_1
{
    LABEL "Pressure value with alias";
    HELP [blank];
    MEMBERS
    {
        __UA_Value,               PressureOut_1;
        __PADIM_Damping,          PressureDamp_1;
        __PADIM_ActualValue,      InternalPress_1;
        __PADIM_SimulationValue,  PressureSimVal_1;
        __PADIM_SimulationState,  PressuresSimOn_1;
        __HasComponent,           MeasuredTemperature3;
    }
}

SEMANTIC_MAP PressureTemp_mp
{
    "OPC-UA//__UA_/AnalogUnitType",\
    "OPC-UA//__UA_/Float",\
    "0112/2///61987#ABA618#001"\
  : MeasuredTemperature3
}
VARIABLE MeasuredTemperature3
{
   LABEL UpperCase_MeasuredTemperature3_LABEL_TEXT
   HELP [blank];
   CLASS INPUT & DYNAMIC;
   HANDLING  READ;
   VALIDITY TRUE;
   TYPE FLOAT
   {
       EDIT_FORMAT ".1f";
       DISPLAY_FORMAT ".1f";
       DEFAULT_VALUE 0.0;
   }
}
UNIT TemperatureUnit_relation
{
   TemperatureUnit: MeasuredTemperature3
}
```

Figure 30 – EDD example adding a component

To make this example work, it is mandatory to define a reference type collection which contains a definition for '__HasComponent' as alias for the 'HasComponent' reference type (see 5.4).

Because a 'HasComponent' references an instance of an object or variable type and not just a property with a datatype, the object or variable type shall be assigned explicitly – in this case an 'AnalogUnitType' from the UA namespace has been mapped.

Mapping the datatype 'Float' to the variable is shown just for completeness but is not really needed because the implicit typecast would do the same in this case (see 6.13).

5.14.4 Example: Adding an OPC-UA alias to a variable

In the following example illustrated in Figure 31 and Figure 32, an alias 'ProcessValue' has been added to the variable 'PressureVar_1'.

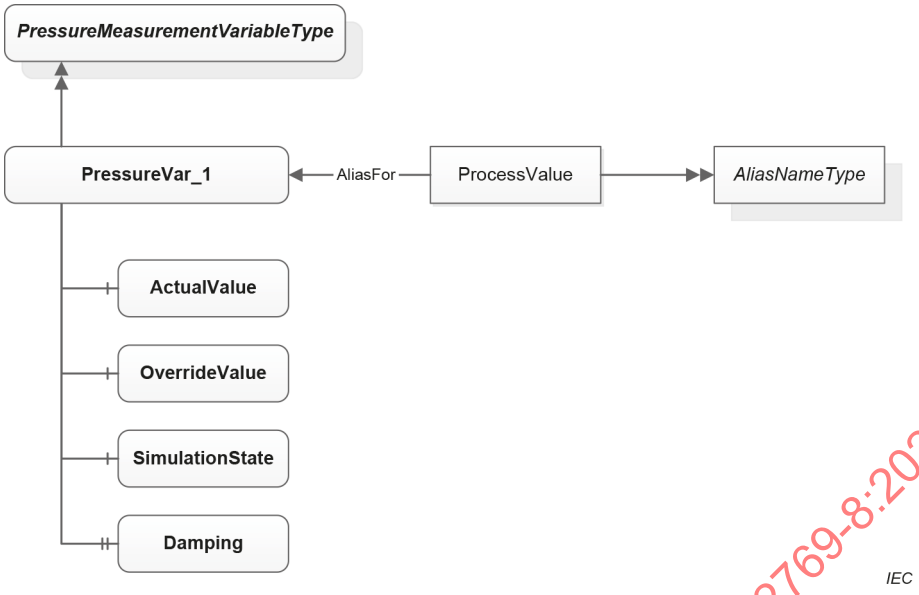


Figure 31 – Adding an alias

```
COLLECTION PressureVar_1
{
    LABEL "Pressure value with alias";
    HELP [blank];
    MEMBERS
    {
        __UA_Value,                PressureOut_1;
        __PADIM_Damping,           PressureDamp_1;
        __PADIM_ActualValue,       InternalPress_1;
        __PADIM_SimulationValue,   PressureSimVal_1;
        __PADIM_SimulationState,   PressuresSimOn_1;
        __HasAlias,                ProcessValueTag_COL;
    }
}

SEMANTIC_MAP ProcessTags
{
    "OPC-UA//__UA_/AliasNameType/": ProcessValueTag_COL
}

COLLECTION ProcessValueTag_COL
{
    LABEL "Process value tag";
    HELP [blank];
    MEMBERS
    {
        __UA_BrowseName,   LOC_ProcessValueTag;
    }
}

VARIABLE LOC_ProcessValueTag
{
    LABEL    "Process Value Tag";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "ProcessValue";
}
```

Figure 32 – EDD example adding an alias

To make this example work, it is mandatory to define a reference type collection which contains a definition for '__HasAlias' as alias for the 'HasAlias' reference type.

Furthermore, the implicit rule for mapping the BrowseName has been overruled to assign an appropriate name to the alias (see 6.1).

6 Implicit rules

6.1 BrowseName of OPC-UA object

6.1.1 General

If the BrowseName is not given by the instance declaration referencing a variable or object, the identifier of the EDD construct shall be used as BrowseName for the instance of the mapped OPC-UA object or variable type. This is the case for container like object types like 'FolderType', 'SignalSetType', etc..

6.1.2 Overruling of BrowseName implicit rule

The BrowseName implicit rule can be overruled by explicitly mapping an adequate EDD variable to the BrowseName attribute. The value of the EDD variable will be mapped to the BrowseName attribute.

6.2 DisplayName of OPC-UA object

6.2.1 General

If the OPC-UA object is linked to a DictionaryEntry, the DisplayName shall contain the content of the corresponding property from the dictionary entry (e.g., 'preferred name' in the case of IEC Common Data Dictionary (CDD) [2]).

If the EDD construct has a label, the DisplayName shall contain the content of the label.

If the EDD construct does not have a label, the DisplayName shall contain the name of the type where the postfix 'Type' is replaced by a sequence number starting with 1. Ordering is based on the alphanumerical sorting of the EDD item name.

6.2.2 Overruling of DisplayName implicit rule

The DisplayName implicit rule can be overruled by explicitly mapping an adequate EDD variable to the DisplayName attribute. The value of the EDD variable will be mapped to the DisplayName attribute.

6.3 HANDLING and AccessLevel

The state of the HANDLING attribute of an EDD construct is reflected by Bit0 and Bit1 of AccessLevel of an OPC-UA variable.

READ → Bit0
WRITE Bit1

6.4 VALIDITY and availability

The state of the VALIDITY attribute of an EDD construct is reflected by the availability of an OPC-UA object or variable in the address space.

FALSE → The object or variable is not visible and not accessible.
TRUE → The object or variable is visible and accessible according to its AccessLevel.

6.5 Return values of EDD methods

6.5.1 EDD methods mapped to OPC-UA objects, variables, properties or attributes

The method shall be executed before the value is read. Based on the implicit typecasts the return value of the method shall be transferred to the mapped data item. The data item will be set to read only.

6.5.2 EDD methods mapped to OPC-UA methods

If the type of the EDD method is unsigned integer, the return value shall be interpreted as operational result code which has been set by the EDD method and shall be returned to the OPC-UA client as status code of the CallMethodResult structure.

For any other type definition – including no type definition – the operational service result shall always be ‘Good’ (0x00000000).

6.6 Units

For an EDD unit variable which does not have a SEMANTIC_MAP defining a unit enumeration with semantic ids, the protocol specific units shall be mapped to UNECE units according to the FDI® protocol profile and as recommended in ‘IEC 62541-8’ context of EUInformation.

The value, the display name and the description of the unit shall always be taken from the UNECE definition wherever used on the OPC-UA side (e.g. EUInformation, EnumValueType, etc.).

6.7 Range

If exactly one pair MIN_VALUE/MAX_VALUE of an EDD variable is defined, this pair shall be implicitly mapped to the range property of the OPC-UA object.

If several pairs of MIN_VALUE/MAX_VALUE of an EDD variable are defined, all pairs shall be ignored as if there was no definition of MIN_VALUE/MAX_VALUE.

If no pair of MIN_VALUE/MAX_VALUE is defined, the range property of the OPC-UA object shall be set to the minimum respectively maximum value of the data type of the EDD variable.

6.8 Forward cast

If the value of an EDD variable cannot be casted to the datatype of the value of the OPC-UA object, the value shall be left empty – this can only happen if the mapping has not been defined correctly.

Refer also to 6.13.

6.9 Backward cast

If it is not possible to unambiguously cast back the datatype of the value of the OPC-UA object to the datatype of the EDD variable, the value of the OPC-UA object shall be read only.

Refer also to 6.13.

6.10 Abstract OPC-UA types

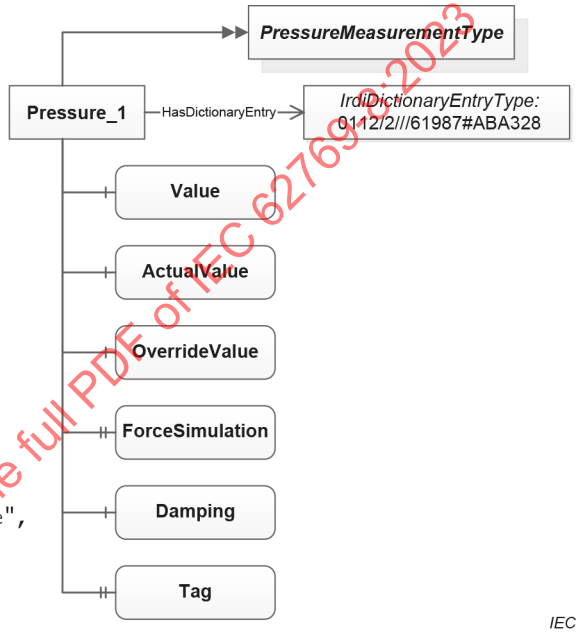
Abstract OPC-UA types shall not be mapped to EDD constructs.

6.11 Unmapped mandatory OPC-UA properties, components and folders

Mandatory components, properties or folders shall be available on the OPC-UA side no matter if they have been mapped on the EDD-side or not. The value of an unmapped component, property or folder shall be the value according to its type definition (e.g. null, empty string, empty folder, etc.).

6.12 Semantic Identifiers and Dictionary References

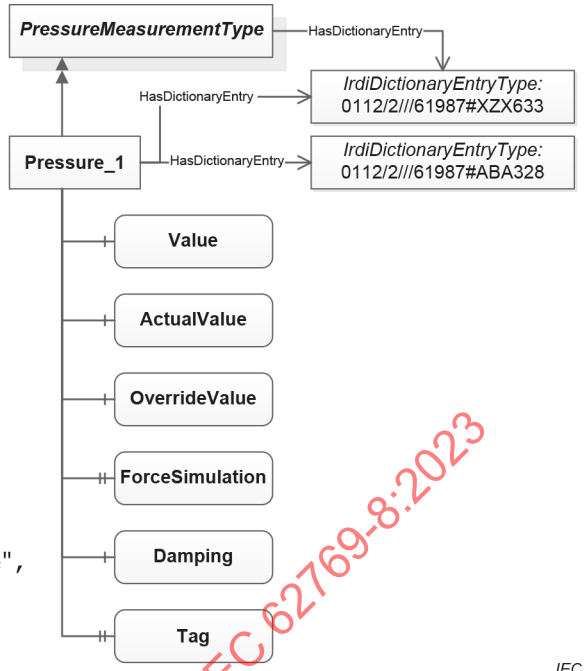
A semantic identifier mapped to an EDD construct which itself is mapped to an OPC-UA object shall be represented as an instance of a *DictionaryEntryType* which is linked to the OPC-UA object by a HasDictionaryEntry ReferenceType (see Figure 33).




```
SEMANTIC_MAP PressureType_MAP
{
   "OPC-UA//PADIM/PressureMeasurementType",
   "0112/2//61987#ABA328"
   : Pressure_1
}
```

Figure 33 – Explicit mapping of dictionary entries

If an OPC-UA object type already has a dictionary entry linked to it by its definition in the scope of its namespace (e.g. PADIM), explicitly mapped dictionary entries will be linked additionally to the appropriate OPC-UA object. An existing dictionary entry defined in the scope of a specific OPC-UA namespace shall not be replaced by an explicitly mapped dictionary entry (see Figure 34).




```
SEMANTIC_MAP PressureType_MAP
{
  "OPC-UA//PADIM/PressureMeasurementType",
  "0112/2//61987#ABA328"
  : Pressure_1
}
```

Figure 34 – Combination with an implicitly mapped dictionary entry

6.13 Implicit Type Casts for OPC-UA DataTypes

Table 2 defines implicit Type Casts between EDDL and OPC-UA data types.

Table 2 – Implicit Type Casts

EDD	OPC-UA	Forward	Backward
Any	BaseDataType	Most appropriate datatype according to IEC 62769-5.	Most appropriate datatype according to IEC 62769-5.
string	string	According to FDI® information model. Refer to IEC 62769-5	According to FDI® information model. Refer to IEC 62769-5
number	string	Number shall be represented as string	Not supported, OPC-UA side shall be read-only
integer	float	Integer shall be represented as float	Not supported, OPC-UA side shall be read-only
Enumerated	Unsigned integer	According to FDI® information model. Refer to IEC 62769-5	According to FDI® information model. Refer to IEC 62769-5
Enumerated	String	The description of the current enum value shall be displayed	Not supported, OPC-UA side shall be read-only
Enumerated	MultiStateValue-DiscreteType	According to FDI® information model. Refer to IEC 62769-5	According to FDI® information model. Refer to IEC 62769-5
Enumerated	MultiStateDictionaryEntry-DiscreteType	According to FDI® information model. Refer to IEC 62769-5	According to FDI® information model. Refer to IEC 62769-5

7 Mapping the EDD device model into PA-DIM (OPC 30081)

7.1 General

PA-DIM is an information model for process automation and is defined in IEC 62541-100 with the following namespace URI:

http://opcfoundation.org/UA/PADIM/

The mapping rules contained in the following subclauses are solely valid for OPC-UA objects belonging to the namespace of PA-DIM.

7.2 Explicit mapping of sub-devices

Devices having sub-devices are modeled using the SubDevices component of PADIMType (see Figure 35). The SubDevices component is an instance of ConfigurableObjectType which is usually configured by the client by adding instances of the types referenced by the folder SupportedTypes of the SubDevices component. In our case, it does not make sense to let the client add instances to the SubDevices component because the sub-devices shall be defined by the business logic of the EDD. According to the pattern for organizing sub-devices defined in IEC 62541-100, an empty folder 'SupportedTypes' indicates that all sub-devices that could be added to the SubDevices component were already added.

Therefore, the folder SupportedTypes is always empty because in our case the EDD is controlling which sub-devices are available and there are no more instances left for the client to add. Or in other words, the instances that could be added were already added by the EDD. Therefore, the folder SupportedTypes shall not be mapped to provide an empty folder on the OPC-UA side (see 6.11).

NOTE Currently the mapping of *ConfigurableObjectType* is only defined in context of sub-devices for devices of type PADIMType.

7.3 Adding sub-devices

Sub-devices are added by explicit mapping with reference types. Therefore, a reference alias collection with an alias for the reference type hasComponent shall be defined which will be used to add these instances to the component SubDevices. In the example shown in Figure 36, an untyped EDD menu is referenced by the hasComponent alias for not having to define several hasComponent aliases to add several components. See 5.14 for further details.

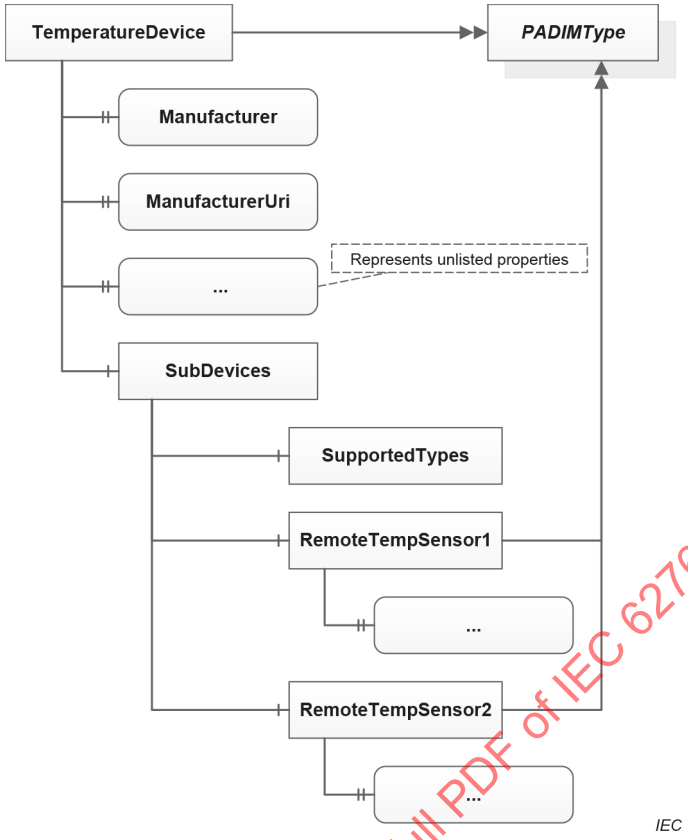


Figure 35 – Subdevices

```
SEMANTIC_MAP DeviceTypeMap
{
    "OPC-UA//__PADIM_/PADIMType": TemperatureDevice, RemoteTempSensor1,
RemoteTempSensor2
}

COLLECTION TemperatureDevice
{
    LABEL "fancy temperature device";
    MEMBERS
    {
       __DI_Manufacturer,          DeviceManufacturer_local;
       __DI_ManufacturerUri,       DeviceManufacturerUrl;
       __DI_Model,                 DeviceNameString;
       __DI_SerialNumber,          DeviceSerialNumber;
       __DI_ProductCode,           DeviceOrderCode;
       __DI_HardwareRevision,      PaPbHardwareRevision;
       __DI_SoftwareRevision,      PaPbSoftwareRevision;
       __DI_RevisionCounter,       ConfigurationCounter;
       __DI_ProductInstanceUri,    METH_CalcInstanceUri;
       __DI_AssetId,               PaPbDescriptor;
       __DI_DeviceHealth,          METH_CalcDeviceHealthStatus;
       __DI_DeviceHealthAlarms,    MENU_DeviceHealthAlarms;
       __PADIM_SignalSet,          MENU_Signals;
       __PADIM_SubDevices,         SubDevColl;
    }
}

/*****************************************************************************/
/*                                                                           */
/* There is no mapping for the 'SupportedTypes' folder by intention.         */
/* Because this folder is mandatory on the OPC-UA side the OPC-UA server is  */
/* forced to provide an empty folder for it which is what we actually need.  */
/*                                                                           */
/*****************************************************************************/
COLLECTION SubDevColl
{
    LABEL "Configurable object for sub devices";
    MEMBERS
    {
       /* __HasComponent is defined in reference type alias collection */
       __HasComponent,             SubDeviceMenu;
    }
}


/*****************************************************************************/
/*                                                                           */
/* Non-typed menu serves as invisible container for the components.          */
/* The menu items appear as direct children of the 'SubDevices' component in */
/* the OPC-UA address space                                                  */
/*                                                                           */
/*****************************************************************************/
```

```
MENU SubDeviceMenu
{
    LABEL "I'm hiding";
    ITEMS
    {
        RemoteTempSensor1,
        RemoteTempSensor2
    }
}

COLLECTION RemoteTempSensor1
{
    LABEL "Sub device 1";
    VALIDITY IF (PluggedOne > 0) {TRUE;} ELSE { FALSE;}
    MEMBERS
    {
       __DI_Manufacturer,          DeviceManufacturer_local;
       __DI_ManufacturerUri,       DeviceManufacturerUrl;
       ...                  ,      ...                  ;
    }
}

COLLECTION RemoteTempSensor2
{
    LABEL "Sub device 2";
    VALIDITY IF (PluggedTwo > 0) {TRUE;} ELSE { FALSE;}
    MEMBERS
    {
       __DI_Manufacturer,          DeviceManufacturer_local;
       __DI_ManufacturerUri,       DeviceManufacturerUrl;
       ...                  ,      ...                  ;
    }
}
```

Figure 36 – Subdevices example

Bibliography

[1] NAMUR NE 107, *Self-Monitoring and Diagnosis of Field Devices*

[2] IEC CDD, available at https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/ [viewed 2023-02-07]

SOMMAIRE

AVANT-PROPOS 57
1 Domaine d'application 59
2 Références normatives 59
3 Termes, définitions, abréviations, acronymes et conventions 60
3.1 Termes et définitions 60
3.2 Abréviations et acronymes 60
3.3 Conventions 60
3.3.1 Utilisation de majuscules 60
3.3.2 Notation graphique 60
4 Vue d'ensemble 62
5 Principes de base du mapping explicite 64
5.1 Mappings sémantiques pour marquer des constructions EDD 64
5.2 Collections d'alias 64
5.2.1 Généralités 64
5.2.2 Syntaxe d'un identificateur sémantique pour un mapping d'alias 65
5.3 Collection d'Alias d'espaces de noms 65
5.4 Collection d'Alias de Types de références 66
5.5 Mappings sémantiques pour un mapping de type OPC-UA 68
5.5.1 Généralités 68
5.5.2 Syntaxe d'un identificateur sémantique pour OPC-UA 68
5.6 Mappings sémantiques pour le mapping d'unités 69
5.6.1 Généralités 69
5.6.2 Syntaxe d'un identificateur sémantique pour les Unités 69
5.7 Mapping explicite de types de variables OPC-UA 69
5.8 Mapping explicite de types OPC-UA complexes 71
5.9 Mapping explicite d'un objet imbriqué et de types de variables 73
5.9.1 Généralités 73
5.9.2 Utilisation de collections 73
5.9.3 Utilisation de menus 73
5.9.4 Diagramme OPC-UA d'un exemple de mapping imbriqué 73
5.9.5 Extrait EDD d'un exemple de mapping imbriqué 75
5.10 Mapping explicite de méthodes 78
5.10.1 Mapping de méthodes EDD avec des objets, des variables ou des propriétés OPC-UA 78
5.10.2 Mapping de méthodes EDD avec des méthodes OPC-UA 78
5.11 Mapping explicite d'alarmes 80
5.12 Mapping explicite d'unités 88
5.13 Mapping explicite de données agrégées au moyen de méthodes 89
5.14 Mapping explicite avec des types de références 91
5.14.1 Généralités 91
5.14.2 Exemple: Ajout d'une propriété supplémentaire à une instance de variable 92
5.14.3 Exemple: Ajout d'une variable supplémentaire à une instance d'une variable ou d'un objet 93
5.14.4 Exemple: Ajout d'un alias OPC-UA à une variable 95
6 Règles implicites 97
6.1 Nom d'exploration d'un objet OPC-UA 97

6.1.1 Généralités 97
6.1.2 Annulation de la règle implicite BrowseName 97
6.2 Nom d'affichage d'un objet OPC-UA 97
6.2.1 Généralités 97
6.2.2 Annulation de la règle implicite DisplayName 97
6.3 HANDLING et AccessLevel 97
6.4 VALIDITY et disponibilité 97
6.5 Valeurs de retour des méthodes EDD 98
6.5.1 Méthodes EDD mappées avec des objets, des variables, des propriétés ou des attributs OPC-UA 98
6.5.2 Méthodes EDD mappées avec des méthodes OPC-UA 98
6.6 Unités 98
6.7 Plage 98
6.8 Transtypage ascendant 98
6.9 Transtypage descendant 98
6.10 Types OPC-UA abstraits 99
6.11 Propriétés, composants et dossiers OPC-UA obligatoires non mappés 99
6.12 Identificateurs sémantiques et Références de Dictionnaire 99
6.13 Transtypages implicites pour les types de données OPC-UA 100
7 Mapping du modèle de l'appareil EDD avec le modèle PA-DIM (OPC 30081) 101
7.1 Généralités 101
7.2 Mapping explicite de sous-appareils 101
7.3 Ajout de sous-appareils 101
Bibliographie 105

Figure 1 – Notation graphique de l'OPC UA pour les NodeClasses 61
Figure 2 – Notation graphique de l'OPC UA pour les Références 61
Figure 3 – Exemple de notation graphique de l'OPC UA 62
Figure 4 – Référence de Type optimisée 62
Figure 5 – Similarité entre les objets OPC-UA et les collections EDD 63
Figure 6 – Syntaxe des identificateurs sémantiques utilisés pour les points d'entrée EDD 64
Figure 7 – Syntaxe d'un identificateur sémantique pour un mapping d'alias 65
Figure 8 – Collection d'espaces de noms 66
Figure 9 – Collection de Types de références 67
Figure 10 – Utilisation de SEMANTIC_MAP pour les types OPC-UA 68
Figure 11 – Syntaxe d'un identificateur de type OPC-UA 68
Figure 12 – Exemple de mapping sémantique 68
Figure 13 – Syntaxe d'un Identificateur d'Unité 69
Figure 14 – Exemple de mapping le plus simple 69
Figure 15 – Variable EDD mappée avec un type BaseDataVariableType OPC-UA 70
Figure 16 – Exemple de mapping simple avec plage et unité 71
Figure 17 – Mapping d'une collection avec une variable OPC-UA 72
Figure 18 – Objets et variables imbriqués 74
Figure 19 – Exemples d'objets et de variables imbriqués 78
Figure 20 – Méthode simple 79

Figure 21 – Exemple de mapping de méthode simple 80
Figure 22 – Alarmes prises en charge 85
Figure 23 – Exemple de mapping d'alarme 88
Figure 24 – Exemple EDD de mapping explicite d'unités 89
Figure 25 – Instance de PADIMType 90
Figure 26 – Méthode de type DD_STRING mappée avec une variable de chaîne 91
Figure 27 – Ajout d'une propriété qui n'est pas définie dans un type mappé 92
Figure 28 – Exemple EDD d'ajout d'une propriété 93
Figure 29 – Ajout d'un composant 94
Figure 30 – Exemple EDD d'ajout d'un composant 95
Figure 31 – Ajout d'un alias 96
Figure 32 – Exemple EDD d'ajout d'un alias 96
Figure 33 – Mapping explicite d'entrées de dictionnaire 99
Figure 34 – Association à une entrée de dictionnaire explicitement mappée 100
Figure 35 – Sous-appareils 102
Figure 36 – Exemples de sous-appareils 104

Tableau 1 – Mapping des propriétés d'alarmes 82
Tableau 2 – Transtypages implicites 100

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

INTÉGRATION DES APPAREILS DE TERRAIN (FDI®) –

Partie 8: Mapping de l'EDD avec l'OPC-UA

AVANT-PROPOS

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

L'IEC 62769-8 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels. Il s'agit d'une Norme internationale.

Le texte de cette Norme internationale est issu des documents suivants:

Projet	Rapport de vote
65E/851/CDV	65E/909/RVC

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à son approbation.

La langue employée pour l'élaboration de cette Norme internationale est l'anglais.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2, il a été développé selon les Directives ISO/IEC, Partie 1 et les Directives ISO/IEC, Supplément IEC, disponibles sous www.iec.ch/members_experts/refdocs. Les principaux types de documents développés par l'IEC sont décrits plus en détail sous www.iec.ch/publications.

Une liste de toutes les parties de la série IEC 62769, publiées sous le titre général *Intégration des appareils de terrain (FDI®)*, se trouve sur le site web de l'IEC.

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

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

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

INTÉGRATION DES APPAREILS DE TERRAIN (FDI®) –

Partie 8: Mapping de l'EDD avec l'OPC-UA

1 Domaine d'application

La présente partie de l'IEC 62769 spécifie comment la vue interne d'un modèle d'appareil représentée par l'EDD peut être transférée dans une vue externe sous forme de modèle d'information OPC-UA en mappant les constructions EDD avec des objets OPC-UA.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

NOTE Pour les références non datées, l'édition du document de référence (y compris les éventuels amendements) qui s'applique pour une version de technologie FDI®[1] spécifique est définie dans le document FDI® Technology Management et sur les portails de support de FieldComm Group et de PI International.

IEC 61804-3, *Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produit (EDDL) – Partie 3: Sémantique et syntaxe EDDL*

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

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

IEC 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'information*

IEC 62541-8, *Architecture unifiée OPC – Partie 8: Accès aux données*

IEC 62541-9:2020, *Architecture unifiée OPC – Partie 9: Alarmes et Conditions*

OPC 10000-17, *OPC Unified Architecture – Part 17: Alias Names* (disponible en anglais seulement)

OPC 10000-19, *OPC Unified Architecture – Part 19: Dictionary Reference* (disponible en anglais seulement)

IEC 62541-100, *Architecture unifiée OPC – Partie 100: Interface d'appareils*

IEC 62769-1, *Intégration des appareils de terrain (FDI®) – Partie 1: Vue d'ensemble*

IEC 62769-5, *Intégration des appareils de terrain (FDI®) – Partie 5: Modèle d'Information FDI®*

---

[1] FDI® est une marque déposée de l'organisation à but non lucratif Fieldbus Foundation, Inc. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

IEC 62769-6, *Intégration des appareils de terrain (FDI®) – Partie 6: Mappings de technologies FDI®*

ISO/IEC 11179-6, *Technologies de l'information – Registres de métadonnées (RM) – Partie 6: Enregistrement des données*

OPC 30081, *Process Automation Devices – PADIM* (disponible en anglais seulement)

UN/CEFACT, Recommandation 20 de la CEE-ONU, Codes des unités de mesure utilisées dans le commerce international
disponible à l'adresse https://www.unece.org/cefact/codesfortrade/codes_index.html [consulté le 2023-02-07]

3 Termes, définitions, abréviations, acronymes et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions de l'IEC 62769-1, l'IEC 62769-5, l'IEC 62769-6, l'IEC 62541-3, l'IEC 62541-4, l'IEC 62541-5, l'IEC 62541-8, l'IEC 62541-9, l'OPC 10000-17, IEC 62541-100 et de l'OPC 30081 s'appliquent.

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

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

3.2 Abréviations et acronymes

Pour les besoins du présent document, les abréviations et acronymes de l'IEC 62769-1 ainsi que les suivants s'appliquent.

PA-DIM (Process Automation Device Information Model)	Modèle d'information pour les appareils d'automatisation de processus

3.3 Conventions

3.3.1 Utilisation de majuscules

La mise en majuscules de la première lettre des mots est utilisée dans la série IEC 62769 pour souligner un terme défini spécifique à la FDI®.

3.3.2 Notation graphique

L'OPC UA définit une notation graphique pour un AdressSpace de l'OPC UA. Elle définit des symboles graphiques pour l'ensemble des NodeClasses et concernant la façon dont les différents types de Références entre les Nodes peuvent être visualisés. La Figure 1 représente les symboles pour les NodeClasses utilisés dans le présent document. Les types qui représentent des NodeClasses comportent toujours une ombre.

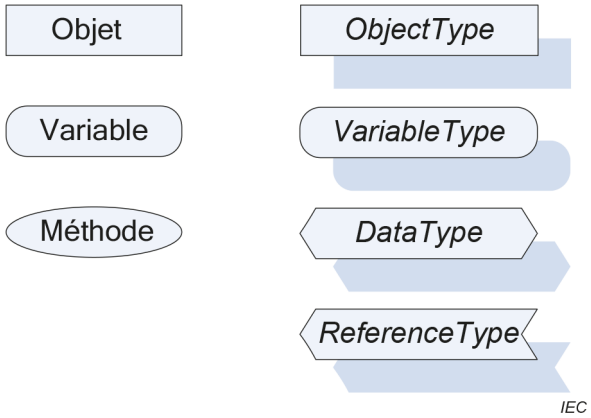


Figure 1 – Notation graphique de l'OPC UA pour les NodeClasses

La Figure 2 représente les symboles pour les ReferenceTypes utilisés dans le présent document. Le symbole de Référence pointe généralement du Node d'origine vers le Node cible. La seule exception est la Référence HasSubType. Les Références les plus importantes, telles que HasComponent, HasProperty, HasTypeDefinition et HasSubType comportent des symboles spécifiques qui permettent de ne pas indiquer le nom de la Référence. Pour les autres ReferenceTypes ou ReferenceTypes dérivés, le nom du ReferenceType est utilisé avec le symbole.

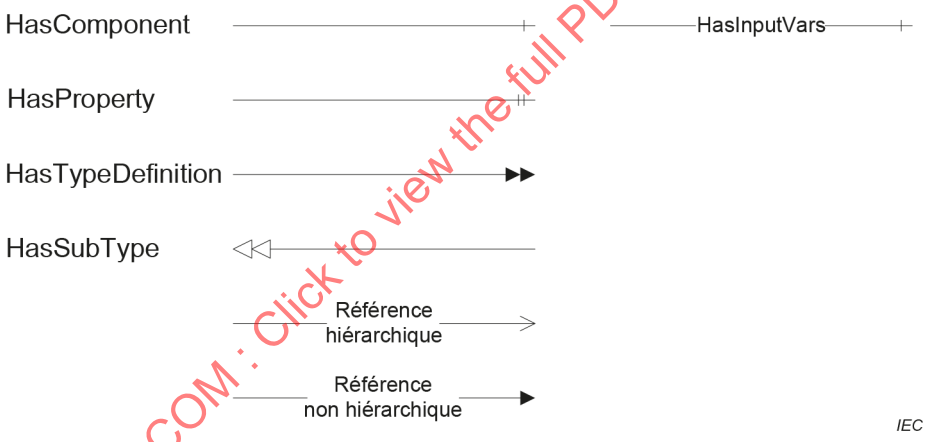


Figure 2 – Notation graphique de l'OPC UA pour les Références

La Figure 3 représente un exemple type de l'utilisation de la notation graphique. Object_A et Object_B sont des instances d'ObjectType_Y indiquées par les Références HasTypeDefinition. L'ObjectType_Y est dérivé de l'ObjectType_X indiqué par la Référence HasSubType. L'Object_A comporte les composants Variable_1, Variable_2 et Method_1.

Afin de décrire les composants d'un Object sur l'ObjectType, les mêmes NodeClasses et Références sont utilisées sur l'Object et sur l'ObjectType, comme cela est indiqué dans l'exemple pour l'ObjectType_Y. Les Nodes utilisés pour décrire un ObjectType sont des Nodes de déclaration d'instance.

Afin de fournir des informations plus détaillées pour un Node, un sous-ensemble ou l'ensemble des Attributs et leurs valeurs peuvent être ajoutés à un symbole graphique (voir, par exemple, Variable_1, le composant de l'Object_A à la Figure 3).

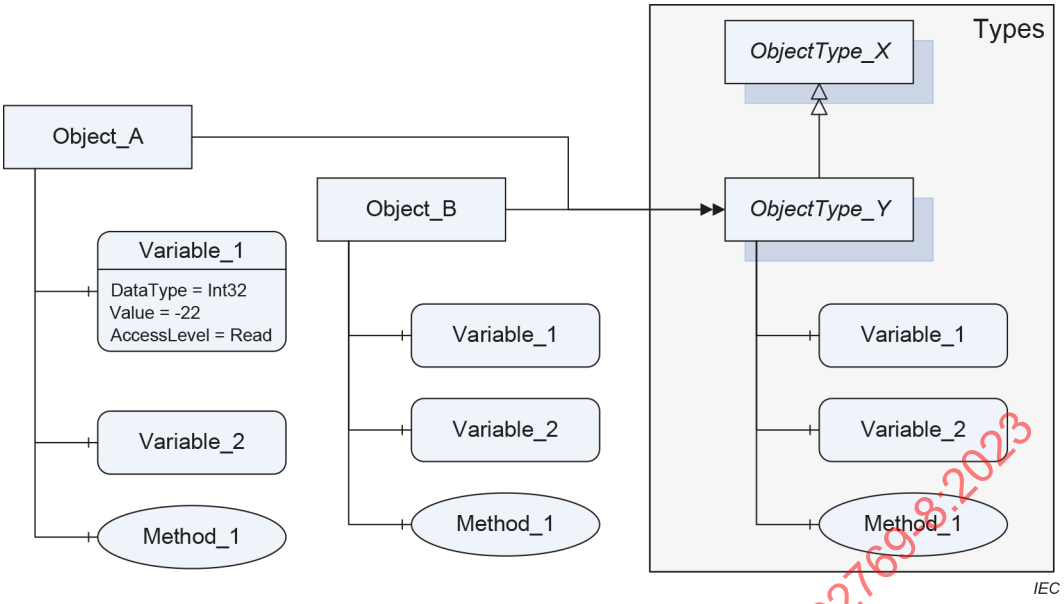


Figure 3 – Exemple de notation graphique de l'OPC UA

Afin d'améliorer la lisibilité, le présent document inclut fréquemment le nom de type à l'intérieur de la boîte d'instance, plutôt que d'afficher les deux boîtes et la référence entre celles-ci. Cette optimisation est représentée à la Figure 4.

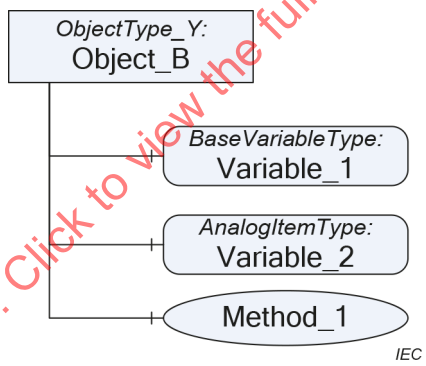


Figure 4 – Référence de Type optimisée

4 Vue d'ensemble

Il existe deux types de mécanismes de mapping, à savoir le mapping explicite et le mapping implicite. Le mapping explicite est assuré par les constructions EDD SEMANTIC_MAP associées à des COLLECTIONS, des MÉTHODES et des VARIABLES, ce qui inclut le mapping des énumérations, et est réalisé par le développeur EDD. Le mapping implicite est assuré par la mise en œuvre du serveur OPC-UA en combinaison avec un Serveur FDI® et englobe les définitions comme les transtypages par défaut (valeurs numériques en float64, par exemple)

voire le mapping de listes complètes de codes d'unités d'un domaine de bus de terrain (HART®[2], PROFIBUS®[3], etc.) avec un domaine OPC-UA (UNECE, CDD, etc.).

L'observation des objets OPC-UA qui contiennent des éléments de données nommés, comme des attributs, des propriétés, des variables et d'autres objets, fait apparaître que les objets EDD les plus semblables sont les collections qui comportent des éléments de données nommés comme des variables, des menus et d'autres collections (voir Figure 5).

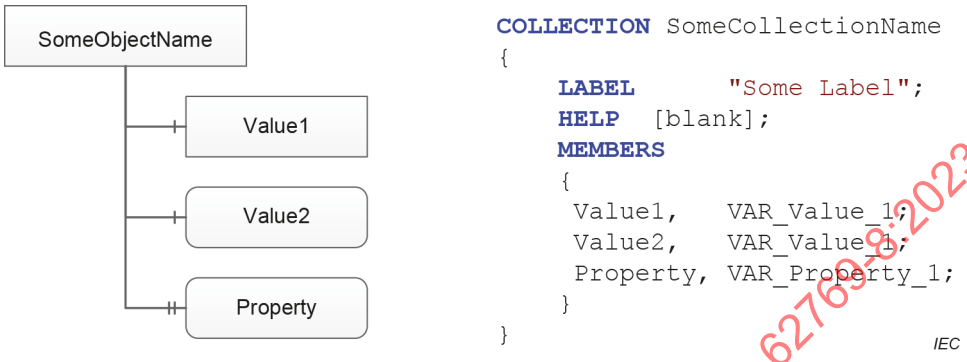


Figure 5 – Similarité entre les objets OPC-UA et les collections EDD

La collection de constructions EDD est en fait l'élément clé qui explique fondamentalement comment les données du modèle d'appareil EDD doivent être mappées avec un modèle d'information OPC-UA. Les articles ci-après décrivent et définissent comment ce mapping doit être réalisé. Par conséquent, le présent document revêt un caractère normatif pour les développeurs EDD et les développeurs de serveurs OPC qui accèdent à un Serveur FDI® en vue de publier un modèle d'information OPC-UA d'après un espace de noms OPC-UA spécifique, ainsi qu'aux informations de mapping fournies par l'EDD.

L'Article 5 explique en détail le principe de fonctionnement du mapping, en particulier pour un modèle d'information OPC-UA qui repose sur un espace de noms.

En s'appuyant sur l'Article 5, l'Article 7 fournit des informations concernant le mapping de l'EDD avec le PA-DIM. Des définitions normatives supplémentaires pour le mapping de données de bus de terrain spécifiques (d'identification, par exemple) avec le PA-DIM sont fournies dans les

---

2 HART® est une marque déposée de l'organisation à but non lucratif FieldComm Group, Inc. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

3 PROFIBUS® est la marque déposée de l'organisation à but non lucratif PROFIBUS Nutzerorganisation e.V. (PNO). Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

spécifications de profil FDI® pour HART, FF, PROFINET®[4], PROFIBUS, Modbus®[5] et ISA100 Wireless®[6].

Avant de procéder à un mapping de l'EDD, il est fortement recommandé de bien comprendre l'OPC-UA en ce qui concerne la définition des types d'objets, des types de variables et des types de références.

5 Principes de base du mapping explicite

5.1 Mappings sémantiques pour marquer des constructions EDD

Afin de ne pas devoir utiliser de conventions de dénomination destinées aux constructions EDD pour lier une finalité précise à une construction EDD, des mappings sémantiques doivent être utilisés en vue de marquer une construction EDD d'un identificateur sémantique spécifiquement défini. A l'heure actuelle, il existe trois définitions de syntaxe pour trois finalités données. Leur utilisation est expliquée en détail dans le contexte correspondant.

La syntaxe des identificateurs sémantiques utilisés pour les points d'entrée EDD est représentée à la Figure 6.

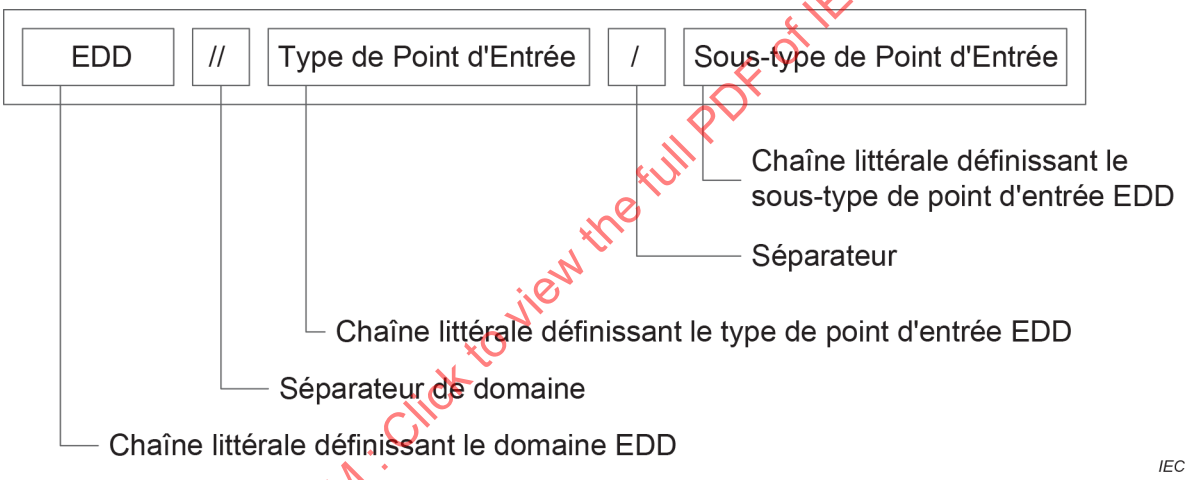


Figure 6 – Syntaxe des identificateurs sémantiques utilisés pour les points d'entrée EDD

5.2 Collections d'alias

5.2.1 Généralités

Des collections d'alias sont utilisées en vue de définir des alias pour les chaînes longues qui ont une signification particulière. Il convient que l'alias soit court, et celui-ci doit être unique dans l'ensemble de l'EDD. Les alias sont utilisés comme identificateurs de membre, comme

---

[4] PROFINET® est la marque déposée de l'organisation à but non lucratif PROFIBUS Nutzerorganisation e.V. (PNO). Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

[5] Modbus® est une marque déposée de Schneider Electric USA, Inc. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

[6] ISA100 Wireless® est la marque déposée de l'organisation à but non lucratif Automation Standards Compliance Institute. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

une partie des identificateurs de membre ou dans des SEMANTIC_MAP pour un mapping de type OPC-UA. Les collections d'alias constituent les principaux points d'entrée pour résoudre un mapping EDD-OPC-UA et ne doivent pas dépendre les unes des autres (voir 5.3 et 5.4).

5.2.2 Syntaxe d'un identificateur sémantique pour un mapping d'alias

La syntaxe d'un identificateur sémantique pour un mapping d'alias est représentée à la Figure 7.

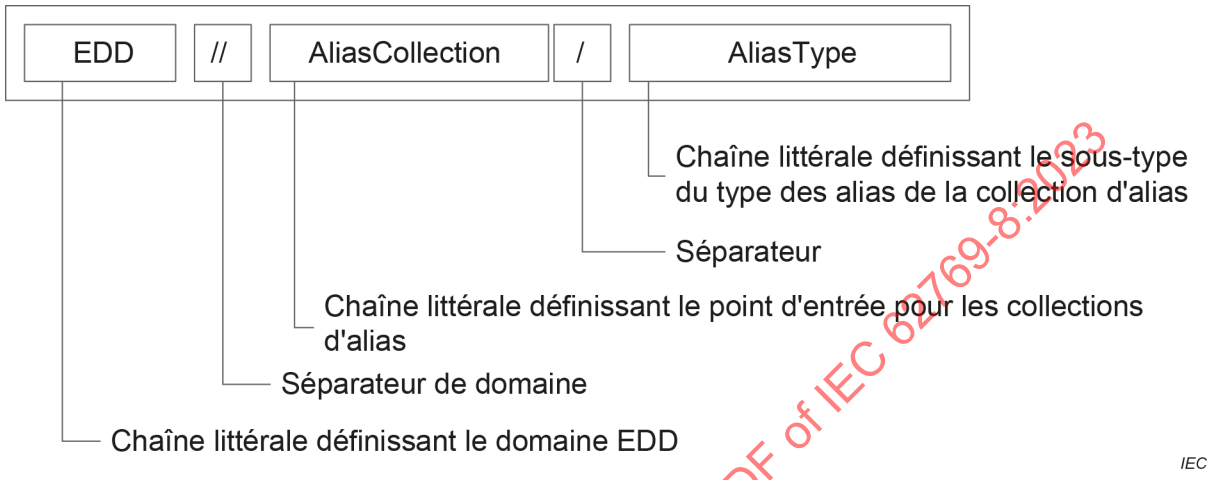


Figure 7 – Syntaxe d'un identificateur sémantique pour un mapping d'alias

5.3 Collection d'Alias d'espaces de noms

Pour éviter les conflits au niveau des noms, tout type d'OPC-UA est défini dans le domaine d'application d'un espace de noms spécifique. Une collection d'espaces de noms et des variables appropriées doivent être définies pour attribuer un nom convenable aux URI d'espaces de noms. L'identificateur de membre de l'élément de collection doit être utilisé comme alias chaque fois qu'un identificateur d'espace de noms est nécessaire. Dans l'exemple suivant, "__UA_" représente l'espace de noms http://opcfoundation.org/UA/, qui est défini par la valeur par défaut de la variable référencée "UA_Namespace".

Il est obligatoire de spécifier au moins une collection d'espaces de noms pour une EDD afin de définir le mapping de l'EDD avec l'OPC-UA.

Définition:
Le nom de la collection d'espaces de noms doit être marqué par un SEMANTIC_MAP avec l'ID sémantique suivant:

"EDD//AliasCollection/Namespaces"

Le nom de la COLLECTION d'espaces de noms, le nom du SEMANTIC_MAP et le nom des variables de chaîne référencées peuvent être choisis librement. La valeur par défaut de la variable de chaîne référencée doit contenir un espace de noms valide.

La Figure 8 représente un exemple de Collection d'espaces de noms EDD.

```
SEMANTIC_MAP Namespace_Map
{
    "EDD//AliasCollection/Namespaces": Namespace_Collection
}

COLLECTION OF VARIABLE Namespace_Collection
{
    LABEL "OPC-UA Namespaces";
    MEMBERS
    {
       __UA_,        UA_Namespace;
       __DI_,        DI_Namespace;
       __PADIM_,     PADIM_Namespace;
       UNECE,        UNECE_Namespace;
    }
}

VARIABLE UA_Namespace
{
    LABEL    "UA namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/";
}

VARIABLE DI_Namespace
{
    LABEL    "DI namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/DI/";
}

VARIABLE PADIM_Namespace
{
    LABEL    "PADIM namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/PADIM/";
}

VARIABLE UNECE_Namespace
{
    LABEL    "UNECE namespace";
    HELP     [blank];
    CLASS    LOCAL;
    HANDLING READ;
    TYPE     ASCII(100);
    DEFAULT_VALUE    "http://www.opcfoundation.org/UA/units/un/cefact/";
}
```


Figure 8 – Collection d'espaces de noms

5.4 Collection d'Alias de Types de références

En ce qui concerne les références OPC-UA pour lesquelles le type de référence spécifique ne peut pas être dérivé sans ambiguïté des deux objets reliés par la référence, le type de référence doit être explicitement défini.

La collection n'est obligatoire que si les mappings décrits en 5.14 sont utilisés.

Le nom de la collection de types de références doit être marqué par un SEMANTIC_MAP avec l'identificateur sémantique suivant:

"EDD//AliasCollection/ReferenceTypes"

Le nom de la COLLECTION de types de références, le nom du SEMANTIC_MAP et le nom des variables de chaîne référencées peuvent être choisis librement. La valeur par défaut de la variable de chaîne référencée doit contenir un identificateur de nœud valide d'un type de référence.

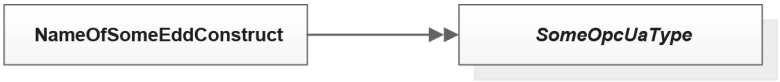
```
SEMANTIC_MAP ReferenceType_Map

{
    "EDD//AliasCollection//ReferenceTypes": ReferenceType_Collection
}

COLLECTION OF VARIABLE ReferenceType_Collection
{
     LABEL "OPC-UA reference types"
     MEMBERS
     {
         __HasAlias,       HasAlias_Definition;
         __HasComponent,   HasComponent_Def;
         __HasProperty,    HasProperty_dfn;
     }
}

VARIABLE HasAlias_Definition
{
    LABEL     "HasAlias reference type";
    HELP      [blank];
    CLASS     LOCAL;
    HANDLING READ;
    TYPE      ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/HasAlias";
}

VARIABLE HasProperty_dfn
{
    LABEL     "HasProperty reference type";
    HELP      [blank];
    CLASS     LOCAL;
    HANDLING READ;
    TYPE      ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/HasProperty";
}


VARIABLE HasComponent_Def
{
    LABEL     "HasComponent reference type";
    HELP      [blank];
    CLASS     LOCAL;
    HANDLING READ;
    TYPE      ASCII(100);
    DEFAULT_VALUE    "http://opcfoundation.org/UA/HasComponent";
}
```


Figure 9 – Collection de Types de références

5.5 Mappings sémantiques pour un mapping de type OPC-UA

5.5.1 Généralités

Le SEMANTIC_MAP de la construction EDD doit être utilisé pour attribuer un type d'objet ou un type de variable OPC-UA à une construction EDD (des collections dans la plupart des cas) (voir Figure 10). D'après cette attribution, le serveur OPC-UA instancie un objet du type d'objet ou du type de variable spécifié et le remplit des données de la construction EDD.



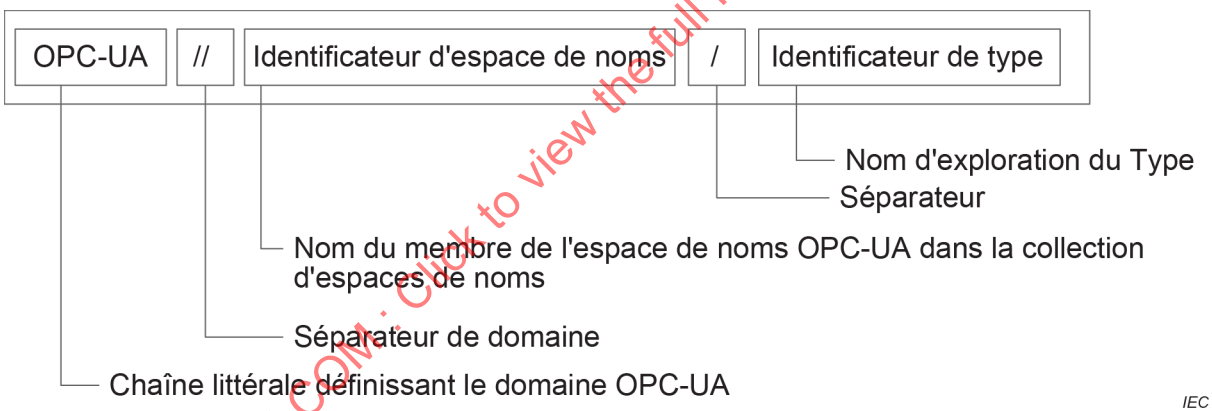

```
SEMANTIC_MAP SomeExplicitTypeMap
{
      "OPC-UA//SomeNamespace/SomeOpcUaType" : NameOfSomeEddConstruct
}
```

IEC

Figure 10 – Utilisation de SEMANTIC_MAP pour les types OPC-UA

5.5.2 Syntaxe d'un identificateur sémantique pour OPC-UA

L'identificateur sémantique qui référence le type d'objet OPC-UA doit être conforme à la syntaxe représentée à la Figure 11.



IEC

Figure 11 – Syntaxe d'un identificateur de type OPC-UA

En ce qui concerne l'exemple du 5.2, un mapping sémantique peut être semblable à l'exemple indiqué à la Figure 12.

```
SEMANTIC_MAP PressureVariableMap

{

   "OPC-UA//__PADIM_/PressureMeasurementVariableType" : PressureVariableColl

}
```

IEC

Figure 12 – Exemple de mapping sémantique

Dans la Figure 12, le type d'objet "PressureMeasurementVariableType" de l'espace de noms "http://opcfoundation.org/UA/PADIM/" est mappé avec une construction EDD qui porte le nom "PressureVariableColl".

5.6 Mappings sémantiques pour le mapping d'unités

5.6.1 Généralités

D'après l'IEC 61804-3, SEMANTIC_MAP peut être utilisé pour mapper des identificateurs sémantiques avec des énumérations d'unités de variables d'unités EDD. Les identificateurs d'unité doivent être utilisés dans l'énumération du type OPC-UA mappé avec la variable d'unité EDD et en tant que codes d'unité dans la structure EUInformation des variables OPC-UA. Cette définition s'applique uniquement aux identificateurs d'unité non conformes à la convention de dénomination de l'ISO/IEC 11179-6.

5.6.2 Syntaxe d'un identificateur sémantique pour les Unités

L'identificateur sémantique qui référence une unité non conforme à l'ISO/IEC 11179-6 doit respecter la syntaxe suivante représentée à la Figure 13.

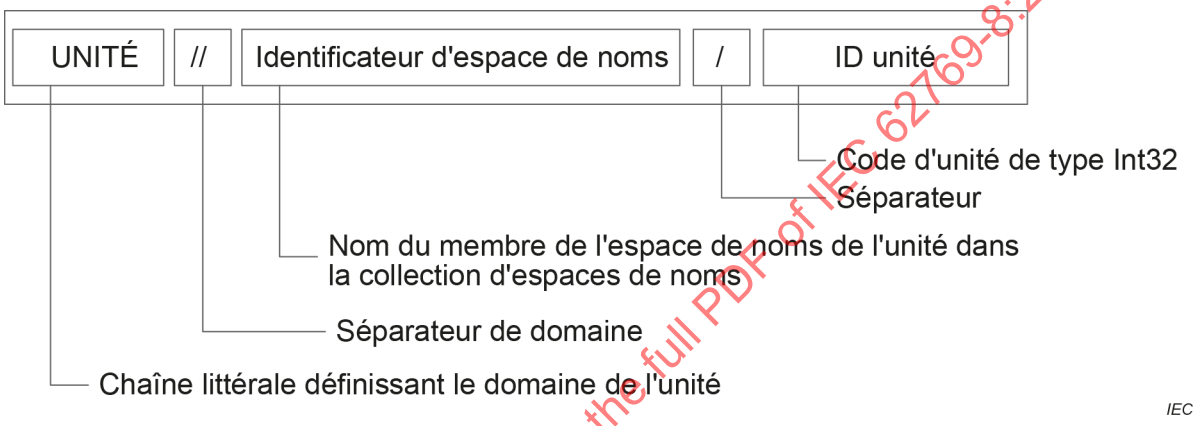


Figure 13 – Syntaxe d'un Identificateur d'Unité

5.7 Mapping explicite de types de variables OPC-UA

Le cas le plus élémentaire qui effectue un mapping d'une variable EDD avec une instance d'un type de variable OPC-UA est représenté à la Figure 14.

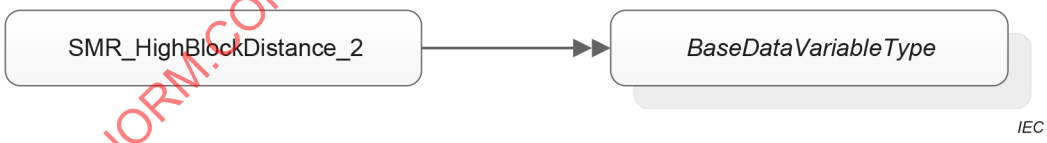


Figure 14 – Exemple de mapping le plus simple

Dans ce cas, la valeur de la variable EDD SMR_HighBlockDistance_2 est mappée avec la valeur d'une variable OPC-UA qui porte le nom SMR_HighBlockDistance_2 en tant qu'instance de BaseDataVariableType (voir Figure 15).

```
SEMANTIC_MAP BlockingDistance_mp
{
    "OPC-UA//__UA_/BaseDataVariableType": SMR_HighBlockDistance_2
}

UNIT SU_Length_UR
{
    SU_Length_1: SMR_HighBlockDistance_2
}

VARIABLE SMR_HighBlockDistance_2
{
       LABEL T_19_BlockingDistance;
       HELP  T_BlockDistance_TT;
       CLASS DEVICE;

       HANDLING IF (HWLock||SILLock||SWLock||WHGLock) {READ;} ELSE {READ & WRITE;}

       TYPE  FLOAT
       {

              MAX_VALUE
                     SELECT (SU_Length_1)
                     {
                            CASE e_Length_milliMeter_LONG:/*Max*/200000.0;
                            CASE e_Length_Meter_LONG:/*Max*/200.0;
                            CASE e_Length_Foot_LONG:/*Max*/656.167979002625;
                            CASE e_Length_Inch_LONG:/*Max*/7874.0157480315;
                     }


              MIN_VALUE
                     SELECT (SU_Length_1)
                     {
                            CASE e_Length_milliMeter_LONG:/*Min*/0.0;
                            CASE e_Length_Meter_LONG:/*Min*/0.0;
                            CASE e_Length_Foot_LONG:/*Min*/0.0;
                            CASE e_Length_Inch_LONG:/*Min*/0.0;
                     }

              EDIT_FORMAT
                     SELECT (SU_DecimalPlacesLength_1)
                     {
                            CASE e_DecimalPlacesSelector_X_LONG: ".0f";
                            CASE e_DecimalPlacesSelector_X_X_LONG: ".1f";
                            CASE e_DecimalPlacesSelector_X_XX_LONG: ".2f";
                            CASE e_DecimalPlacesSelector_X_XXX_LONG: ".3f";
                            CASE e_DecimalPlacesSelector_X_XXXX_LONG: ".4f";
                     }

              DISPLAY_FORMAT
                     SELECT (SU_DecimalPlacesLength_1)
                     {
                            CASE e_DecimalPlacesSelector_X_LONG: ".0f";
                            CASE e_DecimalPlacesSelector_X_X_LONG: ".1f";
                            CASE e_DecimalPlacesSelector_X_XX_LONG: ".2f";
                            CASE e_DecimalPlacesSelector_X_XXX_LONG: ".3f";
                            CASE e_DecimalPlacesSelector_X_XXXX_LONG: ".4f";
                     }

       }
       DEFAULT_VALUE 0;
}
```


Figure 15 – Variable EDD mappée avec un type BaseDataVariableType OPC-UA

En raison d'une règle implicite (voir 6.1) le nom de la variable EDD est devenu le nom d'exploration (BrowseName) de la variable OPC-UA. Une autre règle implicite a été appliquée pour l'attribut HANDLING (voir 6.3) et un transtypage implicite a été employé pour mapper le type de l'attribut de valeur de la variable EDD avec un type de données approprié dans l'objet OPC-UA (voir 6.13). Outre les attributs name, label, value et value datatype, aucun autre attribut de la variable EDD n'a pas pu être implicitement mappé avec cet objet OPC-UA, pas même l'unité. Cela s'explique par le fait qu'un type BaseVariableType ne prend pas en charge les unités ni même une plage qui comporte par exemple une valeur minimale (MIN_VALUE) ou une valeur maximale (MAX_VALUE). A cette fin, un type de données plus approprié qui comporte des propriétés supplémentaires pour une plage technique et une unité technique, comme AnalogUnitRangeType, peut être utilisé.

```
SEMANTIC_MAP BlockingDistance_mp
{
    "OPC-UA//__UA_/AnalogUnitRangeType" : SMR_HighBlockDistance_2
}
```

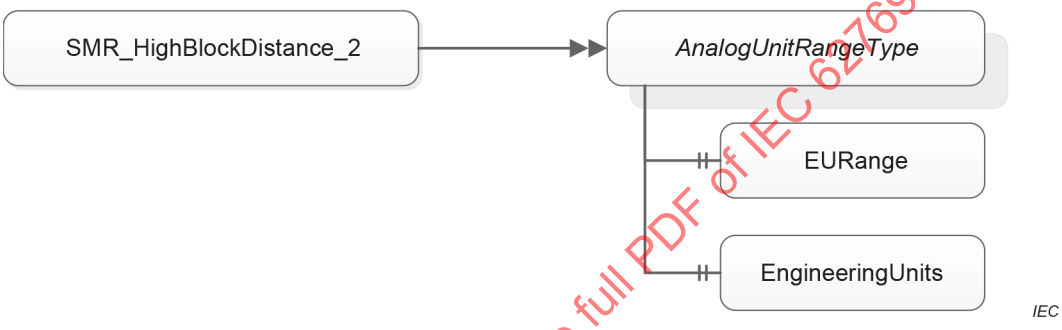


Figure 16 – Exemple de mapping simple avec plage et unité

Dans ce cas, deux règles implicites supplémentaires ont été appliquées, une pour l'unité (voir 6.5) et une pour la plage (voir 6.7). Si le mapping par défaut des règles implicites ne répond pas aux besoins, ces règles peuvent être remplacées par un mapping plus explicite (voir 5.8 et 5.9).

5.8 Mapping explicite de types OPC-UA complexes

Pour les types OPC-UA qui contiennent plusieurs attributs, objets, variables ou propriétés qui ne peuvent pas être implicitement mappés au moyen d'une règle de mapping implicite, ou s'il est nécessaire que le mapping implicite soit annulé, une collection EDD est mappée avec une instance d'un type d'objet ou de variable OPC-UA.

Dans l'exemple représenté à la Figure 17, une instance de PressureMeasurementVariableType tirée de l'espace de noms http://opcfoundation.org/UA/PADIM/ est mappée avec la collection "Pressure_1".

```
SEMANTIC_MAP PressureVariableMap
{
   "OPC-UA//__PADIM_/PressureMeasurementVariableType" : Pressure_1
}

COLLECTION Pressure_1
{
   LABEL "PV";
   HELP [pressure_help];
   MEMBERS
   {
        __UA_Value,                press.DIGITAL_VALUE;
        __PADIM_ActualValue,       pressureInternalValue;
        __PADIM_SimulationValue,   pressureOverrideValue;
        __PADIM_SimulationState,   pressureSimulation;
        __PADIM_Damping,           press.DAMPING;
   }
}
```

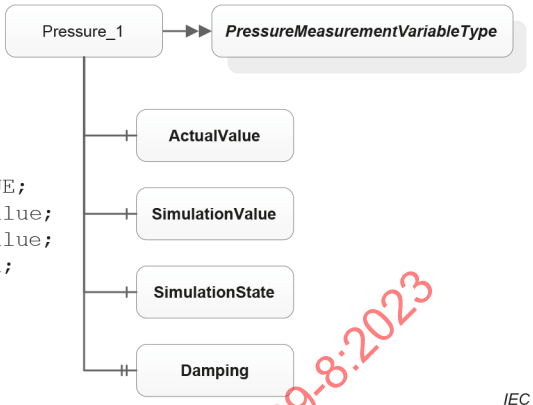


Figure 17 – Mapping d'une collection avec une variable OPC-UA

Les identificateurs de membre préfixés (par exemple, __PADIM_Value, __PADIM_ActualValue, etc.) des membres de la collection sont utilisés pour mapper les objets EDD référencés (par exemple, press.DIGITAL_VALUE, pressureInternalValue, etc.) avec les propriétés, les variables ou les composants de l'objet OPC-UA au moyen de leur nom d'exploration correspondant. Pour "ActualValue", "SimulationValue", "SimulationState" et "Damping", le mapping est très explicite.

Même si aucune valeur n'est explicitement indiquée dans le type OPC-UA du côté droit de l'image, elle est présente sous forme d'attribut avec le nom "Value" faisant partie intégrante de chaque type de variable. Cet attribut est indiqué dans la spécification de base de l'OPC-UA et le préfixe __UA_ est donc utilisé conformément à la définition de la collection d'espaces de noms (voir 5.3).

Voir la définition complète des attributs pour les types de variables dans l'IEC 62541-3.

Ajouter l'alias d'espace de noms comme préfixe au nom de membre permet de s'assurer que les objets EDD sont clairement attribués à l'élément correspondant du côté OPC-UA, même lorsque les noms d'exploration sont identiques (mais appartiennent à des espaces de noms différents). Un effet secondaire est que pour les EDD HART, l'identificateur d'espace de noms peut être choisi d'une manière qui n'entraîne aucun conflit avec des noms de membre (identificateur de membre) déjà existants.

Il n'est pas nécessaire que l'ordre des membres de la collection EDD soit identique à celui des déclarations d'instance (propriétés, variables, attributs, etc.) du type OPC-UA.

Ce mécanisme de mapping implique que les éléments de données mappés appartiennent à un type d'objet ou de variable OPC-UA en tant qu'attributs ou sont liés au type par un type de référence hasComponent ou hasProperty. Les autres relations d'objets fondées sur d'autres types de références doivent être explicitement déclarées. Cela vaut également pour les composants ou les propriétés supplémentaires qui n'appartiennent pas au type d'objet d'origine mappé avec la collection. Pour plus d'informations, se référer au 5.14.

De plus, les règles de mapping implicite s'appliquent effectivement de la même manière que si le mapping avait été réalisé directement, comme cela est indiqué en 5.7. Pour cet exemple précis, cela signifie que l'unité et la MIN_VALUE/MAX_VALUE de la variable placée derrière "press.DIGITAL_VALUE" sont implicitement mappées avec "EngineeringUnit" et "EURange", respectivement, du type "PressureMeasurementVariableType", qui est indirectement dérivé du type AnalogUnitRangeType.

5.9 Mapping explicite d'un objet imbriqué et de types de variables

5.9.1 Généralités

Définir une seule collection peut ne pas être suffisant pour mapper toutes les informations mises à disposition par un objet EDD avec un type d'objet ou de variable complexe. En définissant des collections et des menus EDD imbriqués supplémentaires, où chaque collection/menu fait référence à une déclaration d'instance OPC-UA d'un type complexe, l'imbrication des collections/menus EDD indique les relations d'imbrication des types d'objets OPC-UA et des types de variables OPC-UA, respectivement.

Si le type d'objet OPC-UA mappé représente un modèle d'information complet (PADIMType, par exemple), l'imbrication est susceptible de s'étendre sur plusieurs niveaux. L'exemple suivant doit montrer comment cela peut être fait et indiquer à quel moment utiliser des collections EDD et des menus EDD (voir Figure 17).

Lors de la configuration des collections et des menus, une attention particulière doit être portée à la distinction entre composition et dérivation. Alors que la composition engendre toujours un niveau d'imbrication supplémentaire, la dérivation engendre des identificateurs de membre supplémentaires au même niveau que le reste du type mappé; ces déclarations d'instances de référence ne se perçoivent pas de manière évidente en observant simplement le type dérivé.

5.9.2 Utilisation de collections

Lorsqu'il existe une relation clairement définie entre des déclarations d'instances et des éléments de données ou des conteneurs de données et que le nom des déclarations d'instances peut être attribué sans ambiguïté à précisément une construction EDD, une collection doit être utilisée pour représenter le type parent de ces déclarations d'instances. Voir 5.8 pour un exemple.

5.9.3 Utilisation de menus

Lorsque le nom d'une déclaration d'instance ne peut pas être attribué sans ambiguïté à un seul élément de données ou conteneur de données, un menu EDD doit être mappé avec la déclaration d'instance afin de servir de conteneur pour plusieurs éléments de données ou conteneurs de données qui sont tous du même type que celui défini par la déclaration d'instance à laquelle est mappé le menu.

Par exemple, cela vaut pour les déclarations d'instances de type "FolderType" ou "SignalSetType". Dans l'exemple suivant, cela a été appliqué pour le "SignalSet" de "PADIMType" (pour de plus amples informations sur PADIMType, voir OPC 30081).

5.9.4 Diagramme OPC-UA d'un exemple de mapping imbriqué

La Figure 18 représente un exemple de mapping imbriqué d'objets et de variables.

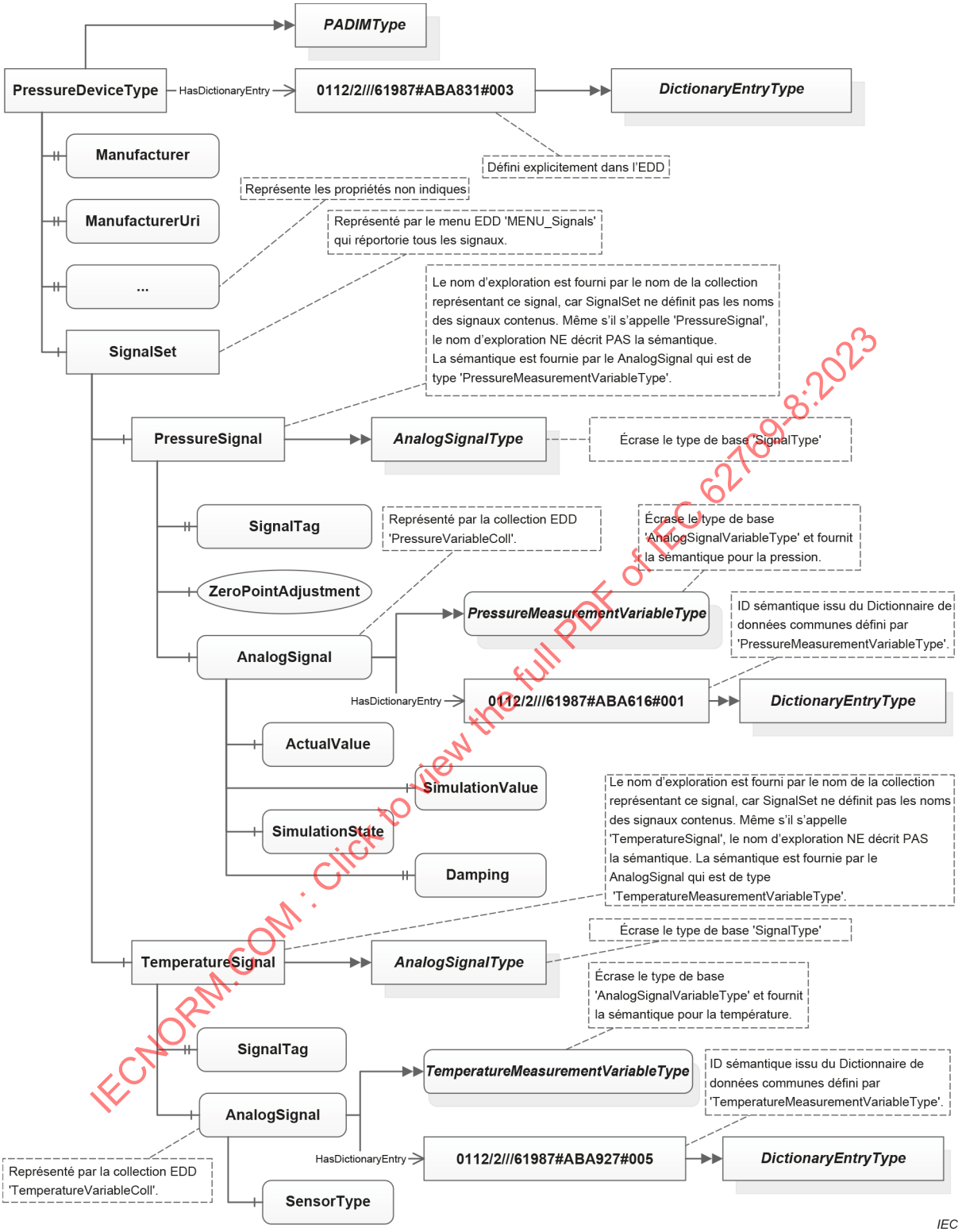


Figure 18 – Objets et variables imbriqués

5.9.5 Extrait EDD d'un exemple de mapping imbriqué

5.9.5.1 Généralités

Cet exemple montre comment les constructions EDD doivent être utilisées pour mapper des parties du modèle d'appareil interne d'une EDD avec un modèle d'information complexe dans l'OPC-UA (PA-DIM, par exemple). Même si les mêmes principes s'appliquent pour mapper des événements et des alarmes OPC-UA, ceux-ci sont décrits dans un article distinct afin de ne pas alourdir cet exemple.

5.9.5.2 Mapping explicite de plages

Même si "EngineeringUnits" du type "AnalogUnitRangeType" est implicitement mappé avec l'unité de la variable EDD mappée, les plages "EURange" et "InstrumentRange" prennent de manière inadéquate la valeur par défaut MIN_FLOAT et MAX_FLOAT, respectivement, si aucune valeur MIN_VALUE/MAX_VALUE n'a été définie dans la variable EDD. Dans les exemples suivants, deux collections supplémentaires sont définies pour mapper de manière explicite les plages "EURange" et "InstrumentRange". Ces deux plages sont définies par le type "AnalogUnitRangeType", qui est un type de base indirect du type "PressureMeasurementVariableType".

L'exemple admet par hypothèse que des variables locales ont été définies pour les limites supérieures et inférieures, respectivement, des plages référencées.

5.9.5.3 Remplacement du type d'une instance de déclaration

Le SEMANTIC_MAP "AnalogSignalMap" est nécessaire étant donné que la déclaration d'instance "SignalSet" est un type de "SignalType" et que les deux signaux qui correspondent à la pression et à la température sont des types de "AnalogSignalType".

Puisque le signal "AnalogSignal" du type "AnalogSignalType" est un type de base générique (AnalogSignalVariableType) qui ne reflète aucun principe de mesure, le SEMANTIC_MAP "PressureVariableMap" supplémentaire est utilisé pour remplacer le type de base par le type "PressureMeasurementVariableType" afin de refléter les principes de mesure de la pression. Cela ne fonctionne que si le type de remplacement est dérivé de manière directe ou indirecte du type de base. Il en est de même pour "TemperatureVariableMap".

5.9.5.4 Suggestions supplémentaires

Englober "AnalogSignalVariableType" avec "AnalogSignalType" est nécessaire pour amener les méthodes du type d'objets "AnalogSignalType" dans le contexte du type de variables "AnalogSignalVariableType", car dans l'OPC-UA, ajouter des méthodes directement à des types de variables n'est pas admis (pour des raisons historiques).

```
/* Assigns pressure device semantic and OPC-UA PADIMType */
SEMANTIC_MAP DeviceTypeMap
{
    "OPC-UA//__PADIM_/PADIMType", "0112/2///61987#ABA831#003": PressureDeviceType
}

/* Although called 'PressureDeviceType' the collection name does NOT add any semantic */
/* The semantic comes in by the semantic map above.                                    */
COLLECTION PressureDeviceType
{
    LABEL "Pressure devicetype";
    MEMBERS
    {
       __DI_Manufacturer,           DeviceManufacturer_local;
       __DI_ManufacturerUri,  DeviceManufacturerUrl;
       __DI_Model,                  DeviceNameString;
       __DI_SerialNumber,           DeviceSerialNumber;
       __DI_ProductCode,            DeviceOrderCode;
       __DI_HardwareRevision, PaPbHardwareRevision;
       __DI_SoftwareRevision, PaPbSoftwareRevision;
       __DI_RevisionCounter,  ConfigurationCounter;
       __DI_ProductInstanceUri,     METH_CalcInstanceUri;
       __DI_AssetId,                PaPbDescriptor;
       __DI_DeviceHealth,           METH_CalcDeviceHealthStatus;
       __DI_DeviceHealthAlarms,     MENU_DeviceHealthAlarms;
       __PADIM_SignalSet,           MENU_Signals;
    }
}

/* Signals from PA-DIM SignalSet    */
MENU MENU_Signals
{
    LABEL "SignalSet";
    ITEMS
    {
        PressureSignal,
        TemperatureSignal
    }
}

/* Defines OPC-UA type for both signals */
SEMANTIC_MAP AnalogSignalMap
{
    "OPC-UA//__PADIM_/AnalogSignalType": PressureSignal, TemperatureSignal
}

/* Eplicitly mapped to the __PADIM_ type 'AnalogSignalType'*/
COLLECTION PressureSignal
{
   VALIDITY IF (OperatingMode==0) {TRUE;} ELSE { FALSE;}
   LABEL "Pressure signal";
   MEMBERS
   {
       __PADIM_SignalTag,                  PrimValDesc;
       __PADIM_AnalogSignal,        PressureVariableColl;
       __PADIM_ZeroPointAdjustment, SetPressPv2Zero;
   }
}

SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/PressureMeasurementVariableType": PressureVariableColl
}

/* Eplicitly mapped to the __PADIM_ type 'PressureMeasurementVariableType'*/
COLLECTION PressureVariableColl
{
   LABEL "Pressure PV";
   MEMBERS
```

```
    {
        __UA_Value,                  PaTbPrimaryValueValue;
        __PADIM_ActualValue,   PaTbCurrentValue1Value;
        __PADIM_SimulationValue,     Pressure1SimulationValue;
        __PADIM_SimulationState,     SimulationMode;
        __PADIM_Damping,             ActivePressure1Damping;
        __UA_InstrumentRange,  COL_InstRange_PressVal;
        __UA_EURange,                COL_ProcRange_PressVal;
    }
}

/* Implicitly mapped to the __UA_ type 'AnalogUnitRangeType' */
COLLECTION COL_PressureValue
{
LABEL "Pressure value";
   HELP [blank];
   MEMBERS
   {
          __UA_Value,                PaTbPrimaryValueValue;
          __UA_InstrumentRange,      COL_InstRange_PressVal;
          __UA_EURange,              COL_ProcRange_PressVal;
   }
}

/* Implicitly mapped to the __UA_ structure 'Range' */
COLLECTION COL_InstRange_PressVal
{
LABEL "Pressure value instrument range";
   HELP [blank];
   MEMBERS
   {
       __UA_low,       Pressure1SensorLowLimit;
       __UA_high,      Pressure1SensorHighLimit;
   }
}

/* Implicitly mapped to the __UA_ structure 'Range' */
COLLECTION COL_ProcRange_PressVal
{
   LABEL "Pressure value process range";
   HELP [blank];
   MEMBERS
   {
       __UA_low,       PaTbScaleIn0;
       __UA_high,      PaTbScaleIn100;
   }
}

/* Eplicitly mapped to the __PADIM_ type 'AnalogSignalType'*/
COLLECTION TemperatureSignal
{
   VALIDITY IF (OperatingMode==1) {TRUE;} ELSE { FALSE;}
   LABEL "Temperature signal";
   MEMBERS
   {
       __PADIM_SignalTag,           SecValDesc;
       __PADIM_AnalogSignal,  TemperatureVariableColl;
   }
}

SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/TemperatureMeasurementVariableType": TemperatureVariableColl
}

/* Eplicitly mapped to the __PADIM_ type 'TemperatureMeasurementVariableType'         */
/* Optional components were not available and therefore not mapped                    */
COLLECTION TemperatureVariableColl
{
   LABEL "Temperature SV";
```

```
    MEMBERS
    {
        __UA_Value,             VarTemperatureValue;
        __PADIM_SensorType,     VarSensType;
    }
}
/* This method returns a DD_STRING that contains the ProductInstanceUri which         */
/* (obviously) does not exist in the device as such and therefore must calculated     */
/* based on three variables where at least the 'DeviceSerialNumber' must be read from */
/* the device to make it unique.                                                      */
/* The other two variables could be local variables with adequate default values.     */
/*                                                                                    */
/* Note:                                                                              */
/* Including a device name is not really necessary if the serial number is guaranteed */
/* to be unique for the ManufactureUrl                                                */
METHOD METH_CalcInstanceUri
{
    LABEL "Calculate product instance URI";
    TYPE DD_STRING;
    DEFINITION
    {
        DD_STRING uri;

        uri = DeviceManufacturerUrl + "/" + DeviceNameString + "/" +
DeviceSerialNumber;
        return uri;
    }
}
```

Figure 19 – Exemples d'objets et de variables imbriqués

5.10 Mapping explicite de méthodes

5.10.1 Mapping de méthodes EDD avec des objets, des variables ou des propriétés OPC-UA

Des méthodes EDD peuvent être utilisées pour calculer des valeurs à la demande. Cela est également utile si un élément de données particulier attendu par le modèle d'information OPC-UA n'existe pas dans l'EDD. Lorsqu'une méthode EDD est mappée avec un élément de données OPC-UA, la valeur de retour est intégrée à la valeur de l'élément de données.

Voir "ProductInstanceUri" à la Figure 19, qui n'existe sous forme de valeur correcte ni dans l'EDD ni dans l'appareil (voir 6.5).

5.10.2 Mapping de méthodes EDD avec des méthodes OPC-UA

Le mapping de méthodes fonctionne essentiellement de la même manière que le mapping de types complexes. Étant donné qu'une méthode appartient toujours à une instance d'un type d'objet, une méthode simple sans argument ni valeur de retour conforme à la Figure 20 est mappée de la même manière que celle décrite en 5.8.

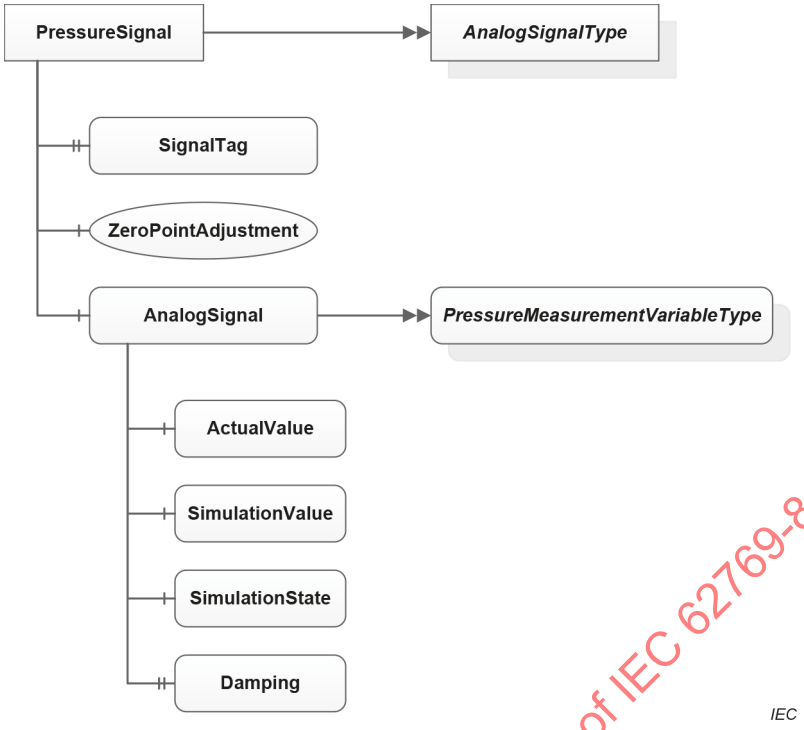


Figure 20 – Méthode simple

Toutefois, "Common Service Result Codes" doit toujours être renvoyé (par exemple, Good, Bad_InternalError Bad_CertificateInvalid, Bad_DataTypeIdUnknown, Bad_Shutdown, Bad_Timeout, etc.) en tant que code de statut de l'en-tête de réponse. Les codes de résultat de service de l'en-tête de réponse sont fournis par le serveur OPC-UA et ne sont donc pas liés au fonctionnement ou à l'exécution de la méthode EDD interne. Tout statut opérationnel généré par l'exécution de la méthode doit être renvoyé en tant que valeur de retour de la méthode et doit néanmoins être défini de manière appropriée dans la définition de la méthode (voir 6.5).

La Figure 19 représente un exemple EDD de mapping de méthode simple.

```
/* Defines OPC-UA type for the PressureSignal*/
SEMANTIC_MAP PressureSignalMap
{
    "OPC-UA//__PADIM_/AnalogSignalType": PressureSignal
}

/* Eplicitly mapped to the __PADIM_ type 'AnalogSignalType'*/
COLLECTION PressureSignal
{
   VALIDITY IF (OperatingMode==0) {TRUE;} ELSE { FALSE;}
   LABEL "Pressure signal";
   MEMBERS
   {
        __PADIM_SignalTag,                PrimValDesc;
        __PADIM_AnalogSignal,             PressureVariableColl;
        __PADIM_ZeroPointAdjustment,      SetPressPv2Zero;
   }
}
METHOD SetPressPv2Zero
{
    LABEL     "Sets the current PV to zero";
    TYPE      UNSIGNED_INTEGER(2);
    DEFINITION
    {
              UNSIGNED_INTEGER(2) result;
              BOOLEAN timeout, succeeded;
```

```
            /* initialize result with 'BadUnexpectedError' */
            result =  = 0x80010000;

            timeout   = TRUE;
            succeeded = FALSE;

            /*******************************************************/
            /*                                                     */
            /* do something and set booleans accordingly           */
            /*                                                     */
            /*******************************************************/


            if (timeout == true)
            {
                /* set result to 'BadTimeout' */
                result =  = 0x800A0000;
            }
            else if (succeeded)
            {
                /* set result to 'Good' */
                result =  = 0x00000000;
            }

            return result;
    }
}


SEMANTIC_MAP PressureVariableMap
{
    "OPC-UA//__PADIM_/PressureMeasurementVariableType": PressureVariableColl
}

/* Eplicitly mapped to the __PADIM_ type 'PressureMeasurementVariableType'*/
COLLECTION PressureVariableColl
{
   LABEL "Pressure PV";
   MEMBERS
   {
       __UA_Value,              PaTbPrimaryValueValue;
       __PADIM_ActualValue,     PaTbCurrentValue1Value;
       __PADIM_SimulationValue, Pressure1SimulationValue;
       __PADIM_SimulationState, SimulationMode;
       __PADIM_Damping,         ActivePressure1Damping;
       __UA_InstrumentRange,    COL_InstRange_PressVal;
       __UA_EURange,            COL_ProcRange_PressVal;
   }
}
```


Figure 21 – Exemple de mapping de méthode simple

5.11 Mapping explicite d'alarmes

Seuls les quatre types d'alarmes dérivés de *DeviceHealthDiagnosticAlarmType*et définis par l'IEC 62541-100 doivent être pris en charge dans la première édition du présent doucment (voir Figure 22 et Figure 23). D'autres types d'alarmes seront ajoutés selon les besoins dans de futures éditions.

- FailureAlarmType,
- CheckFunctionAlarmType,
- OffSpecAlarmType,
- MaintenanceRequiredAlarmType.

Ces types d'alarmes indiquent l'état namur conformément à la NE 107 [1][7].

Les types d'alarmes sont mappés de la même manière que les types d'objets et les types de variables (voir 5.9).

Même si les quatre types d'alarmes peuvent être activés, seule une alarme doit être active à la fois.

Les alarmes et leurs événements ne doivent pas être réglés ou déclenchés par la configuration d'un ensemble de données hors-ligne.

NOTE Les quatre alarmes de santé des appareils sont directement dérivées de *DeviceHealthDiagnosticAlarmType* sans ajout d'une quelconque propriété supplémentaire. Il en est de même pour *DeviceHealthDiagnosticAlarmType*, qui est directement dérivé de *InstrumentDiagnosticAlarmType*.

Le Tableau 1 indique comment gérer toutes les propriétés obligatoires ainsi que certaines des propriétés facultatives. Le concept qui sous-tend les indications sur la manière de gérer ces propriétés repose sur l'approche selon laquelle ActiveState est l'état le plus important qu'il est nécessaire de mettre à jour d'après l'état actuel de l'appareil alors que tous les objets d'alarme sont parallèlement visibles dans l'espace d'adressage.

7 Les chiffres entre crochets renvoient à la Bibliographie.

Tableau 1 – Mapping des propriétés d'alarmes

	DeviceHealthDiagnosticAlarmType		
	InstrumentDiagnosticAlarmType		
	OffNormalAlarmType		
	BrowseName	Valeur	Défini par
	NormalState	NULL	Hôte
	DiscreteAlarmType		
	AlarmConditionType		
	BrowseName	Valeur	Défini par
	ActiveState	Doit être explicitement mappée avec une variable booléenne qui reflète l'état de cette alarme. Seule l'une des quatre alarmes DeviceHealthDiagnosticAlarms prises en charge doit être active à la fois. True = active False = inactive Le texte localisé de TrueState et FalseState, respectivement, doit être défini par l'hôte conformément à la recommandation donnée dans l'IEC 62541-9:2020, Annexe A. Il s'agit de la seule variable d'état qui doit être définie par l'EDD pour convenablement représenter l'état de l'appareil. Toutes les autres variables d'état interagissent avec l'utilisateur du système d'alarme de l'hôte. Informatif: Le serveur déclenche des événements lors de transitions d'états. Du point de vue de l'EDD, il s'agit de la propriété la plus importante qui doit être gérée par l'EDD pour fournir des états d'alarme appropriés.	EDD
	InputNode	NULL	Hôte
	SuppressedState	Partie du système d'alarme de l'hôte. Ne doit pas être définie par l'EDD. Tout mapping explicite de cette variable doit être ignoré par l'hôte et/ou doit être considéré comme une erreur lors de la tokenisation de l'EDD.	Hôte
	OutofServiceState	Partie du système d'alarme de l'hôte. Ne doit pas être définie par l'EDD. Tout mapping explicite de cette variable doit être ignoré par l'hôte et/ou doit être considéré comme une erreur lors de la tokenisation de l'EDD.	Hôte
	ShelvingState	Partie du système d'alarme de l'hôte. Ne doit pas être définie par l'EDD. Tout mapping explicite de cette variable doit être ignoré par l'hôte et/ou doit être considéré comme une erreur lors de la tokenisation de l'EDD.	Hôte
	SuppressedOrShelved	Partie du système d'alarme de l'hôte. Ne doit pas être définie par l'EDD. Tout mapping explicite de cette variable doit être ignoré par l'hôte et/ou doit être considéré comme une erreur lors de la tokenisation de l'EDD. Si l'hôte ne prend en charge aucun des états décrits ci-dessus, la valeur est définie par défaut sur false. Pour plus d'informations, voir l'IEC 62541-9.	Hôte

	AcknowledgeableConditionType		
	BrowseName	Valeur	Défini par
	AckedState	Partie du système d'alarme de l'hôte. Ne doit pas être définie par l'EDD. Tout mapping explicite de cette variable doit être ignoré par l'hôte et/ou doit être considéré comme une erreur lors de la tokenisation de l'EDD. Si l'hôte ne prend pas en charge l'acquittement, la valeur est définie par défaut sur true. Pour plus d'informations, voir l'IEC 62541-9.	Hôte
	ConfirmedState	Partie du système d'alarme de l'hôte. Ne doit pas être définie par l'EDD. Tout mapping explicite de cette variable doit être ignoré par l'hôte et/ou doit être considéré comme une erreur lors de la tokenisation de l'EDD. Si l'hôte ne prend pas en charge la confirmation, la valeur est définie par défaut sur true. Pour plus d'informations, voir l'IEC 62541-9.	Hôte
	ConditionType		
	BrowseName	Valeur	Défini par
	ConditionClassId	Doit être définie sur l'ID de nœud de MaintenanceConditionClassType.	Hôte
	ConditionClassName	Doit être définie sur la chaîne "MaintenanceConditionClassType".	Hôte
	ConditionName	Nom d'état NAMUR NE 107 [1] sous forme de chaîne, qui est associé à l'alarme de santé concrète de l'appareil.	Hôte
	BranchId	NULL	Hôte
	Retain	Si l'alarme est active, Retain doit être défini sur true et dans le cas contraire sur false. L'hôte synchronise cet état d'après l'état de l'alarme.	Hôte
	EnabledState	Dès que l'objet d'alarme est disponible dans l'espace d'adressage, cet état doit être défini sur true, et dès que l'objet d'alarme disparaît, il doit être défini sur false. Ceci est important pour que les clients soient informés par les notifications d'événement correspondantes de l'existence de nouveaux objets d'alarme auxquels il est souhaitable qu'ils s'abonnent, ou desquels ils doivent se désabonner, car ils n'existent plus. Si la validité de l'objet EDD avec lequel l'alarme est mappée n'est pas conditionnée ou est explicitement définie sur true, ce fanion doit être défini sur true lors du chargement de l'EDD et sur false lors du déchargement de l'EDD, respectivement, pour déclencher les notifications d'événement appropriées. La validité de l'objet EDD peut comporter des attributs conditionnels pour contrôler la disponibilité de l'alarme en fonction de l'état de l'appareil (lors du téléchargement sur l'appareil, par exemple). Pour plus d'informations, voir l'IEC 62541-9.	Hôte/EDD
	Quality	Doit être définie sur "Good".	Hôte
	LastSeverity	Définie par l'hôte à la modification de la sévérité actuelle. Voir au-dessous de *BaseEventType* pour apprendre à régler la sévérité actuelle.	Hôte
	Comment	La valeur par défaut est NULL, mais elle peut être définie par l'hôte. Ne doit pas être définie par l'EDD.	Hôte
	ClientUserId	Définie par l'hôte si comment n'a pas la valeur NULL.	Hôte

	BaseEventType		
	BrowseName	Valeur	Défini par
	EventId	Voir l'IEC 62541-5	Hôte
	EventType	Voir l'IEC 62541-5	Hôte
	SourceNode	ID de nœud de l'instance PADIMType.	Hôte
	SourceName	Nom d'affichage de l'instance PADIMType.	Hôte
	Time	Voir l'IEC 62541-5	Hôte
	ReceiveTime	Même valeur que celle de "Time", voir ci-dessus.	Hôte
	Message	Doit être mappée de manière explicite par l'EDD de manière à contenir des informations relatives à la cause et à la solution.	EDD
	Severity	Si elle n'est pas mappée par l'EDD, le serveur règle cette valeur à un niveau approprié qui est logique dans le contexte du système dont il fait partie. Si elle est mappée par l'EDD, elle doit être réglée sur une valeur comprise entre 1 et 1 000. Le serveur est autorisé à remplacer la valeur définie par l'EDD. Voir l'IEC 62541-5 pour plus d'informations. NOTE La modification de la sévérité déclenche une notification d'événement.	Hôte/EDD

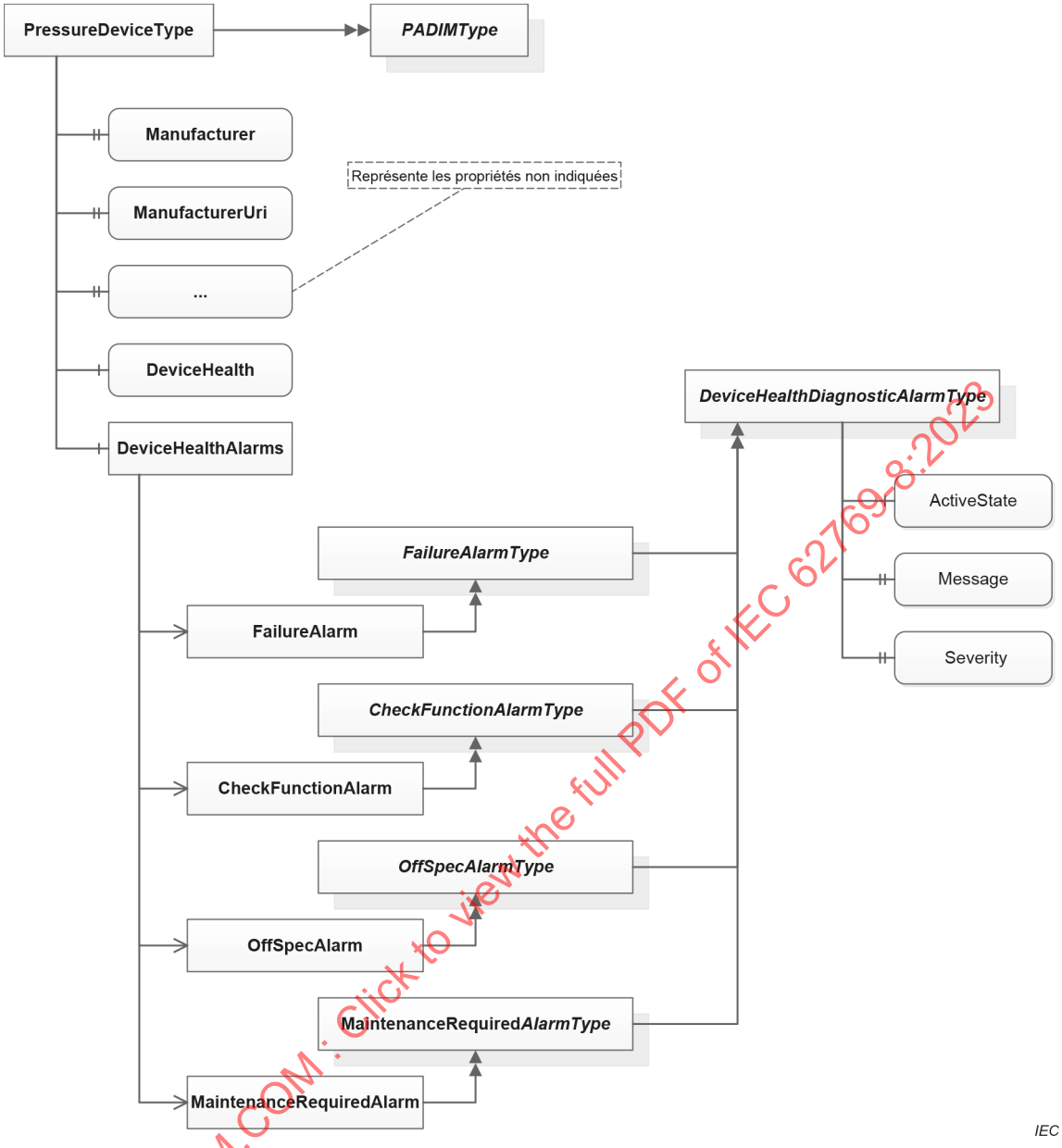


Figure 22 – Alarmes prises en charge

```
SEMANTIC_MAP DeviceTypeMap
{
    "OPC-UA//__PADIM_/PADIMType": PressureDeviceType
}

COLLECTION PressureDeviceType
{
  LABEL "Pressure devicetype";
  MEMBERS
  {
       __DI_Manufacturer,          __resource_block_2[0].PARAM.MANUFAC_ID;
       __DI_ManufacturerUri,       __resource_block_2[0].PARAM.MANUFACTURER_URI;
       __DI_Model,                 __resource_block_2[0].PARAM.DEV_TYPE;
       __DI_SerialNumber,          __resource_block_2[0].PARAM.DEVICE_SN;
       __DI_ProductCode,           __resource_block_2[0].PARAM.DEVICE_PRODUCT_CODE;
       __DI_HardwareRevision,      __resource_block_2[0].PARAM.HARDWARE_REVISION;
       __DI_SoftwareRevision,      __resource_block_2[0].PARAM.DEV_REV;
       __DI_RevisionCounter,       __resource_block_2[0].LOCAL_PARAM.GLOBAL_ST_REV;
       __DI_ProductInstanceUri,    __resource_block_2[0].METHOD.RB_METH_CALCINSTANCE_URI;
       __DI_AssetId,               __resource_block_2[0].PARAM.ASSET_ID;

//Helper methods to fill in information NE107 status
       __DI_DeviceHealth,
__resource_block_2[0].METHOD.RB_METH_CALC_DEVICE_HEALTH;

       //optional but strongly recommended to provide this information
       __DI_DeviceHealthAlarms,    DEVICE_HEALTH_ALARMS;
  }
}

METHOD RB_METH_CALC_DEVICE_HEALTH
{
    LABEL "Calculate NE107";
    TYPE unsigned char;
    DEFINITION
    {
       unsigned char status;
       unsigned char NE107_status;
       DD_STRING strEnum;

   int iFailure, iMaintReq, iFunCheck, iOffSpec;

   NE107_status = 0;
   iFailure = 0;
   iMaintReq = 0;
   iFunCheck = 0;
       iOffSpec = 0;

       ReadCommand(read_DeviceStatus);
       status = DeviceStatus;

       switch (status)
       {
       case e_DeviceStatus_Status_Good:
       NE107_status = 0;
       break;
       case e_DeviceStatus_Status_Failure:
       NE107_status = 1;
       iFailure = 1;
       break;
       case e_DeviceStatus_Status_MaintenanceReq:
       NE107_status = 4;
       iMaintReq = 1;
       break;
       case e_DeviceStatus_Status_FunctionCheck:
       NE107_status = 2;
       iFunCheck = 1;
       break;
       case e_DeviceStatus_Status_OutOfSpec:
```