International Standard

ISO 22166-201

First edition
2024-02

Robotics — Modularity for service robots —

Part 201: Common information model for modules

Robotique — Modularité des robots de service —

Partie 201: Modèle d'information commun pour les modules

Reference number
ISO 22166-201:2024(en)

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 299, *Robotics*.

A list of all parts in the ISO 22166 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document provides a common information model (CIM) for modules which compose service robots based on ISO 22166-1. Based on the CIM, modules can easily be connected and data exchanged between them. It has been designed to enhance the interoperability, reusability, and composability of modules. CIM is a meta-model and is a base model for all kinds of modules such as hardware modules, modules with hardware and software aspects, software modules, and their composite modules. Hence CIM provides meta characteristics that modules should possess.

Module makers, module integrators, robot makers, robot developers, and robot system integrators are able to use CIM in order to obtain the necessary modules more easily, utilize various modules according to function and budget, and develop a new (composite) module based on CIM. The CIM is able to make design of modules, operation and maintenance of service robots help.

The CIM provides four information types:

— information for module identification;

— information for module selection;

— information for module integration;

— information for module operation and maintenance.

The CIM, as a meta model, consists of number of submodels, which are Module ID, Properties, Input and Output Variables, Status, Services, Infrastructure, Safety and Security Level, Modelling, Executable Form.

The implementation model of CIM will be provided in upcoming standards, with an example being ISO 22166-202 for software modules.

The ISO 22166 series applies to composite modules and various types of service robots.

This document presents requirements and guidelines on the information model of a service robot's modules for ensuring nine principles of modularity presented in ISO 22166-1.

Robotics — Modularity for service robots —

Part 201:
Common information model for modules

1 Scope

This document specifies requirements and guidelines for the common information model (CIM) for modules of service robots to achieve interoperability, reusability, and composability.

This document specifies the structure of the CIM and details the intended use and meaning of its attributes and subclasses.

This document applies to service robots.

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.

ISO 22166-1:2021, *Robotics — Modularity for service robots — Part 1: General requirements*

IETF RFC 4122, *A Universally Unique IDentifier (UUID) URN Namespace*

IEEE/Open Group 1003.1-2017, *IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
information model
IM
abstraction and representation of the entities in a managed environment, their properties, operations, and the way that they relate to each other

[SOURCE: ISO 22166-1:2021, 3.1.11 — modified]

3.2
common information model
CIM
information model that modules most frequently use in service robots

3.3
module
component or assembly of components with defined interfaces accompanied with property profiles to facilitate system design, integration, interoperability, and reuse

[SOURCE: ISO 22166-1:2021, 3.3.12]

3.4
module property
attribute or characteristic of a module

[SOURCE: ISO 22166-1:2021, 3.3.14]

3.5
software module
module whose implementation consists purely of programmed algorithms

[SOURCE: ISO 22166-1:2021, 3.4.4]

3.6
hardware module
module whose implementation consists purely of physical parts, including mechanical parts, electronic circuits, and any software, such as firmware, not externally accessible through the communication interface

[SOURCE: ISO 22166-1:2021, 3.4.3]

3.7
module with hardware aspects and software aspects
hardware-software module
module whose implementation consists of physical parts, software, and a communication interface that allows data exchange with other modules

3.8
module manager
functions for lifecycle management of module instances including instantiation of modules

3.9
instance
particular entity instantiated from a specific software module or particular entity of a specific module in hardware aspects

Note 1 to entry: In object-oriented programming, instance means a specific realization of an object.

Note 2 to entry: In hardware aspects, instance means one of the same modules used in a composite module.

3.10
performance level
PL
discrete level used to specify the ability of safety-related parts of control systems to perform a safety function under foreseeable conditions

[SOURCE: ISO 13849-1:2023]

3.11
safety integrity level
SIL
discrete level (one out of a possible three) for describing the capability to perform a safety function where safety integrity level three has the highest level, and safety integrity level one has the lowest

[SOURCE: IEC 62061:2021, 3.11]

3.12
executable form
form of the code in which the software or firmware is managed and controlled completely by the operational environment of the module and does not require compilation (e.g. no source code, object code, or just-in-time compiled code)

Note 1 to entry: Its two primary types are compiled programs and scripts.

Note 2 to entry: It can include the set of files and/or directories that make up a complete module or be a single file.

[SOURCE: ISO/IEC 19790:2012, 3.42]

3.13
hardware aspects
information regarding properties and functions necessary for a module and its physical interconnection and regarding the allowed range of physical properties of the operational environment

[SOURCE: ISO 22166-1:2021, 3.3.5]

3.14
software aspects
information regarding the external software properties necessary for a module and its interface and the execution life cycle of that module's function

[SOURCE: ISO 22166-1:2021, 3.3.21]

4 Common information model for modules

4.1 General

Common information for modules shall consist of the items in Table 4.1. The naming of properties and classes used in the information model shall follow the naming rules in Annex A. In Table 4.1, the symbols "M" and "O" denote mandatory and optional, respectively. CIM in Figure 4.1 is described in detail in 4.3. The class models for CIM are drawn based on ISO/IEC 19505-1:2012. The information provided in Table 4.1 and Figure 4.1 can be used in the design, development, operation, and maintenance stages and Annex D shows at which stage the information provided by the information model is used.

NOTE 1 A CIM for managing the common information in Table 4.1 is illustrated in Figure 4.4.

NOTE 2 In this document, CIM representation in XML is used just as examples to help readers understand the meaning of attributes in the class model.

"Module Name" of Item No. 1 in Table 4.1 is the name representing the module. Hereafter, Item No. "N" in Table 4.1 is abbreviated as Item "N".

"Description" of Item 2 provides the overview of the module, what the module is, what it does, and how it is used.

"Manufacturer" of Item 3 provides contact information for the designer, developer, or manufacturer of the module.

"Examples" of Item 4 provides typical use cases of the module.

"Information model version" of Item 5 is the version number of the information model that this document specifies.

Table 4.1 — Common information for modules and the corresponding group tag (used in Figure 4.4)

No.	Item	Common information model [b]	Information models for modules [c]			Related group/tag name (Abbreviation of each group)
			hard-ware-soft-ware module	hardware module	software module	
1	Module Name	M	M	M	M	GenInfo
2	Description	O	O	O	O	
3	Manufacturer	M	M	M	M	
4	Examples	O	O	O	O	
5	Information model version	M	M	M	M	IDnType
6	Module ID	M	M	M	M	
7	Hardware Aspects	O	M	M	-	
8	Software Aspects	O	M	-	M	
9	Module properties [a]	M	M	M	M	Properties
10	Inputs	O	O	O	O	IOVariables
11	Outputs	O	O	O	O	
12	Status	O	O	O	M	—
13	Services (capabilities)	O	M	O	M	Services
14	Infrastructure	O	O	M	M	Infra
15	Safety/security	O	M	O	M	SafeSecure
16	Modelling	O	O	O	O	Modelling
17	ExecutableForm	O	O	O	M	ExecutableForm

[a] It is mandatory only to those which can be influenced (set) from the outside or at least to those which have an expected effect on other modules.

[b] All items are mandatory for CIM.

[c] For information models such as hardware-software modules, hardware modules, or software modules, some types of items can be omitted depending on their functionalities. In particular, for information models of hardware modules and software modules, the items “software aspects” and “hardware aspects” are not included, respectively.

“Module ID” of Item 6 shall be the unique identifier of the module within a system as described in Annex B. Module ID includes information such as which type a module is, hardware-software, hardware, or software, and whether a module is a basic or a composite module.

“Hardware Aspects” of Item 7 and “Software Aspects” of Item 8 relate only to composite modules. If a module is composed of two or more hardware-software modules, software modules, and/or hardware modules, the module IDs are listed in “Hardware Aspects” if they are hardware or hardware-software modules, otherwise (if they are software modules) the module IDs are listed in “Software Aspects”.

“Module properties” of Item 9 are values that are generally used in the initialization of modules. Module properties are classified into mandatory and optional, and shall be listed. The relationship between modules is represented in this item, whose detailed contents will be provided in upcoming standards such as ISO 22166-202.

NOTE 3 Information about the relationships between modules can be provided differently depending on the module type.

NOTE 4 Environment constraints are also considered as properties, examples of which are operating temperature, operating humidity, and maximum allowed mechanical shock. These parameters related to the behaviours of the module can be set in the properties. For example, each coefficient used in the PID (Proportional, Integral and Differential) control algorithm is used once in initialization and is changed and used several times during the execution of related software modules.

“Inputs/Outputs” of Items 10 and 11 describe the names of variables for data transfer into and/or out of a module.

EXAMPLE 1 Inputs and outputs of a servo control software module are the encoder value and motor control values, respectively.

NOTE 5 Properties are kinds of input values from the viewpoint of modules, but have to be distinguished in that Inputs are related to the environment of the module, but Properties are related to parameters contained in the module. For example, Inputs of the servo control software module are encoder values, but its Properties are P, I, and D coefficients, where P, I, and D denote Proportional, Integral and Differential.

“Status” of Item 12 describes the status of a module that is operating.

NOTE 6 Status is not utilized at design and development stages.

“Services (capabilities)” of Item 13 describes the interfaces that a module provides and utilizes for robot services.

NOTE 7 Service means performing one or more functions of a module for other modules via a pre-described interface.

EXAMPLE 2 Examples of function format for software aspects are shown in Table 4.2. In this example, data types such as int16 and unit 8 are defined in Table C.5.

Table 4.2 — Example of function format for software modules

Name	Arguments	Return value	Description
initialize	Integer init_ival1, Real init_rval2	Integer	Initialization using 2 arguments Return value: (0: success, negative value: error type)
	Integer init_ival1, Real init_rval2, Integer init_ival3	Integer	Initialization using 3 arguments Return value: (0: success, negative value: error type)

EXAMPLE 3 An example of a function format for an electrical/electronic aspect is shown in Table 4.3, where the arguments “connType”, “keying”, and “busProtocol” mean the type of connector, the gender of the connector, and the protocol type that the module uses. The function is used to check that the peer module utilizes the proper electrical aspects of the module. Values for connector type can be USB-A, RJ45, DB-9, etc. The value for keying is male or female. Values for busProtocol can be USB, Ethernet, EtherCAT, RS232, etc.

Table 4.3 — Example of function format for electrical/electronic adaptability

Name	Arguments	Return value	Description
checkElecConnectivity	String connType, String keying, String busProtocol	Boolean	Check electrical/electronic adaptability using 3 arguments Return value: (True: success, False: error)

EXAMPLE 4 Examples of a function format for mechanical aspects are provided in Table 4.4. As in the electrical/electronic aspect, the function is used to check that the peer module is utilizing the proper mechanical aspects of the module. However, unlike the electrical/electronic aspect, the function format for the mechanical aspect can be more complicated due to a huge variety being used in practice. Consequently, only two simplified categories are listed here, joint and link.

Table 4.4 — Example of function format for mechanical adaptability

Name	Arguments	Return value	Description
linkConnectivity	Real origin, Real mass, Real Inertia[], String shape, Real size, Real axis[], String connection, Real collision[]	Boolean	Check mechanical link adaptability using 8 arguments Return value: (True: success, False: error)
jointConnectivity	Real origin[], Real axis[], Real limits, Real damping, Real friction	Boolean	Check mechanical joint adaptability using 5 arguments Return value: (True: success, False: error)

“Infrastructure” of Item 14 lists hardware and/or software that the modules commonly use or connect to.

EXAMPLE 5 Examples of Infrastructure are power type, middleware type, databus type, and database type.

“Safety/Security” of Item 15 describes the safety-related performance level and the security information provided by the module.

For the Safety/Security of a general service robot, a safety-related performance level is used for the module, which is defined in ISO 13849-1, and the security-related level is listed for the module, where the security level is “0 – 4”. Security level “0” means that there are no security measures. The security levels “1-4” are defined in IEC 62443-4-2:2019. However, for specific robot types such as medical robots and physical assistant robots, other safety-related standards and security standards should be utilized. A performance level is generally provided for each single safety function. If a module has several safety functions, the module shall provide a performance level of the module using the combination of the performance levels of all the safety functions of the module or via verification and validation of the module’s overall safety-related function. The details are provided in 4.3.8.

“Modelling” of Item 16 provides different kinds of models for simulation or design purposes.

“ExecutableForm” of Item 17 provides program codes executed to achieve or support the module’s purpose.

Classes provided in this document are able to be described using the table format in Annex E.

4.2 Relationship between CIM and specific IMs

The common information model for modules shall be used in information models for all types of modules, which are hardware-software modules, software modules, and hardware modules. Their relationships are shown in Figure 4.1. Hardware-software modules are composed of hardware aspects and software aspects. The relationship between information models of hardware modules, software modules, and hardware-software modules is illustrated in Figure 4.2.

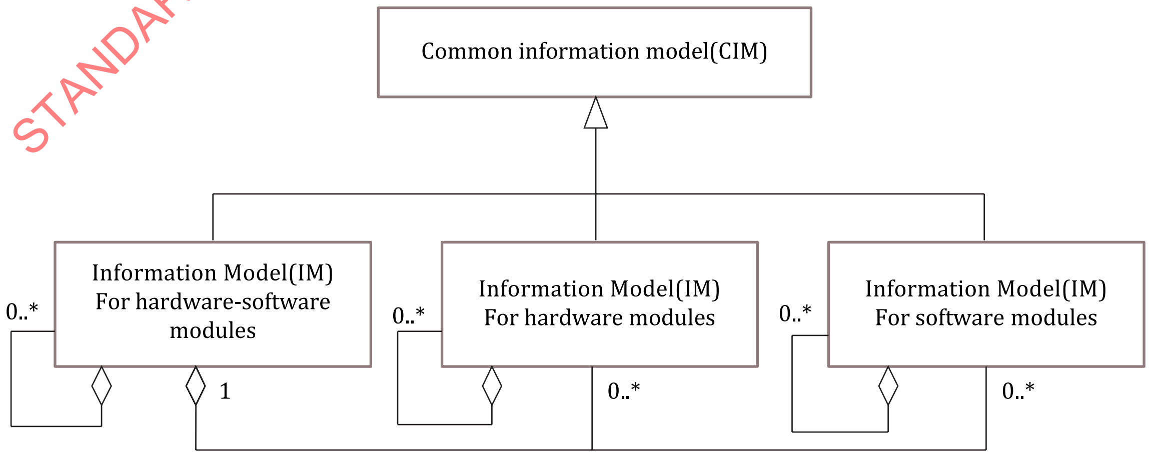

Common information model(CIM)
Information Model(IM) For hardware-software modules
Information Model(IM) For hardware modules
Information Model(IM) For software modules
0..*
1
0..*
0..*
0..*
0..*
0..*

Figure 4.1 — Relationships between information models for modules

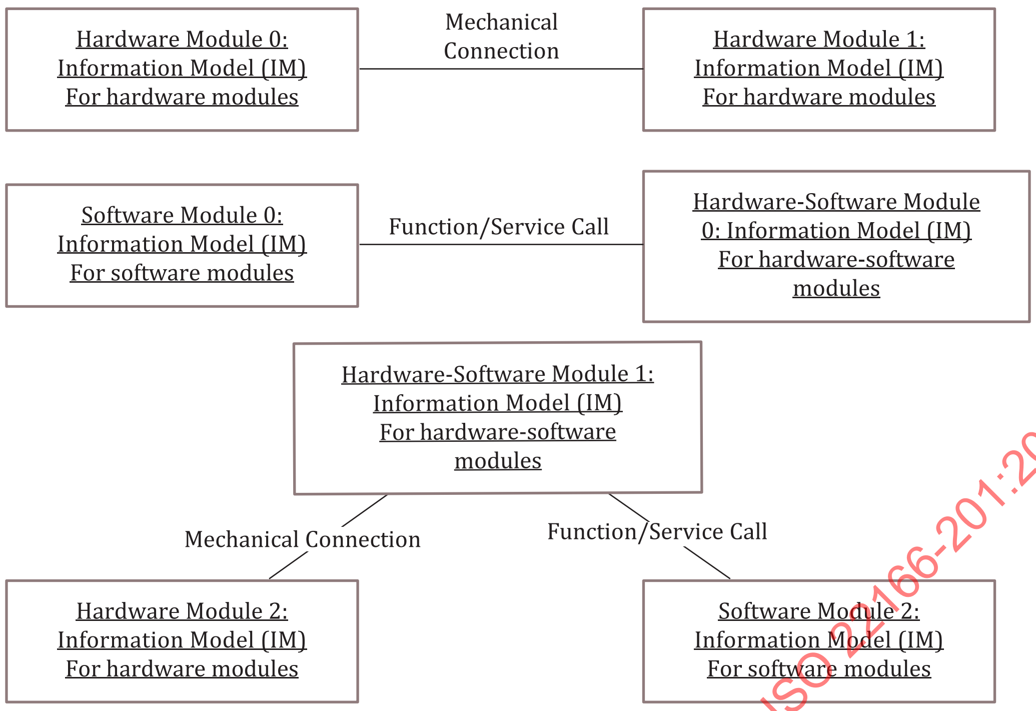


Figure 4.2 — Example of relationships between information models of modules

4.3 Class for common information model

4.3.1 General

The common information model shall be grouped into 9 inner classes as illustrated in Figure 4.3, where eight classes are suggested from Table 4.1 and an additional class, Status, is related to the status-related information of a module. This class is mainly used during operation of the module. The four items of the GenInfo group in Table 4.1, which are Module name, Description, Manufacturer, and Examples, become the member attributes of the class Common Information Model (CIM).

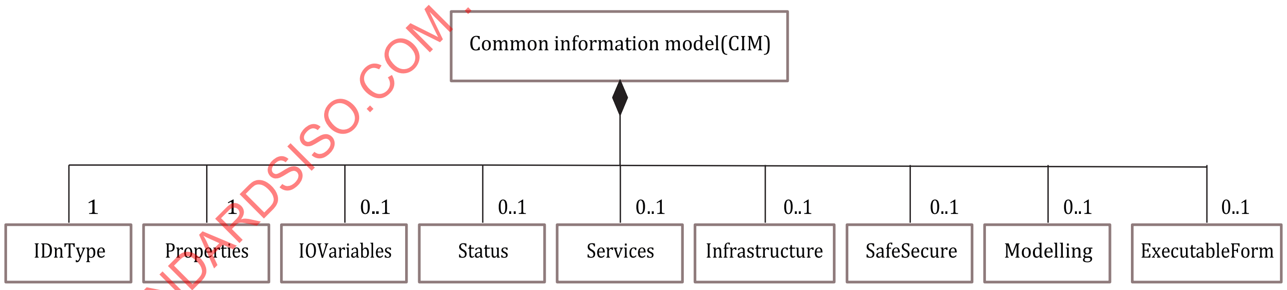


Figure 4.3 — Classes of the common information model

The class CIM (Common Information Model) shall have attributes given in Figure 4.4 and Table 4.5 and its attributes are based on ISO 22166-1:2021, Clauses 4 to 7. Values of attributes such as ModuleName, Description, Manufacturer, and Examples are provided in Annex C. IDnType, Properties, IOVariables, Status, services, infrastructure, SafeSecure, modelling, and ExecutableForm in Figure 4.4 refer to the class described in 4.3.2 to 4.3.10.

NOTE 1 Attributes can be declared as one of the following: private (-), protected (*), or public (+). The name of an attribute is given first and the data type of the attribute is defined next. The separation symbol between an attribute's name and its data type is a colon (:). If attributes are declared as public, it is not necessary to define the functions to access those attributes.

NOTE 2 The four items moduleName, description, manufacturer, and examples are attributes of class CIM.

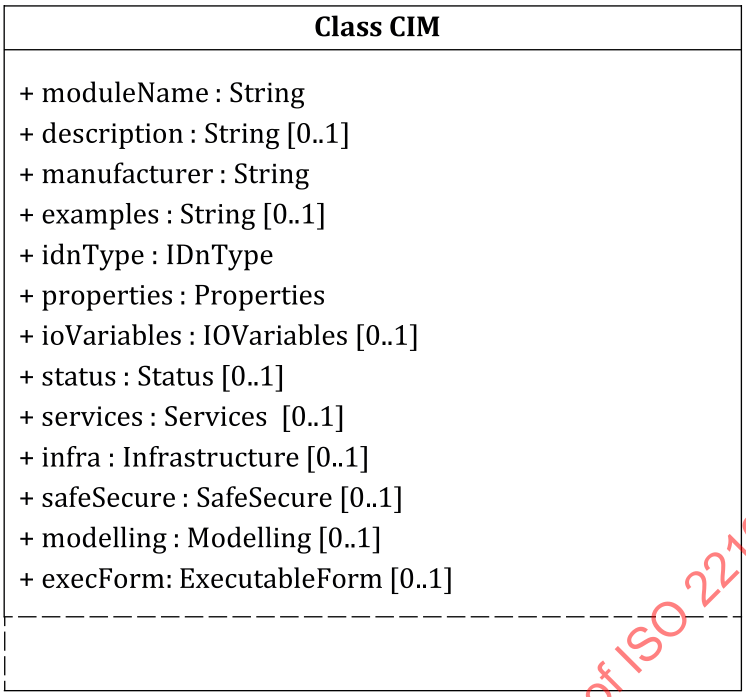

Class CIM
+ moduleName : String
+ description : String [0..1]
+ manufacturer : String
+ examples : String [0..1]
+ idnType : IDnType
+ properties : Properties
+ ioVariables : IOVariables [0..1]
+ status : Status [0..1]
+ services : Services [0..1]
+ infra : Infrastructure [0..1]
+ safeSecure : SafeSecure [0..1]
+ modelling : Modelling [0..1]
+ execForm: ExecutableForm [0..1]

Figure 4.4 — Class diagram of class CIM

Table 4.5 — Class CIM

Description: The root class for the CIM (Common Information Model).				
Derived From: N/A				
Attributes:				
moduleName	String	M	1	The name representing the module
description	String	O	1	The overview of the module, what the module is, what it does, and how it is used.
manufacturer	String	M	1	The contact information for the designer, developer, or manufacturer of the module
examples	String	O	1	Typical use cases of the module
idnType	IDnType	M	1	The module ID includes a unique identifier, instance ID, and owned software/hardware aspects as their IDs. See the details in 4.3.2.
properties	Properties	M	1	A list of configurable parameters of the module. See the details in 4.3.3.
ioVariables	IOVariables	O	1	Information for Input and Output Variables of a module. See the details in 4.3.4
status	Status	O	1	Information for the status and health condition of a module. See the details in 4.3.5
services	Services	O	1	The interfaces that a module provides and utilizes for robot services. See the details in 4.3.6.
infra	Infrastructure	O	1	Information for the infrastructure of a module includes Electric Power, Data Bus, Database, and Ingress Protection. See the details in 4.3.7

Table 4.5 *(continued)*

safeSecure	SafeSecure	O	1	The safety level and the security level that the module provides. See the details in 4.3.8.
modelling	Modelling	O	1	The modelling-related information that the module supports for simulation. See the details in 4.3.9.
execForm	ExecutableForm	O	1	The program-related information executed to achieve or support the purpose of the module. See the details in 4.3.10.

4.3.2 Class for module ID

Information for a Module ID shall be defined in class IDnType which shall have attributes given in Figure 4.5 and Table 4.6. Values of the attribute "moduleID" and other attributes in Table 4.6 are provided via Annex B and Annex C. When the same type of modules are used in a composite module, the modules shall be identified individually. For this purpose, an Instance ID (or IID) is needed. An IID for a software module is dynamically assigned by the module manager. An IID for a module with hardware aspects is statically assigned by the module manufacturer. An IID shall be unique if it is used. The information model version is the version number of the information model used when the module is specified. The information model version is upgraded whenever attributes (including methods) of the classes presented in this document are updated.

NOTE 1 A module with hardware aspects is a hardware module or a module with hardware and software aspects.

The attributes "hwAspects" and "swAspects" are lists of module IDs for hardware aspects and software aspects, respectively.

On the instantiation of this class, the instanceID shall be set using its moduleID. If a module is added to a composite module, the composite module shall check whether or not its moduleID is duplicated in the composite module. If duplicated (or the same module exists), the module manager/composite module manufacturer shall assign a new instance ID to the module.

NOTE 2 The Module ID is provided by the manufacturer of a module.

NOTE 3 The moduleID including IID is used in order to access the module. The default IID is "0".

NOTE 4 The informationModelVersion is "1.0" when the information model defined in this document is used.

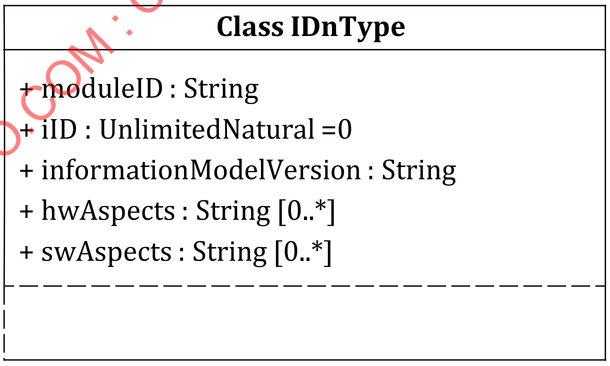

Class IDnType
+ moduleID : String
+ iID : UnlimitedNatural =0
+ informationModelVersion : String
+ hwAspects : String [0..*]
+ swAspects : String [0..*]

Figure 4.5 — Class diagram of class IdnType

Table 4.6 — Class IDnType

Description: Information for Module ID shall be defined in class IDnType.				
Derived From: N/A				
Attributes:				
moduleID	String	M	1	See Annex B for the moduleID format
iID	UnlimitedNatural	M	1	Instance ID, default value = 0.

Table 4.6 *(continued)*

informationModelVersion	String	M	1	Version number of the information model
hwAspects	String	O	N	List of module IDs for hardware modules and modules with hardware aspects and software aspects
swAspects	String	O	N	List of module IDs for software modules

4.3.3 Class for properties

Information for properties of a module shall be defined in class Properties which shall provide the properties of a module as in Figure 4.6 and Tables 4.7 to 4.9, which include information necessary for the execution of the module and information about the operating environment or condition of the module. Figure 4.6 is the class diagram for class Properties and shows the relationship between classes given in Table 4.7 to Table 4.9. These detailed attributes such as the member name and the related data type shall be defined using values of the tag "Property" in Annex C.

The following examples are given in XML and have to be transformed as in Figure 4.6:

```
<!-- example for hardware-software module -->
<Property name="maxRatedCurrent" value="15" type="float32" unit="ampere" description =
"maximum of rated current for motor" />
<Property name="angleResolution" value="1" type="float32" unit="degree" description =
"Angle Resolution of Lidar Sensor" />
<Property name="minOpRange" value="0.05" type="float32" unit="meter" description =
"minimum operation range of Lidar Sensor" />
<Property name="maxOpRange" value="10" type="float32" unit="meter" description = "maximum
operation range of Lidar Sensor" />
<Property name="commModule" value="Ethernet microUSB" type="array of string" unit="null"
description = "communication protocol type of Lidar Sensor" />
<!-- example for software module -->
<Property name="Weight" value="150" type="float32" unit="gram" description = "weight of
Lidar Sensor" />  <-- example for hardware-software module -->
<Property name="maxRadius" value="2" type="float32" unit="meter" description = "maximum
radius for Obstacle Avoidance" />
<Property name="minRadius" value="0.5" type="float32" unit="meter" description = "minimum
radius for Obstacle Avoidance" />
<!-- example for hardware module -->
<Property name="origin" type="array of float32" unit="cm", "radian" description = "pose
of the inertial reference frame, relative to the link reference frame including xyz and
rpy" value="[0 0 0 0 0 0]" />
<Property name="mass" type="float32" unit="gram" description = "mass of a link" value =15
/>
<Property name="inertia" type="array of float32" unit="g-cm2" description = "inertial
properties of the link, 3x3 matrix" value ="[1 0 0; 0 1 0; 0 0 1]" />
<Property name="shape" type="string" unit="" description = "shape of the visual object"
value =cylinder />
<Property name="size" type="3x1 array of float32" unit="cm" description = "three side
lengths of the box" value ="[100; 200; 100]" />
```

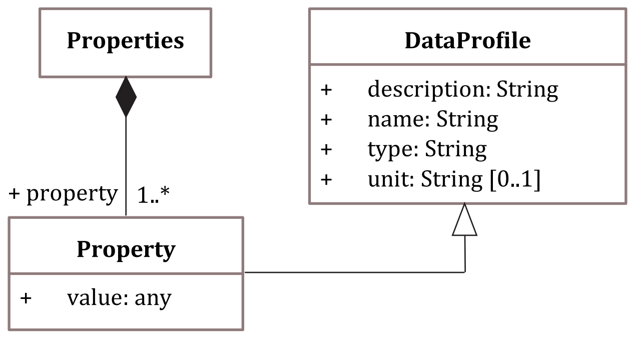


Figure 4.6 — Class diagram for class Properties

Table 4.7 — Class DataProfile

Description: Class for data profiles				
Derived From: N/A				
Attributes:				
description	String	M	1	Explanation of the property
name	String	M	1	Name of the property
type	String	M	1	Data type of the property
unit	String	O	1	Unit of the property. If no unit, unit is null.

Table 4.8 — Class Property

Description: Class for a property				
Derived From: DataProfile				
Attributes:				
value	any	M	1	Value of the property

Table 4.9 — Class Properties

Description: Defines information for properties of a module				
Derived From: N/A				
Attributes:				
property	Property	M	N	List of attributes generated according to the class Property

4.3.4 Class for input and output variables

Information for Input and Output Variables of a module shall be defined in the class IOVariables as in Figure 4.7 and Tables 4.10 to 4.15. The class IOVariables shall provide variables used within the module, which include information necessary for the execution of the module. These variables shall be defined using the class Variable specified in Table 4.12. The class Variable inherits from the class VariableProfile in Table 4.11, which, in turn, inherits from the class DataProfile in Table 4.7. For storing additional information, the class NVList in Table 4.14 can be utilized as a container. The NVList is a list of NameValue objects, where the class NameValue is defined in Table 4.15. The values IN, OUT, and INOUT are defined as enumeration data types in Table 4.10. The detailed attributes such as the member name and the related data type shall be defined using values of the tag IOVariables as in Annex C. Figure 4.7 shows the relationships between classes for the class IOVariables.

The following examples are given in XML and have to be transformed to the form of Figure 4.12:

```
 <IOVariables>
     <Input name="controlValue" type="float32" unit="ampere" description="current control to
motor" /> <!-- IOType=IN -->
```

```
 <Output name="encoder" value= 0 type ="uint32" unit = "none" description = "absolute
encoder's value" /> <!-- IOType=OUT -->
     <Inout name="state" type="uint8" value= 0 unit = "none" description = "status of motor"/>
<!-- IOType=INOUT -->
 </IOVariables>
```

From these tags, the following member variables and their data types are generated.

+ controlValue : float32 // or public float32 controlValue; current control command to motor, input

+ encoder=0 : uint32 // or public uint32 encoder=0; value of absolute encoder, output

+ state=0 : uint8 // or public uint8 state=0; reset or read status of motor, input and output

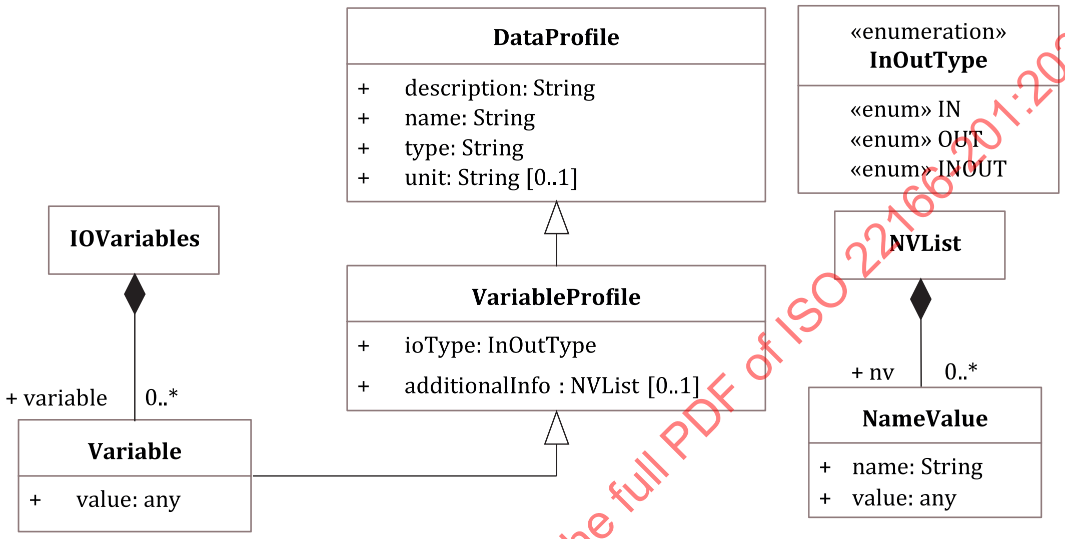


Figure 4.7 — Relationships between classes for class IOVariables

Table 4.10 — Enumeration InOutType

Description: Types of input and output variables	
Attributes:	
IN	Used for variables to which external modules write data
OUT	Used for variables from which external modules reads data
INOUT	Used for variables which external modules read from or write data to.

Table 4.11 — Class VariableProfile

Description: Class for a profile of variables				
Derived From: DataProfile (see Table 4.7)				
Attributes:				
ioType	InOutType	M	1	One of {IN, OUT, INOUT}. See Table 4.10
additionalInfo	NVList	O	1	The class NVList serves as a container for additional information across various classes. See Table 4.14

Table 4.12 — Class Variable

Description: Class for a variable				
Derived From: VariableProfile (see Table 4.11)				
Attributes:				
value	any	M	1	Value of the variable

Table 4.13 — Class IOVariables

Description: Class for input and output variables of a module				
Derived From: N/A				
Attributes:				
variable	Variable	O	N	List of attributes generated according to the tag Variable

Table 4.14 — Class NVList

Description: A list for NameValue for additional information across various classes.				
Derived From: N/A				
Attributes:				
nv	NameValue	O	N	A list of NameValue. See Table 4.15

Table 4.15 — Class NameValue

Description: A list for NameValue for additional information across various classes.				
Derived From: N/A				
Attributes:				
name	String	M	1	Name of additional information
value	any	M	1	Value of additional information

4.3.5 Class for status

Information for the status and health condition of a module shall be defined in class Status as in Figure 4.8 and Table 4.17. Class Status provides the status of a module which represents the health condition of the module if the hardware information model exists, the status of the execution life cycle if the software information model exists, and the error type that may have occurred in operation.

Error numbers shall use those defined in IEEE/Open Group 1003.1-2017. Additional error numbers may be defined for each software module, which shall not conflict with error numbers in POSIX.1003.1.

NOTE 1 Examples of HealthCond are as follows: GH (good health), H1 (Health1), H2 (Health2), and Fa (Failure).

NOTE 2 Examples of errors used in ErrorType are as follows: Operation not permitted, Authentication Error, Bad Parameter, Unsupported Service, out of Range, and Precondition not met. Values of ErrorType will be defined in detail in ISO 22166-202 and other documents.

NOTE 3 ExeStatus is defined in ISO 22166-1 and shall be one of following: CREATED, IDLE, EXECUTING, DESTRUCTED, and ERROR. These values are defined as enumeration data types in Table 4.16.

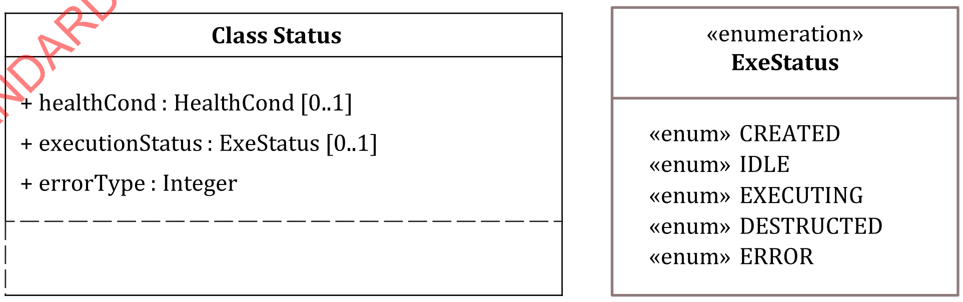


Figure 4.8 — Class diagram for class Status

Table 4.16 — Enumeration ExeStatus

Description: Defines types of execution status	
Attributes:	
CREATED	Created State
IDLE	Idle sate
EXECUTING	Executing State
DESTRUCTED	Destroy state
ERROR	Error state

Table 4.17 — Class Status

Description: Class for the status of a module				
Derived From: N/A				
Attributes:				
healthCond	HealthCond	O	1	In some modules, HealthCond has no values. This enumeration type is as follows: {GH, H1, H2, Fa} where GH, H1, H2, and Fa mean good health, Health type 1, Health type 2, and Failure.
executionStatus	ExeStatus	O	1	In some modules, ExeStatus can get no values (see Table 4.16)
errorType	Integer	M	1	Depending on module type. See NOTE 2.

4.3.6 Class for services

Information for Services of a Module shall be defined in class Services which provides the methods (or functions) that the module supports, as in Figure 4.9 and Table 4.18. Attributes and methods in Table 4.18 shall be obtained from CIMServicePackage, defined in IDL, as provided below or obtained from classes defined in Tables 4.19 to 4.24. Figure 4.9 shows the relationships between classes for class Services, which are provided in Tables 4.18 - 4.24.

NOTE Services in this subclause are defined using ISO/IEC 19516.

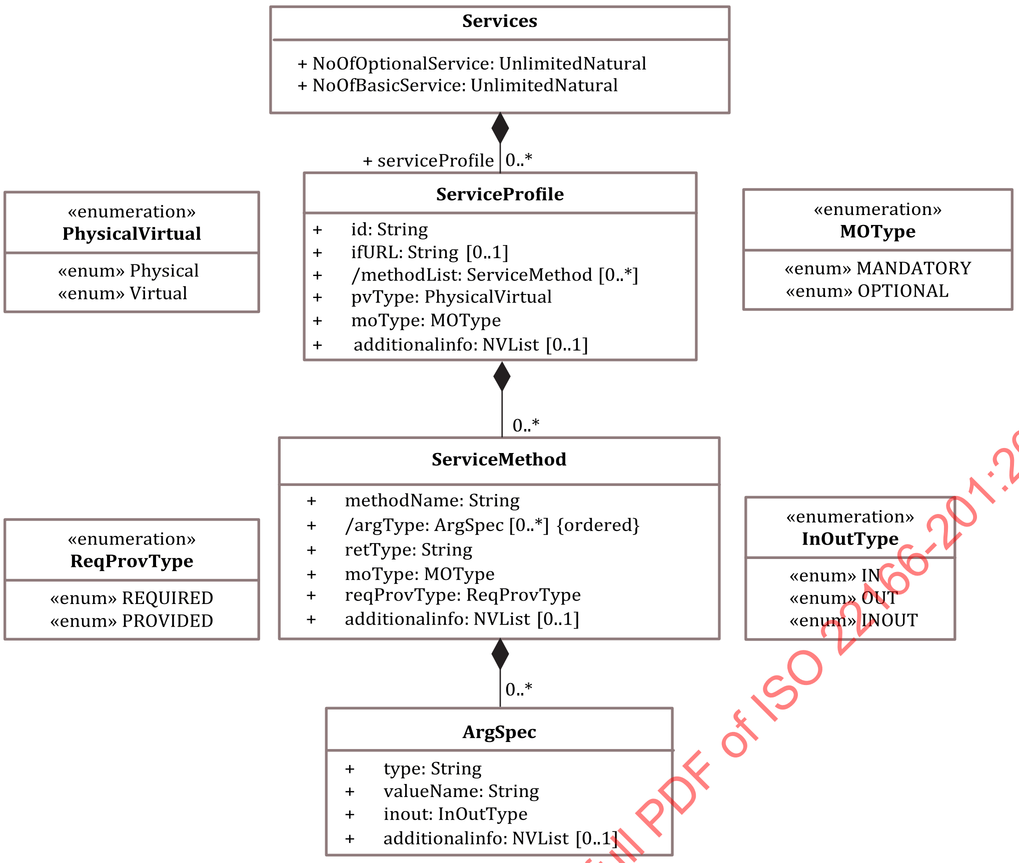


Figure 4.9 — Relationships between classes for class Services

```
CIMServicePackage written in IDL:

Module CIMServicePackage {
 interface CIMService;    // abstract declaration
 struct NameValue
 {
  string name;
  any value;   // "any" means any data type
 };
 // Mandatory/Optional enum type
 enum MOType
 {
       MANDATORY,
       OPTIONAL
 };
 struct ArgSpec {
     string type;             // argument type
     string valueName;        // argument name
     InOut inout;            // enumeration {IN, OUT, INOUT}
     NVList additionalInfo;    // additional information
  } ;
  typedef sequence<ArgSpec> ArgSpecList;  //sequence means a list or vector of
                                          // given data
  struct ServiceMethod
  {
     string methodName;
     ArgSpecList argType;
     string retType;
     MOType moType;         // enumeration {"MANDATORY", "OPTIONAL"}
     ReqProvType reqProvType; // Provided or required method.
     NVList additionalInfo;    // additional information
  };
  typedef sequence<NameValue> NVList;
  typedef sequence<ServiceMethod> MethodList;
  enumeration PhysicalVirtual
  {
       PHYSICAL,
       VIRTUAL
  };
  // ServiceProfile
  struct ServiceProfile
  {
    string id;
    PhysicalVirtual pvType;   // Enumeration PHYSICAL, VIRTUAL
    MethodList methodList;  // mandatory/optional method defined in CIMService
    MOTypr moType;        // MOType: {MANDATORY, OPTIONAL}.
    NVList additionalInfo;    //  additional information
  };
  interface CIMService
  {
    // define here for prototypes of methods for CIM Service
    // format: <type_spec | void> <method_identifier> ( <parameter decls> )
    // <parameter decls> ::= <para_dcl>
    // <param_dcl> ::= <"in"|"out"|"inout"> <type_spec> <parameter_name>
    // example:
    //   uint8 initialize(in int16 val1, in float64 val2);
    //   boolean checkElecConnectivity(string connType, string keying,
    //                                string busProtocol)
    //   boolean linkConnectivity(array origin, float mass, array inertia,
    //     string shape, array size,array axis,string connection,array collision)
  };
};
```

Table 4.18 — Class Services

Description: Class for services of a module				
Derived From: N/A				
Attributes:				
NoOfBasicService	UnlimitedNatural	M	1	Number of Basic services provided
NoOfOptionalService	UnlimitedNatural	M	1	Number of Optional services provided
serviceProfile	ServiceProfile	O	N	Prototype for basic (mandatory) Services, serviceProfile can be a list. See Table 4.20.

Table 4.19 — Enumeration PhysicalVirtual

Description: Defines type of interface	
Attributes:	
Physical	Physical interface such as a mechanical or electrical interface
Virtual	Virtual interface such as a software interface

Table 4.20 — Class ServiceProfile

Description: Class for a profile of services				
Derived From: N/A				
Attributes:				
id	String	M	1	Name of service profile. One or more service profiles can be provided.
ifURL	String	O	1	URL/path of file defined in IDL(see CIMServicePackage) for service.
methodList	ServiceMethod	O	N	List of methods for service, defined in XML. See Table 4.23.
Only one type of two types of ifURL and methodList shall be provided. If not provided, its value is NULL.				

Table 4.20 *(continued)*

pvType	PhysicalVirtual	M	1	PhysicalVirtual : enumeration data type {Physical, Virtual} (see Table 4.19). Physical (Mechanical/Elec) Method or Virtual (Software) Method
moType	MOType	M	1	MOType: enumeration data type {MANDATORY, OPTIONAL}. See Table 4.21.
additionalInfo	NVList	O	1	Arguments and their default initialization values for service. See Table 4.14.
Only one type of two types of ifURL and methodList shall be provided. If not provided, its value is NULL.				

Table 4.21 — Enumeration MOType

Description: Define type of mandatory or optional services	
Attributes:	
MANDATORY	Mandatory
OPTIONAL	Optional

Table 4.22 — Enumeration ReqProvType

Description: Define type of required or provided services	
Attributes:	
REQUIRED	Required service.
PROVIDED	Provided service.

Table 4.23 — Class ServiceMethod

Description: Class for a service method				
Derived From: N/A				
Attributes:				
methodName	String	M	1	
argType	ArgSpec	O	N	Ordered list of arguments used in a method. See Table 4.24.
retType	String	M	1	Data type of return value. See Table C.5.
moType	MOType	M	1	Mandatory or optional method. See Table 4.21.
reqProvType	ReqProvType	M	1	Provided or required method. See Table 4.22.
additionalInfo	NVList	O	1	A list for NameValue for additional information. See Table 4.14.

Table 4.24 — Class ArgSpec

Description: Class for specifying an argument				
Derived From: N/A				
Attributes:				
type	String	M	1	Data type of an argument used in a method (see Table C.5)
valueName	String	M	1	Name of an argument used in a method
inout	InOutType	M	1	See Table 4.10.
additionalInfo	NVList	O	1	A list for NameValue for additional information. See Table 4.14.

4.3.7 Class for infrastructure

Information for the infrastructure of a module shall be defined in class Infrastructure which provides the types of infrastructures that the module supports, as shown in Figure 4.10 and Table 4.27. This subclause defines the following infrastructure modules: Electric Power, Data Bus, Database, and Ingress Protection (IP).

NOTE 1 If an additional Infrastructure module exists, the module can be defined in the related information model.

Class DataBus and class Power used in class Infrastructure are shown in Table 4.25 and Table 4.26, respectively.

NOTE 2 Values for typePhyMac and typeApp in Table 4.25, and values for dbType in Table 4.27 are defined in ISO 22166-202.

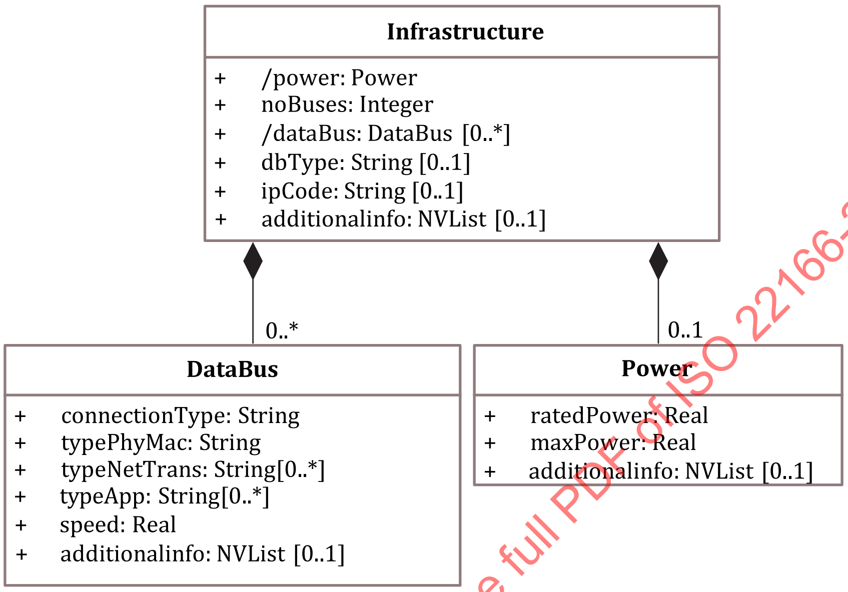


Figure 4.10 — Relationships between classes for class Infrastructure

Table 4.25 — Class DataBus

Description: Class for a data bus				
Derived From: N/A				
Attributes:				
connectionType	String	M	1	Connection type:USB-A, USB-C, USD-C, RG45, DB9, DB15, DB25
typePhyMac	String	M	1	Physical/MAC protocol type: USB, CAN2.0, EtherCAT, Ethernet, RS485
typeNetTrans	String	O	N	Network Protocol and Transport Protocol, IP, TCP, CANopen
typeApp	String	O	N	Session/presentation/application protocol: list of protocols
speed	Real	M	1	Transmission speed, unit: Kbps
additionalInfo	NVList	O	1	A list for NameValue for additional information, See Table 4.14.

Table 4.26 — Class Power

Description: Class for power				
Derived From: N/A				
Attributes:				
ratedPower	Real	M	1	Rated Power in watts
maxPower	Real	M	1	Maximum Power in watts
additionalInfo	NVList	O	1	A list for NameValue for additional information. See Table 4.14.

Table 4.27 — Class Infrastructure

Description: Class for infrastructure				
Derived From: N/A				
Attributes:				
power	Power	0	1	See Table 4.26,
noBuses	Integer	0	1	The number of databuses used in the module.
dataBus	DataBus	0	N	Data buses (or communication protocols) that the module provides. If noBuses is greater than 0, this field is mandatory. See Table 4.25.
dbType	String	0	1	Types of database mgt. system. DBMS type used in a module (e.g. Oracle, SQL, etc.).
ipCode	String	0	1	IP code. Ingress Protection IEC 60529 Ed.2.2 b: 2013; two digits;
additionalInfo	NVList	0	1	A list for NameValue for additional information. See Table 4.14.

4.3.8 Class for safety and security

The class SafeSecure shall present the safety level and the security level that the module provides, which are in Figure 4.11 and Table 4.38. Figure 4.11 shows the relationships between classes for class SafeSecure, which are provided in Tables 4.28 to 4.38.

Measures for the individual safety function shall be specified using the safety function type and the performance level (none, a to e) or the safety integrity level (0, 1 to 3) for the safety function type. A module can include zero or more safety functions. The safety function shall be defined using class SafetyFunction as in Table 4.34. Various types of safety functions are shown in Table 4.28. The safety level for safety functions shall be provided using SIL or PL. The security level shall be provided for security functions.

Measures for the individual cyber security shall be specified using the security type and the security level (0 to 4) for the type. The various types of security measures are provided in Table 4.29. A module can include zero or more security measures. The measure for individual cyber security shall be defined using class CyberSecurity as in Table 4.37. If the module provides two or more security measures, the module shall provide the overall security level for the module using the combination of the security levels of the provided security measures of the module or via the verification and validation of the module's overall security measures.

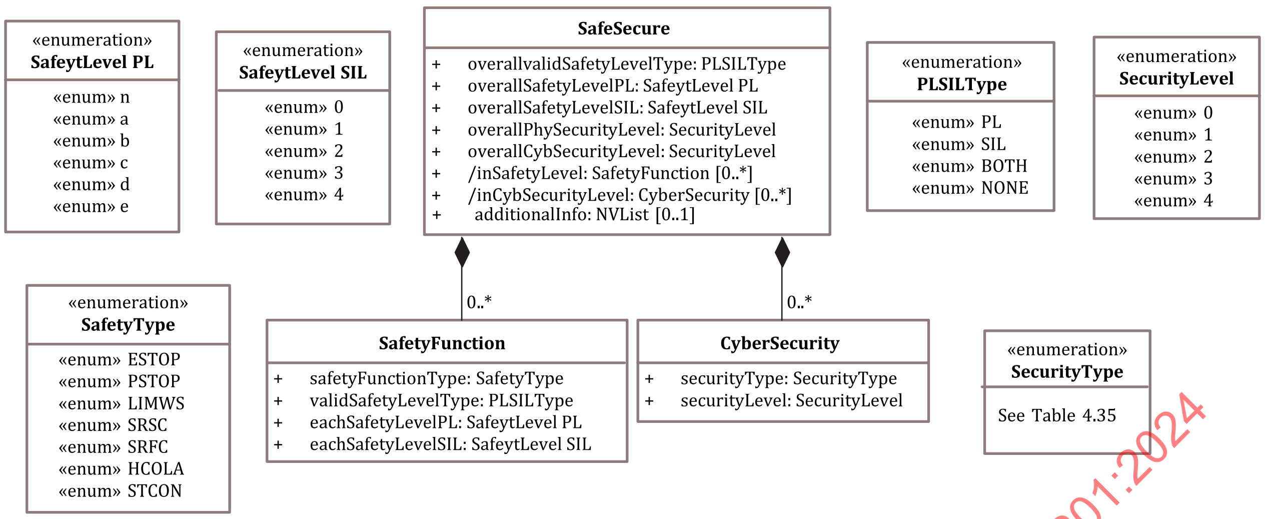
The safety level of a module is only provided as information when assessing the safety of the system. Even though the safety level for the safety function of a module is given, a composite module shall be independently evaluated for the safety function of itself according to system-level safety evaluation methods.

NOTE 1 The safety level for safety function is based on ISO 13482:2014, and the security level for security function is based on IEC 62443-4-2:2019.

NOTE 2 Values for attributes in Table 4.34 and Table 4.37 are read-only, provided by the manufacturer.

NOTE 3 The performance level for no safety function is indicated by "none" or "n".

NOTE 4 The safety-related functions and/or the security-related functions are provided in class Services, which can be used for safety/security management.



«enumeration» SafeytLevel PL
«enum» n
«enum» a
«enum» b
«enum» c
«enum» d
«enum» e
«enumeration» SafeytLevel SIL
«enum» 0
«enum» 1
«enum» 2
«enum» 3
«enum» 4
SafeSecure
+ overallvalidSafetyLevelType: PLSILType
+ overallSafetyLevelPL: SafeytLevel PL
+ overallSafetyLevelSIL: SafeytLevel SIL
+ overallPhySecurityLevel: SecurityLevel
+ overallCybSecurityLevel: SecurityLevel
+ /inSafetyLevel: SafetyFunction [0..*]
+ /inCybSecurityLevel: CyberSecurity [0..*]
+ additionalInfo: NVList [0..1]
«enumeration» PLSILType
«enum» PL
«enum» SIL
«enum» BOTH
«enum» NONE
«enumeration» SecurityLevel
«enum» 0
«enum» 1
«enum» 2
«enum» 3
«enum» 4
0..*
0..*
«enumeration» SafetyType
«enum» ESTOP
«enum» PSTOP
«enum» LIMWS
«enum» SRSC
«enum» SRFC
«enum» HCOLA
«enum» STCON
SafetyFunction
+ safetyFunctionType: SafetyType
+ validSafetyLevelType: PLSILType
+ eachSafetyLevelPL: SafeytLevel PL
+ eachSafetyLevelSIL: SafeytLevel SIL
CyberSecurity
+ securityType: SecurityType
+ securityLevel: SecurityLevel
«enumeration» SecurityType
See Table 4.35

Figure 4.11 — Relationships between classes for class SafeSecure

Table 4.28 — Types of safety functions and their tags

Safety Functions (ISO 13482:2014)	Tag Type
Emergency stop	ESTOP
Protective stop	PSTOP
Limits to workspace (including forbidden area avoidance)	LIMWS
Safety-related speed control	SRSC
Safety-related force control	SRFC
Hazardous collision avoidance	HCOLA
Stability control (including overload protection)	STCON

NOTE 5 Safety functions not listed in Table 4.28 will be added as needed.

Table 4.29 — Types of measures for cyber security and their tags

Security Measure Type (IEC 62443-4-2:2019)	Tag Type
Human user identification and authentication	HU_IA
Software process and device identification and authentication	SD_IA
Account management	ACNT_MGT
Identifier management	ID_MGT
Authenticator management	AUTH_MGT
Wireless access management	WIRELEE_MGT
Strength of password-based authentication	PW_AUTH
Public key infrastructure certification	PK_CERT
Strength of public key-based authentication	STR_PK_AUTH
Unsuccessful login attempts	LOGIN_NO
Access via untrusted networks	ACC_UNTRUST_NET
Authorization enforcement	AUTHORIZE
Wireless use control	WIRELESS_USE
Session lock	SESS_LOCK
Remote session termination	SESS_TERM
Concurrent session control	SECC_CNTR

Table 4.29 *(continued)*

Security Measure Type (IEC 62443-4-2:2019)	Tag Type
Auditable events	AUDT_EVT
Timestamps	TIMESTM
Non-repudiation	NON_REP
Communication integrity	COMM_INTG
Protection from malicious code	PROT_MALI_CODE
Security functionality verification	SECUR_VERIFY
Software and information integrity	SW_INTGT
Input validation	INPUT_VALD
Deterministic output	DET_OUT
Error handling	ERR_HNDL
Session integrity	SESS_ INTGT
Information confidentiality	INFO_CONFI
Information persistence	INFO_PERS
Use of cryptography	CRYTO
Restricted data flow	RSTIC_FLOW
Denial of service protection	DoS
Resource management	RESOU_MGT
Control system recovery and reconstitution	CNTR_RECOV_RECON

Table 4.30 — Enumeration SafetyType

Description: Safety type of a module. See Tables 4.28 and 4.34.	
Attributes:	
ESTOP	Emergency stop
PSTOP	Protective stop
LIMWS	Limits to workspace (including forbidden area avoidance)
SRSC	Safety-related speed control
SRFC	Safety-related force control
HCOLA	Hazardous collision avoidance
STCON	Stability control (including overload protection)

Table 4.31 — Enumeration SafetyLevelPL

Description: Present Performance level.	
Attributes:	
n	None. No performance level.
a	PL a
b	PL b
c	PL c
d	PL d
e	PL e

Table 4.32 — Enumeration SafetyLevelSIL

Description: Safety Integrity Level	
Attributes:	
0	None. No SIL.
1	SIL 1
2	SIL 2
3	SIL 3
4	SIL 4

Table 4.33 — Enumeration PLSILType

Description: Type of safety representation	
Attributes:	
PL	Safety level represented in PL type
SIL	Safety level represented in SIL type
BOTH	Safety level represented in both SIL and PL types
NONE	Safety level represented in neither SIL nor PL types

Table 4.34 — Class SafetyFunction

Description: Safety function of a module				
Derived From: N/A				
Attributes:				
safetyFunctionType	SafetyType	M	1	SafetyType. See Table 4.30.
validSafetyLevelType	PLSILType	M	1	Providing Safety Level Type. See Table 4.33.
eachSafetyLevelPL	SafetyLevelPL	M	1	See Table 4.31. n: No safety function
eachSafetyLevelSIL	SafetyLevelSIL	M	1	See Table 4.32. 0: No safety function

Table 4.35 — Enumeration SecurityType

Description: Security type of a module. See Table 4.29.	
Attributes:	
HU_IA	Human user identification and authentication
SD_IA	Software process and device identification and authentication
ACNT_MGT	Account management
ID_MGT	Identifier management
AUTH_MGT	Authenticator management
WIRELEE_MGT	Wireless access management
PW_AUTH	Strength of password-based authentication
PK_CERT	Public key infrastructure certification
STR_PK_AUTH	Strength of public key-based authentication
LOGIN_NO	Unsuccessful login attempts
ACC_UNTRUST_NET	Access via untrusted networks
AUTHORIZE	Authorization enforcement
WIRELESS_USE	Wireless use control
SESS_LOCK	Session lock
SESS_TERM	Remote session termination
SECC_CNTR	Concurrent session control

Table 4.35 *(continued)*

AUDT_EVT	Auditable events
TIMESTM	Timestamps
NON_REP	Non-repudiation
COMM_INTG	Communication integrity
PROT_MALI_CODE	Protection from malicious code
SECUR_VERIFY	Security functionality verification
SW_INTGT	Software and information integrity
INPUT_VALD	Input validation
DET_OUT	Deterministic output
ERR_HNDL	Error handling
SESS_ INTGT	Session integrity
INFO_CONFI	Information confidentiality
INFO_PERS	Information persistence
CRYTO	Use of cryptography
RSTIC_FLOW	Restricted data flow
DoS	Denial of service protection
RESOU_MGT	Resource management
CNTR_RECOV_RECON	Control system recovery and reconstitution

Table 4.36 — Enumeration SecurityLevel

Description: Present security level		
Attributes:		
	Cyber Security	Physical Security
0	None. No security level.	None. No security level.
1	Values defined in IEC 62443-4-2:2019.	LatchSensor
2		LockwithKey
3		LockwithActuator
4		Not defined

Table 4.37 — Class CyberSecurity

Description: Class for cyber security				
Derived From: N/A				
Attributes:				
securityType	SecurityType	M	1	See Table 4.35 and Table 4.29.
eachSecurityLevel	SecurityLevel	M	1	See Table 4.36.

Table 4.38 — Class SafeSecure

Description: Class for safety and security of a module. If a module provides a single safety function, its value equals the value of eachSafetyLevel of the safety function. At least one of these two values should be provided.				
Derived From: N/A				
Attributes:				
overallValidSafetyLevelType	PLSILType	M	1	Representation mode of overall safety Level (see Table 4.33)
overallSafetyLevelPL	SafetyLevelPL	M	1	Set PL if overallValidSafetyLevelType is PL or Both (see Table 4.31)

Table 4.38 *(continued)*

overallSafetyLevelSIL	SafetyLevelSIL	M	1	Set SIL if overallValidSafetyLevelType is SIL or Both (see Table 4.32)
overallPhySecurityLevel	SecurityLevel	M	1	Overall Physical Security of a module. See Table 4.36.
overallCybSecurityLevel	SecurityLevel	M	1	Overall Cyber Security of a module. See Table 4.37.
inSafetyLevel	SafetyFunction	O	N	PL of individual safety function. See Table 4.34.
inCybSecurityLevel	CyberSecurity	O	N	Individual security function. See Table 4.37.
additionalInfo	NVList	O	1	A list for NameValue for additional information. See Table 4.14.

4.3.9 Class for modelling

The class Modelling shall provide modelling related information that the module supports for simulation, as in Figure 4.12, Table 4.39 and Table 4.40. Figure 4.12 shows the relationship between classes for class Modelling.

Information for modelling shall provide information that the module provides for simulation, which is classified into information for geometric simulation and information for dynamical simulation or behaviour with physical and/or logical properties. Examples of the former are 3D models and Universal Robotic Description Format (URDF) files. Examples of the latter are driving and/or control programs for the module. This information shall be provided as in Annex C. Module manufacturers can provide more than one file for simulation. If there are no URDF files and/or 3D model files, the corresponding item shall be marked "N/A" (not available).

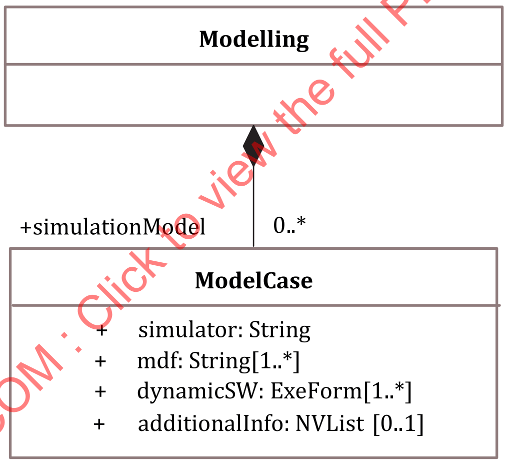


Figure 4.12 — Relationship between classes for class Modelling

Table 4.39 — Class ModelCase

Description: Used to provide paths to files that provide a model for simulation of a module				
Derived From: N/A				
Attributes:				
simulator	String	M	1	List of simulation programs that a module supports
mdf	String	M	N	Path of model description file or path of directory including the model description files, or "N/A". A model description file can include a 3D model.
dynamicSW	ExeForm	M	N	Dynamic programs and/or control programs for the module related simulation
additionalInfo	NVList	O	1	A list for NameValue for additional information. See Table 4.14.

Table 4.40 — Class Modelling

Description: Used when providing for multiple simulators, but a 3D model can be used on only one simulator.				
Derived From: N/A				
Attributes:				
simulationModel	ModelCase	O	N	Provide information of simulation models used in a module. See Table 4.39.

4.3.10 Class for executable form

Class ExecutableForm shall provide the program-related information executed to achieve or support the purpose of the module, as in Figure 4.13 and Table 4.41. Figure 4.13 shows the relationship between classes for class ExecutableForm, which is provided in Tables 4.41 and 4.42. Class ExecutableForm provides paths for the files necessary when a module is executed or other modules execute the program of the module. Those files are classified into the following 2 types: files that need to be executed directly, which are provided by class ExeForm in Table 4.42, and library files that load into memory. The path for files shall be given in URL format. The URL shall be provided only via a profile, which means that user can modify URL by changing the content of the profile.

Class ExeForm provides information needed when an executable form of a module is executed, which consists of a URL for the executable form, shell command, and properties. The shell command provides information necessary as input arguments when the related executable form is executed. The exeProp provides information necessary while the related executable form is being executed.

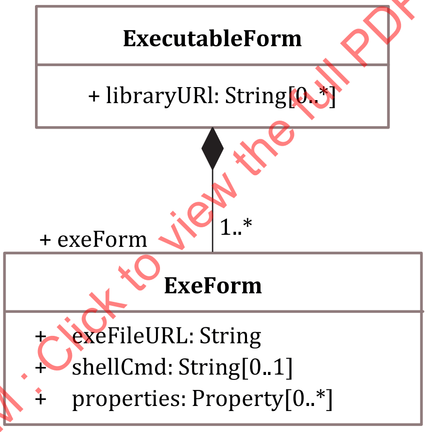


Figure 4.13 — Relationship between classes for class ExecutableForm

Table 4.41 — Class ExecuatbleForm

Description: Provides the program related information executed to achieve or support the purpose of the module. The properties, such as the OS, which are defined in ISO 22166-202, have to be provided.				
Derived From: N/A				
Attributes:				
exeForm	ExeForm	M	N	See Table 4.41.
libraryURL	String	O	N	Binary code or source code used in the module. Source code requires its interpreter. If a library is not needed, this value shall be NULL.

Table 4.42 — Class ExeForm

Description: Provides the program related information that shall be executed for a module				
Derived From: N/A				
Attributes:				
exeFileURL	String	M	1	Path of executable file that shall be executed.
shellCmd	String	O	1	Command line to execute the "exeFileURL" given as information of the module. This command line is necessary if the program "exeFileURL" utilizes any input in the command line. If not needed, this value shall be NULL.
properties	Property	O	N	See Table 4.8. Properties and their values needed while the program "exeFileURL" is being executed. If not needed, this value shall be NULL.

Annex A
(normative)

Naming rules

A.1 General

The naming rules are derived from the guidelines and principles described in document ISO 11179-5:2015. These rules have been tailored to the specific usage of the information model. They are designed to ensure consistent and appropriate naming of both basic and aggregated information entities.

A basic information entity is used in providing information and building information models, and examples include object class, attribute of object class, tag term, and property term. It is defined as an entry Name, which is the name of the entity derived from naming rules.

An aggregated information entity is defined as a name composed of entry names of the basic information entities.

A.2 Naming rules

Rule 1: An entry name shall be unique in the logical data grouping to which the entry belongs.

Rule 2: A class name may start with a capital letter. A member name of a class should be start with a small letter.

Rule 3: An entry name shall not contain consecutive redundant words.

Rule 4: It is recommended that an entry name be less than 10 characters, with the maximum length of less than 256. An abbreviation for the entry name should be defined and used, if necessary.

Rule 5: An entry name shall be separated by a period (.).

Rule 6: Alphanumeric characters may be used in an entry name if not defined in particular.

Rule 7: An entry name should be in singular form unless the meaning itself is plural (e.g. goods).

Rule 8: An entry name may consist of two or more words (e.g. aggregate.information).

Some examples of applying the naming rules are shown in Figure A.1.

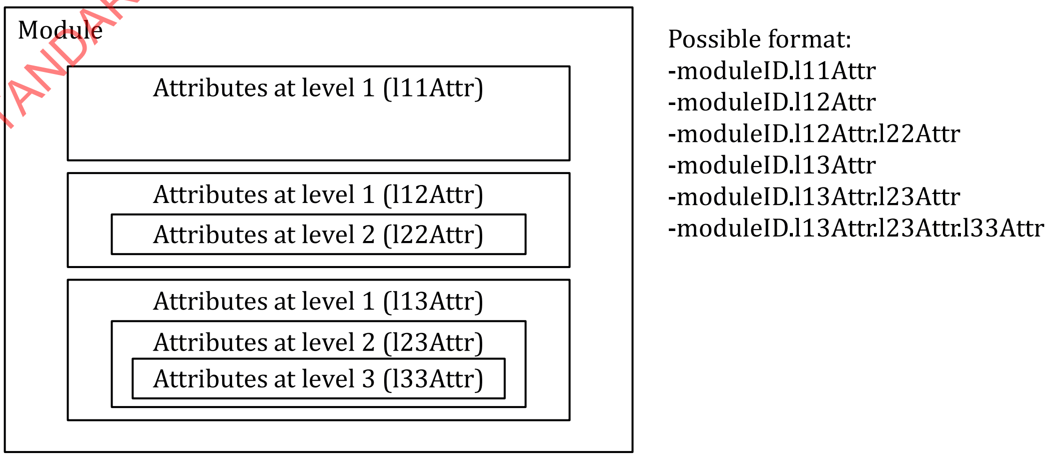

Module
Attributes at level 1 (l11Attr)
Attributes at level 1 (l12Attr)
Attributes at level 2 (l22Attr)
Attributes at level 1 (l13Attr)
Attributes at level 2 (l23Attr)
Attributes at level 3 (l33Attr)
Possible format:
-moduleID.l11Attr
-moduleID.l12Attr
-moduleID.l12Attr.l22Attr
-moduleID.l13Attr
-moduleID.l13Attr.l23Attr
-moduleID.l13Attr.l23Attr.l33Attr

Figure A.1 — Relationships of a module and attributes in each level

Annex B
(normative)

Assignment rule of a module ID

A module ID shall consist of 6 elements, which are illustrated in Figure B.1, and their types are given in Table B.1.

The first element is the Vendor ID or VID which is 16 bytes long and shall be assigned by the Universally Unique Identifier (UUID) IETF RFC 4122. The remaining 5 elements shall be locally assigned by the vendor.

The second element is the product ID or PID which is 4 bytes long and shall consist of important information about the module of 1 byte and a product ID of 3 bytes. The former (1 byte) shall provide the following 5 types of information as in Figure B.1: whether the module is composite or basic, whether software aspects exist in the module, whether hardware aspects exist in the module, whether the module provides a safety-related function, and whether the module provides a security-related function. The values of each field are given in the Table B.2. The latter (3 bytes) shall be the actual product ID, which is represented by alphanumeric characters.

The third element shall be the revision number or Rev, which is 4 bytes long.

The fourth element shall be the serial number (SerialNO), which is 4 bytes long.

The fifth element shall be the category ID (CategoryID) of a module, which is 3 bytes long.

The sixth element shall be the instance ID, (IID), which is 1 byte long and is represented by numeric characters with 0-255. The default value of instance ID is 0. If a module has two or more modules of the same type, the modules shall have different instance IDs.

All data in the module ID are represented in hexadecimal code and/or alpha-numeric code.

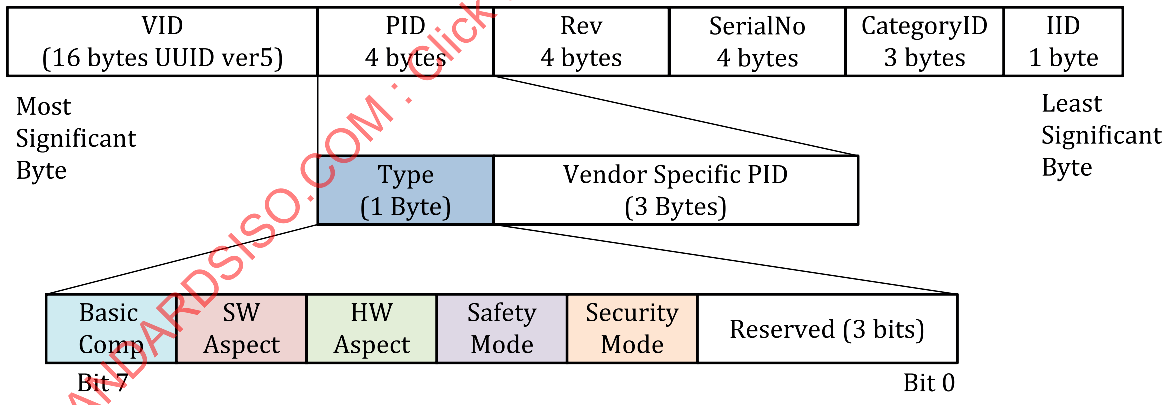


Figure B.1 — Composition of a module ID

Table B.1 — Data types of the elements of a module ID

Name	Data Type	Length (bytes)	Note
VID	UUID	16	Vendor ID
PID	unsigned char	4	Product ID and properties
Rev	unsigned char	4	Revision number
SerialNo	unsigned char	4	Serial number
CategoryID	unsigned char	3	Category ID
IID	unsigned char	1	Instance ID

Table B.2 — Value of each field of Type in Figure B.1

Field name of Type	Value	location in Byte
Basic Comp	0 if a module is a kind of basic module 1 if a module is a kind of composite module	Bit 7 (MSB)
SW Aspect	1 if SW aspects exist 0 if not	Bit 6
HW Aspect	1 if HW aspects exist 0 if not	Bit 5
Safety Mode	1 if a module provides the safety-related function 0 if not	Bit 4
Security Mode	1 if a module provides the security-related function 0 if not	Bit 3
Reserved	-	Bit 2 - 0

CategoryID shall consist of 5 elements, which are illustrated in Figure B.2. As in Figure B.2, the 0-th level is the top level and classified into 4 categories as in Table B.3. The categories from the 1-st level to the 4-th level are provided in upcoming standards such as ISO 22166-202.

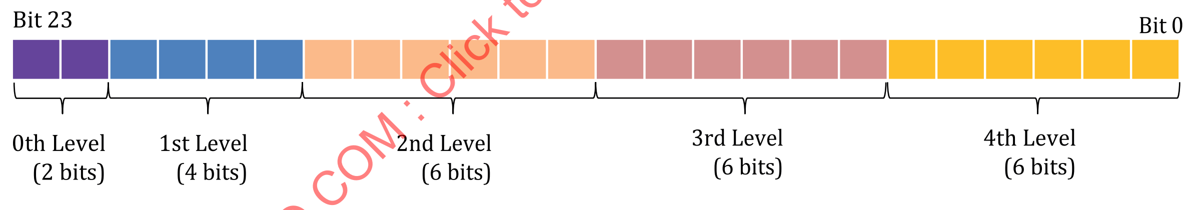


Figure B.2 — Fields for CategoryID

Table B.3 — Classification for the 0-th level in CategoryID of Figure B.2

Classification	Value	Note
Module with HW aspects and SW aspects	00	
Hardware Module	01	
Software Module	10	
Development/Testing Tools	11	Tools/Modules for Development/Analysis/Testing

Annex C
(normative)

Representation of common information

C.1 General

This Annex represents information to be used when creating an object for a class defined in Clause 4. The information can be represented in XML or JSON. In this document, the information is represented in XML. XML schema for information on class CIM shall be used as suggested below and its example is given below. Descriptions of the elements' names are provided in Table C.1. The sub-elements are described in C.2 to C.8.

The XML schema for a CIM of a module is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="GenInfo" type="xs:string"/>
  <xs:element name="IDnType" type="xs:string"/>
  <xs:element name="Properties" type="xs:string"/>
  <xs:element name="IOVariables" type="xs:string"/>
  <xs:element name="Services" type="xs:string"/>
  <xs:element name="Infra" type="xs:string"/>
  <xs:element name="SafeSecure" type="xs:string"/>
  <xs:element name="Modelling" type="xs:string"/>
  <xs:element name="Module">
    <xs:annotation>
      <xs:documentation>version = "1.1" </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="GenInfo"/>
        <xs:element ref="IDnType"/>
        <xs:element ref="Properties"/>
        <xs:element ref="IOVariables"/>
        <xs:element ref="Services"/>
        <xs:element ref="Infra"/>
        <xs:element ref="SafeSecure"/>
        <xs:element ref="Modelling"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on XML schema for a CIM of a module, an example of information for a CIM of a module is provided as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>   <!-- version = "1.1" OK  -->
<Module>
  <GenInfo>             </GenInfo>                  <!-- See C.2 -->
  <IDnType>             </IDnType>                  <!-- See C.3 -->
  <Properties>            </Properties>             <!-- See C.4 -->
  <IOVariables>           </IOVariables>            <!-- See C.4 -->
  <Services>             </Services>                <!-- See C.6 -->
  <Infra>               </Infra>                    <!-- See C.7 -->
  <SafeSecure>          </SafeSecure>               <!-- See C.8 -->
  <Modelling>           </Modelling>                <!-- See C.9 -->
</Module >
```

Table C.1 — XML element for a CIM model of a module

Element Name	description
Module	All properties for a module are defined within this element. This element describes a root element for providing information of a module.

C.2 General information of a module

XML elements for information for GenInfo shall be used as in Table C.2. XML schema for Table C.2 shall be used as suggested below. Examples of information for GenInfo are provided below, written in XML.

Table C.2 — XML elements for GenInfo

Element Name	Description
GenInfo	All properties are defined within this element. This element describes the general information of a module.
ModuleName	Used to define a Module Name
Description	Used for explaining overview of a module
Manufacturer	Used for manufacturer’s information of a module such as contact information and name
Examples	Used for typical use case examples of a module

The XML schema for GenInfo is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="ModuleName" type="xs:string"/>
  <xs:element name="Description" type="xs:string"/>
  <xs:element name="Manufacturer" type="xs:string"/>
  <xs:element name="Examples" type="xs:string"/>
  <xs:element name="GenInfo">
    <xs:annotation>
      <xs:documentation>version = “1.1” OK</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ModuleName"/>
        <xs:element ref="Description"/>
        <xs:element ref="Manufacturer"/>
        <xs:element ref="Examples"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on the XML schema for GenInfo, an example of information for GenInfo is provided as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>   <!-- version = “1.1” OK  -->
<GenInfo>
  <ModuleName> Module Name </ModuleName>        <!-- string -->
  <Description> Description of module  </Description>    <!-- string -->
  <Manufacturer> manufacturer </Manufacturer>         <!-- string -->
  <Examples> list of use case </Examples>              <!-- string -->
</GenInfo>
```

C.3 Information for module ID

XML elements for information for class IDnType shall be used as in Table C.3. XML schema for Table C.2 shall be used as suggested below. Examples of information for class IDnType are provided below. The first example is an example of a basic module and the second example is an example of a composite module with both hardware aspects and software aspects. If only the element “HWLIST” exists, the composite module is a kind of hardware module. If only the element “SWLIST” exists, the composite module is a kind of software module. If two elements “HWLIST” and “SWLIST” exist at the same time, the composite module is a kind of hardware-software module

Table C.3 — XML elements for module ID

Element Name	Description
IDnType	A module ID, an information model version, lists of all modules used in a module are defined within this element. The list of modules is provided only for a composite module.
ID	Used for manufacturer’s unique product reference number for a module (see Annex B). This element can have an attribute “type“ for providing the module type such as basic module or composite module. If its value is “Bas” or the attribute is not defined, the module is a kind of basic module. If its value is “Com”, the module is a kind of composite module.
InformationModelVersion	Provides the version number of the information model used
HWlist	Used for modules with hardware aspects and software aspects and hardware modules in a composite module List of the moduleIDs of the composite module This element has one or more attributes “ModuleID”, which are the IDs of the modules constituting the module.
SWlist	Used for the set of software modules in the composite module List of the moduleIDs of the composite module This element has one or more attributes “ModuleID”, which are the IDs of the modules constituting the module.

The XML schema for module ID is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="ModuleID" type="xs:string"/>
  <xs:element name="ID" type="xs:string"/>
  <xs:element name="InformationModelVersion" type="xs:float"/>
  <xs:element name="HWlist">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ModuleID" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SWlist">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ModuleID" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="IDnType">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ID"/>
        <xs:element ref="InformationModelVersion"/>
        <xs:element ref="HWlist"/>
        <xs:element ref="SWlist"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="type"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on the XML schema for module ID, an example of information for module ID of a basic module is provided as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<IDnType type="Bas">   <!-- example of basic module -->
 <ID> moduleId </ID>  <!-- provided by manufacturer anddefined in Annex B-->
 <InformationModelVersion> 1.0 </InformationModelVersion>  <!-- string -->
```

```
</IDnType>
```

Based on the XML schema for module ID, an example of information for module ID of a composite module is provided as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <IDnType type="Com">          <!-- example of a composite module -->
 <ID> moduleId  </ID>          <!-- ID for the composite module -->
 <InformationModelVersion> 1.0 </InformationModelVersion>  <!-- string -->
 <HWlist> <!-- in the list, IIDs of modules are not defined  -->
   <ModuleID>  hwModuleID1  </ModuleID>
   <ModuleID>  hwModuleID2  </ModuleID>
   <ModuleID>  hwModuleID3  </ModuleID>
   <ModuleID>  hwModuleID4  </ModuleID>
 </HWlist>
 <SWlist>
   <ModuleID>  swModuleID1  </ModuleID>
   <ModuleID>  swModuleID2  </ModuleID>
   <ModuleID>  swModuleID3  </ModuleID>
   <ModuleID>  swModuleID4  </ModuleID>
 </SWlist>
</IDnType>
```

C.4 Information for classes Properties and IOVariables

Most information in classes Properties and IOVariables is related to the execution of modules and used as parameters and/or variables in software aspects. For hardware aspects, it is necessary that some use cases be provided and examples of information for Inputs and Outputs be given.

Regarding information for class Properties defined in 4.3.3, it is mandatory that the name and the value of a property and the data type of the value be provided. The unit of the value and the related description can be included for clarity of use.

Information for class IOVariables defined in 4.3.4 includes the name, value, and data type of the variable, and the input/output type. The unit of the value and the related description can be included for clarity of use. Especially, variables that can be simultaneously utilized as both an input and output variable be marked "input/output".

XML elements for information of Properties, Inputs and Outputs shall be used as in Table C.4. XML schemas for "Properties" and "Inputs and Outputs", which are based on Table C.4, shall be used as suggested below, respectively. An example of information for classes Properties and IOVariables is provided below.

Table C.4 — XML elements for classes Properties and IOVariables

Element Name	Description
Properties	All related properties are defined within this element. NOTE If a namespace is needed, it will be defined in upcoming standards such as ISO 22166-202.
Property	Used to define an individual property in the element "Properties". This element can have an attribute "Complex" for providing for the array data type and complex data types such as structures and classes. If this attribute is an "array" or "class", Property will have an array or class structure. If not defined, Property has a simple data type.
IOVariables	All variables are defined within this element. The type of variable is classified into 3 types: input, output, or input/output. This element can have the attribute "Name" as a domain name. Then variables defined in the element "IOVariables" will have the same domain.
Inputs	Used for the set of input variables
Outputs	Used for the set of output variables
InOuts	Used for the set of variables that can be simultaneously utilized as both input and output variables

Table C.4 *(continued)*

Element Name	Description	
Input	Used for a variable assigned only as an input variable in the element “IOVariables”	The element can have the attribute “Complex“ to provide for the array data type and the complex data types such as structures and classes. If this attribute has an “array” or “class”, variables have an array or a class structure. If not defined, it can a new simple data type in DataProfile (see Table 4.6).
Output	Used for a variable assigned only as an output variable in the element “IOVariables”	
InOut	Used for a variable that can be simultaneously utilized as both an input and output variable in the element “IOVariables”	
name	Name of a property or a variable used in a module (e.g. encoder, ratedCurrent, MaxCurrent, P_Coeff)	
value	Used for initialization of a property or a variable If a property or a variable has the type “array”, this attribute value has one or more values using the attribute “Item”.	
type	Data type of a property or a variable; Its examples are Boolean, Integer, UnlimitedNatual, Real String, Enumeration, void, and any, which are listed in Table C.5.	
complex	Used to define an array, class, vector, or pointer of the data types of Table C.5, examples of which are Boolean, Integer, UnlimitedNatual, Real String, Enumeration, void, and any.	
complexName	Name for complex data type, e.g. In the declaration of “cylinderType class {float32 length; float32 radius;} cylinder;” complexName is “cylinderType” and the name is “cylinder”.	
unit	Unit of a property or a variable (e.g. ampere, metre, Celsius, no unit, etc.)	
description	Explanation of the related property or variable	
additionalInfo	Additional information for variables. This element can have zero or more values utilizing the 'Item' attribute, where each value consists of a 'name' and 'value' pair.	

Table C.5 shows the abstract data type used in the class definition and data types as keywords used in XML, which shall be used in this document. If the element type is enumeration, the new attribute “enumeration” shall be used. The element “unit” means the magnitude of the physical quantity, whose examples are meter, rpm, sec, bps (bit per sec), ampere, volt, watt, and Celsius. In particular, the element “unit” has a special unit of “none”, which means no physical quantity.

Table C.5 — Abstract data type for class definition and data types as keywords for XML

abstract data type used in the class definition	data types as keywords used in XML	Description
Boolean	bool	True(1) or False(0)
Integer	int	Default integer type, depending on OS.
	int8	Signed integer of 1 byte
	int16	Signed integer of 2 bytes
	int32	Signed integer of 4 bytes
	int64	Signed integer of 8 bytes
UnlimitedNatural	uint	Default unsigned integer type, depending on OS.
	uint8	Unsigned integer of 1 byte
	uint16	Unsigned integer of 2 bytes
	uint32	Unsigned integer of 4 bytes
	uint64	Unsigned integer of 8 bytes
Real	float32	Floating point of 4 bytes
	float64	Floating point of 8 bytes
String	string	Sequence of alphanumeric text or other symbols, a kind of class
Enumeration	enumeration	Enumeration type

Table C.5 *(continued)*

abstract data type used in the class definition	data types as keywords used in XML	Description
void	void	Data type that has no values and is used to represent nothing
class	class	Complex data type such as structure or class, which should have a name
any	any	Used to store any type of value in a variable.
Not available	array	Array type, a kind of complex data
	pointer	Pointer type, a kind of complex type.
	vector	Collection of objects, a kind of class

The XML schema for class Properties is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="value">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="Item" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Item" type="xs:float"/>
  <xs:element name="Property">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="value" minOccurs="0"/>
        <xs:element ref="Property" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="optional"/>
      <xs:attribute type="xs:string" name="type" use="optional"/>
      <xs:attribute type="xs:string" name="value" use="optional"/>
      <xs:attribute type="xs:string" name="unit" use="optional"/>
      <xs:attribute type="xs:string" name="description" use="optional"/>
      <xs:attribute type="xs:string" name="complex" use="optional"/>
      <xs:attribute type="xs:string" name="complexName" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Properties">
    <xs:annotation>
      <xs:documentation>example of class type property</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Property" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The XML schema for class IOVariables is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Value" type="xs:float"/>
  <xs:element name="Item">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Value"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="type" type="xs:string"/>
```

```
<xs:element name="value" type="xs:float"/>
<xs:element name="unit" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="additionalInfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Item" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Input">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element ref="name" minOccurs="0"/>
      <xs:element ref="type" minOccurs="0"/>
      <xs:element ref="value" minOccurs="0"/>
      <xs:element ref="unit" minOccurs="0"/>
      <xs:element ref="description" minOccurs="0"/>
      <xs:element ref="additionalInfo" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="name" use="optional"/>
    <xs:attribute type="xs:string" name="type" use="optional"/>
    <xs:attribute type="xs:float" name="value" use="optional"/>
    <xs:attribute type="xs:string" name="unit" use="optional"/>
    <xs:attribute type="xs:string" name="description" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="Output">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element ref="name" minOccurs="0"/>
      <xs:element ref="type" minOccurs="0"/>
      <xs:element ref="unit" minOccurs="0"/>
      <xs:element ref="description" minOccurs="0"/>
      <xs:element ref="additionalInfo" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="name" use="optional"/>
    <xs:attribute type="xs:string" name="type" use="optional"/>
    <xs:attribute type="xs:string" name="unit" use="optional"/>
    <xs:attribute type="xs:string" name="description" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="Inout">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element ref="name" minOccurs="0"/>
      <xs:element ref="type" minOccurs="0"/>
      <xs:element ref="unit" minOccurs="0"/>
      <xs:element ref="description" minOccurs="0"/>
      <xs:element ref="additionalInfo" minOccurs="0"/>
      <xs:element ref="Inout" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="name" use="optional"/>
    <xs:attribute type="xs:string" name="type" use="optional"/>
    <xs:attribute type="xs:string" name="unit" use="optional"/>
    <xs:attribute type="xs:string" name="description" use="optional"/>
    <xs:attribute type="xs:string" name="className" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="Inputs">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Input" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Outputs">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Output" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name="InOuts">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Inout"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="IOVariables">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Inputs"/>
        <xs:element ref="Outputs"/>
        <xs:element ref="InOuts"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on the XML schema for classes Properties and IOVariables, examples of information for classes Properties and IOVariables are provided as follows:

```
Example of information for classes Property and IOVariable in XML:

<?xml version="1.0" encoding="UTF-8" ?>
<Properties>  <!-- properties -->
 <Property name="maxRatedCurrent" type="float32" unit="ampere" description = "maximum of rated
current for motor"  value ="15" />   <!--  example of single property -->
 <Property name="maxRatedVoltage" type="float32" unit="volt" description = "maximum of rated
voltage for motor">
   <value> 5 </value>
 </Property>   <!--  example of single property -->
 <Property name="StatusEmerg" type="boolean" unit="none" description = "existence of emergency
stop button"  value ="True" />
 <Property name="SizeBd" type="uint8" unit="mm" description = "size of board"  value ="30" />
 <Property name="Weight" type="uint16" unit="gram" description = "weight of product"  value
="450000" />
 <Property name="NoPixelDisplay" type="uint32" unit="none" description = "pixel number"  value
="345600" />
 <Property name="NoPixelCamera" type="uint64" unit="none" description = "pixel number"  value
="8294400" />
 <!-- following is example of array: float32 [6] initPos  -->
 <Property name="initPose" complex ="array" type="float32" unit="none" description="initial
pose, x,y,z, yaw,roll,pitch, in order. Unit of yaw, roll, and pitch = degree " >
   <value>
     <Item>  0 </Item>
     <Item>  0 </Item>
     <Item>  0 </Item>
     <Item>  10 </Item>
     <Item>  20 </Item>
     <Item>  10 </Item>
   </value>
 </Property>   <!--  example of array type property -->
<!-- following is example of class: -->
<!--   class cylinderType {          -->
<!--      float32 length;        -->
<!--      float32 radius         -->
<!--    } cylinder;                       -->
 <Property name="cylinder" complex = "class" complexName="cylinderType">
   <Property name="length" type="float32" value = "100.0" unit="millimetre" description =
"length of cylinder" />
   <Property name="radius" type="float32" value = "3.0" unit="millimetre" description =
"radius of cylinder" />
 </Property>   <!--  example of class type property -->
</Properties>

<IOVariables>
 <Inputs>
   <Input name="controlvalue1" type="float32" value="5.0" unit="ampere" description = "current
control command to motor" />
   <Input>
```

```
    <name> controlvalue2 </name>
    <type> float32 </type>
    <value> 5.0 </value>
    <unit> voltage </unit>
    <description> voltage control of motor </description>
    <additionalInfo>
       <Item>
          <Name> min Voltage  </Name>
          <Value> -5.0 </Value>
       </Item>
       <Item>
          <Name> max voltage  </Name>
          <Value> 5.0 </Value>
       </Item>
    </additionalInfo>
   </Input>
  </Inputs>
  <Outputs>
   <Output name="encoder" type ="uint32" unit = "none" description = "state of motor" />
   <Output>
    <name> Status </name>
    <type> uint8 </type>
    <unit> None </unit>
    <description> Status </description>
    <additionalInfo>
       <Item>
          <Name> IDLE  </Name>
          <Value> 0 </Value>
       </Item>
       <Item>
          <Name> WORKING  </Name>
          <Value> 1 </Value>
       </Item>
       <Item>
          <Name> ERROR  </Name>
          <Value> 2 </Value>
       </Item>
    </additionalInfo>
   </Output>
  </Outputs>
  <InOuts>
   <Inout name="pose" type = "" className="poseType">
     <Inout name="x" type="float32" unit="none" description = "position x" />
     <Inout name="y" type="float32" unit="none" description = "position y" />
     <Inout name="z" type="float32" unit="none" description = "position z" />
     <Inout name="yaw" type="float32" unit="degree" description = "yaw" />
     <Inout name="roll" type="float32" unit="degree" description = " roll" />
     <Inout>
       <name> pitch </name>
       <type> float32 </type>
       <unit> degree </unit>
       <description> pitch </description>
       <additionalInfo>
         <Item>
            <Name> min </Name>
            <Value> -100 </Value>
         </Item>
         <Item>
            <Name> max </Name>
            <Value> 100 </Value>
         </Item>
      </additionalInfo>
    </Inout>
   </Inout>
  </InOuts>
</IOVariables>
```

C.5 Information for services of a module

XML elements for information of class Services shall be used as in Table C.6. XML schemas for “Services”, which are based on Table C.6, shall be used as suggested below. Two examples of information for Services (or Functions/Capabilities) are provided below, where one is for IDL and the other is for XML.

This information is generally utilized in programs written in languages such as C/C++, Java, and Python. Hence, it is important to determine the order of arguments and to receive the additional return values via arguments. The data type of the arguments used as the additional return values is pointer in C/C++ or reference type in Java and Python.

Table C.6 — XML elements for class Services

Element Name	Description
Services	All information for Services are defined within this element. This element describes the information about the Services (or Functions/Capabilities) of a module.
Service	One service-related information item is listed. The element has an attribute “type” for providing the related description language such as IDL or XML. If the value of the attribute “type” is IDL, the provided Service is described in IDL and the element value is the path of the related IDL file. If the value of the attribute “type” is XML, the provided Service is described in XML and the element values are provided in other XML elements.
ID	Used to identify the interface that a module uses or provides
PVType	Used to classify the interface type of a module, whose value is either “Physical” or “Virtual” If the module provides a physical interface such as a mechanical interface, the value is “Physical”. If the module provides a software interface such as an API (Application Programming Interface), the value is “Virtual”.
MOType	Used to declare that the provided interface is mandatory or optional.
Properties	Used to define properties that a module has to use (or set) when the module utilizes the provided interface The element has two attributes, “Name” and “Value”, which are the name and the value of the property that the module has.
Methods	Used to define methods that the module has to utilize One or more methods are listed in the element.
Method	Used to define one individual method in the element “Methods”
MethodName	Used to identify the provided method
ArgSpec	Used to define the list of arguments, where an argument consists of the 3 attributes, “Type”, “Name” and “InOutType” “Type” represents the data type of the argument, which is defined in Table C.5, “Name” means the argument name of the method, and “IOType” provides the direction of the argument and is input, output or both. If “Type” is complex, the additional attribute “inDataType” shall be used to define the data type of real data stored in complex data such as an array or class.
RetType	Used to define the data type that the method returns
additionalInfo	Additional information for variables. This element can have zero or more values utilizing the 'Item' attribute, where each value consists of a 'name' and 'value' pair.

The XML schema for class Services is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Value" type="xs:string"/>
  <xs:element name="Item">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Value"/>
      </xs:sequence>
    </xs:complexType>
```

```
</xs:element>
<xs:element name="MethodName">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="value" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="ArgSpec">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element ref="Type" minOccurs="0"/>
      <xs:element ref="Name" minOccurs="0"/>
      <xs:element ref="IOType" minOccurs="0"/>
      <xs:element ref="additionalInfo" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="Type" use="optional"/>
    <xs:attribute type="xs:string" name="Name" use="optional"/>
    <xs:attribute type="xs:string" name="IOType" use="optional"/>
    <xs:attribute type="xs:string" name="inDataType" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="RetType">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="value" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="MOType">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="value" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="ReqProvType">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="value" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="additionalInfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Item"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Type" type="xs:string"/>
<xs:element name="IOType" type="xs:string"/>
<xs:element name="Method">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="MethodName"/>
      <xs:element ref="ArgSpec" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element ref="RetType" minOccurs="0"/>
      <xs:element ref="MOType"/>
      <xs:element ref="ReqProvType"/>
      <xs:element ref="additionalInfo" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name="ID" type="xs:string"/>
  <xs:element name="PVType">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="value" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Methods">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Method" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element ref="additionalInfo" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Properties">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="Name" use="optional"/>
          <xs:attribute type="xs:string" name="Value" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Service">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="ID" minOccurs="0"/>
        <xs:element ref="PVType" minOccurs="0"/>
        <xs:element ref="MOType" minOccurs="0"/>
        <xs:element ref="Properties" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element ref="Methods" minOccurs="0"/>
        <xs:element ref="additionalInfo" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="type" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Services">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Service" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on IDL defined in 4.3.6, an example of information for CIMService is provided as follows:

```
interface Service: CIMService
{
   uint8 initialize(int64 val1, float32 val2); //services provided by module
   uint8 finalize(int32 val1, float32 val2, int32 val3);
   uint8 read(int fd, uint8[] buf, uint16 nbytes);
}
```

Based on the XML schema for class Services, an example of information for class Services is provided as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Services>
 <Service type = "IDL">
    ./IDL/AppLayer/sdo.idl
 </Service>
 <Service type = "IDL">
    ./IDL/AppLayer/sync.idl
 </Service>
 <Service type = "IDL">
```

```
   ./IDL/DeviceDriver/CAN.idl
</Service>
 <Service type = "XML">
  <ID> NMT </ID>ID>
  <PVType> Virtual </PVType>
  <MOType> MANDATORY </MOType>
  <Methods>
    <Method>
      <MethodName value = "NMT_SearchNode" />
      <ArgSpec Type="int8" Name="iNodeID" IOType="IN" />
      <RetType value="bool" />
      <MOType value = "MANDATORY" />
      <ReqProvType value="REQUIRED" />
      <additionalInfo>
         <Item>
            <Name> initialValue </Name>
            <Value> 0 </Value>
         </Item>
      </additionalInfo>
    </Method>
    <Method>
      <MethodName value = "NMT_AddNode" />
      <ArgSpec>
      <Type> int8 </Type>
      <Name> iNodeID </Name>
      <IOType> IN </IOType>
         <additionalInfo>
            <Item>
               <Name> minValue </Name>
               <Value> 20 </Value>
         </Item>
         </additionalInfo>
      </ArgSpec>
      <ArgSpec Type="int8" Name="iType" IOType="IN" />
      <ArgSpec Type="int" Name="iLifeTime" IOType="IN" />
      <RetType value="bool" />
      <MOType value = "MANDATORY" />
      <ReqProvType value="REQUIRED" />
    </Method>
    <Method>
      <MethodName value = "NMT_GetNodeInfo" />
      <ArgSpec Type="int8" Name="iNodeID" IOType="IN" />
      <ArgSpec Type="pointer" inDataType="int" Name="iType" IOType="OUT" />
      <ArgSpec Type="pointer" inDataType="int" Name="pTime" IOType="OUT" />
      <ArgSpec Type="pointer" inDataType="uint8" Name="pLifeTime" IOType="OUT" />
      <RetType value="bool" />
      <MOType value = "MANDATORY" />
      <ReqProvType value="REQUIRED" />
    </Method>
    <additionalInfo>
      <Item>
        <Name> InvokeFirstService </Name>
        <Value> NMT_SearchNode </Value>
      </Item>
    </additionalInfo>
  </Methods>
  <additionalInfo>
   <Item>
      <Name> CANID </Name>
      <Value> 0FF </Value>
   </Item>
  </additionalInfo>
</Service>
<Service type = "XML">
 <ID> RS485Driver </ID>
 <PVType value="Virtual" />
 <MOType> OPTIONAL </MOType>
 <Properties Name = "Baud" Value ="9600" />
 <Properties Name = "STOPBit" Value ="No" />
 <Methods>
   <Method>
      <MethodName value = "initialize" />
```

```
        <ArgSpec Type="int64" Name="val1" IOType="IN" />
        <ArgSpec Type="float32" Name="val2" IOType="IN" />
        <RetType value="uint8" />
        <MOType value = "MANDATORY" />
        <ReqProvType value="PROVIDED" />
      </Method>
      <Method>
        <MethodName value = "initialize" />
        <ArgSpec Type="int64" Name="val1" IOType="IN" />
        <ArgSpec Type="float32" Name="val2" IOType="IN" />
        <ArgSpec Type="float32" Name="val3" IOType="IN" />
        <RetType value="uint8" />
        <MOType value = "OPTIONAL" />
        <ReqProvType value="PROVIDED" />
      </Method>
      <Method>
        <MethodName value = "finalize" />
        <ArgSpec Type="int32" Name="val1" IOType="IN" />
        <ArgSpec Type="float32" Name="val2" IOType="IN" />
        <ArgSpec Type="int32" Name="val3" IOType="IN" />
        <RetType value="uint8" />
        <MOType value = "MANDATORY" />
        <ReqProvType value="PROVIDED" />
      </Method>
      <Method>
        <MethodName value = "read" />
        <ArgSpec Type="int" Name="fd" IOType="IN" />
        <ArgSpec Type="pointer" inDataType ="uint8" Name="val2" IOType="INOUT" />
        <ArgSpec Type="uint16" Name="nbytes" IOType="IN" />
        <MOType value = "MANDATORY" />
        <ReqProvType value="PROVIDED" />
        <additionalInfo>
          <Item>
            <Name> minValue </Name>
            <Value> 0 </Value>
          </Item>
        </additionalInfo>
      </Method>
    </Methods>
  </Service>
</Services>
```

C.6 Information for infrastructure

The information relates to the type of infrastructure support and/or the environmental protection such as power sources, data bus, and ingress protection (IP). The power lines are related to the power type supplied to the module which includes power consumption and the power type that the module supplies if any. The data bus is related to the communication type connected to the module (e.g. Ethernet, EtherCAT, CAN, USB and RS422). IP relates to IP code that the module provides according to IEC 60529 Ed.2.2 b: 2013. XML elements for information of Infrastructure class shall be used as in Table C.7. XML schemas for class Infrastructure, which are based on Table C.7, shall be used as suggested below. An example of information for class Infrastructure is provided below.

NOTE Data bus in class Infrastructure means the bus on which modules communally use in the system.

Information for Databuses provides the following: Protocol type, transmission speed, and supported API, where examples of the protocol type are CAN2.0, EtherCAT, Ethernet, and RS485.

Table C.7 — XML elements for class Infrastructure

Element Name	description
Infra	All properties for class Infrastructure are defined within this element. This element describes the information about the infrastructure of a module.
Powers	Used to define the power that a module uses or provides. Power types are as follows: Electric power, Pneumatic power, and Hydraulic power. NOTE In the current version, ElecPower is defined.
ElecPower	Used to define Electric Power Its attributes are value and unit. Value means the amount of power in units, where “unit” can be “watt“ or “W“, or “mW“.
RatedPower	Used to define the rated power Its attributes are “value” and “unit”.
MaxPower	Used to define the maximum power Its attributes are “value” and “unit”.
DataBuses	Used for defining of the data buses that a module uses
Databus	Used for defining one of the data buses that a module uses
ConnectorType	Used for defining the connector type for the data bus that a module uses The attribute “type” is “MALE” or “FEMALE”, that is the connector type that the module has.
TypePhyMac	Used for defining the types of Physical and MAC protocols that a module uses (e.g. CAN, EtherCAT, Ethernet, RS232, RS485).
TypeNetTrans	Used for defining the types of Network and Transport protocol that a module uses (e.g. IP, TCP, TCP/IP, CANopen). The values are defined in detail in ISO 22166-202.
TypeApp	Used for defining of types of application layers that a module uses One or more application services can be listed as follows: POD, OBD, SYNC. The values will be defined in detail in ISO 22166-202.
Speed	Used for defining of transmission speed that a module provides
DBType	Used to define a type of the database management system that a module uses
IP	Used to define the code of Ingress Protection of IEC 60529.
additionalInfo	Additional information for variables. This element can have zero or more values utilizing the 'Item' attribute, where each value consists of a 'name' and 'value' pair.

The XML schema for class Infrastructure is as follows:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Value" type="xs:float"/>
  <xs:element name="Item">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="Name" minOccurs="0"/>
        <xs:element ref="Value" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="RatedPower">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:byte" name="value"/>
          <xs:attribute type="xs:string" name="unit"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="MaxPower">
    <xs:complexType>
      <xs:simpleContent>
```

```
        <xs:extension base="xs:string">
          <xs:attribute type="xs:float" name="value"/>
          <xs:attribute type="xs:string" name="unit"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="additionalInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Item"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ElecPower">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="RatedPower"/>
        <xs:element ref="MaxPower"/>
        <xs:element ref="additionalInfo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ConnectorType" type="xs:string"/>
  <xs:element name="TypePhyMac" type="xs:string"/>
  <xs:element name="TypeNetTrans" type="xs:string"/>
  <xs:element name="TypeApp">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="Item" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Speed">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:short" name="value" use="optional"/>
          <xs:attribute type="xs:string" name="unit" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Databus">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ConnectorType"/>
        <xs:element ref="TypePhyMac"/>
        <xs:element ref="TypeNetTrans"/>
        <xs:element ref="TypeApp"/>
        <xs:element ref="Speed"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Powers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ElecPower"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DataBuses">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Databus" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element ref="additionalInfo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DBType" type="xs:string"/>
  <xs:element name="IP" type="xs:string"/>
  <xs:element name="Infra">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Powers"/>
        <xs:element ref="DataBuses"/>
        <xs:element ref="DBType"/>
        <xs:element ref="IP"/>
        <xs:element ref="additionalInfo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Based on the XML schema for class Infrastructure, an example of information for class Infrastructure is provided as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Infra>
 <Powers>
  <ElecPower>
    <RatedPower value= "43" unit = "watt" />
    <MaxPower value= "54.2" unit = "watt" />
    <additionalInfo>
      <Item>
        <Name> Max Current (unit:Ampere) </Name>
        <Value> 1 </Value>
      </Item>
    </additionalInfo>
  </ElecPower>
 </Powers>
 <DataBuses>  <!-- list of data buses used in Module -->
  <Databus>
    <ConnectorType> DB9 </ConnectorType>   <!-- D-Sub 9-->
    <TypePhyMac> CAN </TypePhyMac>
    <TypeNetTrans> CANopen </TypeNetTrans>
    <TypeApp> <!-- list for app. Layer of CANopen -->
      <Item> OBD </Item>
      <Item> NMT </Item>
      <Item> SDO </Item>
      <Item> PDO </Item>
      <Item> SYNC </Item>
    </TypeApp>
    <Speed value = "1000" unit="kbps" />  <!-- 1 Mbps, unit Kbps -->
  </Databus>
  <Databus>
    <ConnectorType> RJ45 </ConnectorType>   <!-- Jack for EtherNet -->
    <TypePhyMac> EtherCAT </TypePhyMac>
    <TypeNetTrans> IPTCP </TypeNetTrans>
    <TypeApp> Modbus </TypeApp>
    <Speed value = "10000" unit="kbps" />  <!-- 10 Mbps, unit Kbps -->
  </Databus>
  <additionalInfo>
    <Item>
      <Name> Maximum speed of EtherCAT </Name>
      <Value> 100000 </Value>
    </Item>
  </additionalInfo>
 </DataBuses>
 <DBType> SQL DBMS </DBType> <!-- DBMS type  -->
 <IP> IP23CH  </IP>   <!--  Ingress Protection IEC 60529-->
 <additionalInfo>
  <Item>
    <Name> Minimum IP Code </Name>
    <Value> 32 </Value>
  </Item>
 </additionalInfo>
</Infra>
```