compReference number of working document: **ISO/TC 299 N 000**

Date: 2016-11-14

Reference number of document: **ISO/WD**

Committee identification: ISO/TC 299/WG 6

Secretariat: N/A

Modularity for service robots – Part 202 – Information Model for Software Modules

WD stage

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

To help you, this guide on writing standards was produced by the ISO/TMB and is available at https://www.iso.org/iso/how-to-write-standards.pdf

A model manuscript of a draft International Standard (known as "The Rice Model") is available at https://www.iso.org/iso/model_document-rice_model.pdf

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword ... 3
Introduction ... 4
1 Scope ... 5
2 Normative references ... 5
3 Terms and definitions ... 6
4 Common information model for modules ... 8
4.1 General ... 8
4.2 Relationship between Common Information Model and Information models ... 8
4.3 Class model of common information model ... 9
4.3.1 General ... 9
4.3.2 Class for Module ID and Module Type ... 10
4.3.3 Class for Properties ... 11
4.3.4 Class for Input, Output, and Shared Variables of Module ... 12
4.3.5 Class for Status of Module ... 14
4.3.6 Class for Services of Module ... 14
4.3.7 Class for Infrastructure of Module ... 17
4.3.8 Class for Safety and Security of Module ... 18
4.3.9 Class for Modelling of Module ... 20
Annex A (Normative) Naming Rules ... 22
A.1 Introduction ... 22
A.2 Naming rules ... 22
Annex B Representation of common information (Informative) ... 24
B.1 General ... 24
B.2 Information for General Information of module ... 24
B.3 Information for Module ID ... 25
B.4 Information for Property, Inputs and Outputs ... 26
B.5 Information for Functions (Capabilities) ... 28
B.6 Information for infrastructure ... 29
B.7 Information for Safety and Security ... 30
B.8 Information for Modelling ... 31

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO nnn-n was prepared by Technical Committee ISO/TC 299/WG6, Modularity for service robots.

Introduction

This document has been provided an information model for software modules based on IS 22166-1:2021 Modularity for Service Robot Part 1-General Requirements and WD 22166-201 Modularity for Service Robot Part 201- Common Information Model for Modules. Using the information model for software modules, software modules can be easily connected and exchange data among them. The information model for software modules has been designed to make the interoperability, reusability, and automatic composability of modules increase.

Depending on the application, software modules can be combined using the information model for software modules, and can help to satisfy safety/security requirements of systems or composite modules by utilizing the right configuration of appropriate modules.

Robot makers, robot developers, robot system integrators, and robot software designers/developers/integrators are able to obtain necessary modules more easily and utilize various software modules according to functions and budget.

The information model of the robot software modules presented in this document are focused on the common characteristics which all types of software modules shall have and whose examples are as follows:

1. Module ID
2. Properties;
3. Inputs, Outputs, and Variables
4. Status;
5. Services
6. Infrastructure
7. Safety and Security Level;

The information model for software modules includes some methods to get and put the illustrated characteristics. In particular, this document focuses on software module-related Interfaces, properties, variables, behaviour between modules, and status of modules. In particular, safety and security level can be utilized when a new module consisting of modules is created.

Future editions and parts of this family of International Standard might include more specific requirements on various types of composite modules including service robots.

The document presents guidelines for robot interoperability of software module for ensuring their effective connectivity and their correct functionality and enabling published robot system safety/security requirements to be achieved.

Modularity for service robots – Part 202 – Information model for software modules

1 Scope

This document complies with the ISO 22166 family of standards on modularity for service robots. This document presents requirements and recommendations for information models for software modules used in service robots.

This document presents how the information model for software modules relates to interoperability and reusability and is taking into account of safety and security.

In particular, this document focuses on software module-related Interfaces, properties, variables, behaviour between modules, and status of modules.

2 Normative references

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

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

WD 22166-201, Modularity for service robots – Part 201 — Common information model for modules

ISO 13482:2014, Robots and robotic devices — Safety requirements for personal care robots

ISO 13849-1:2015, Safety of machinery — Safety-related parts of control systems — Part 1: General principles for design

IEC 62443-4-2, Security for industrial automation and control systems – Part 4-2: Technical security requirements for IACS components

ISO/IEC 19505-1:2012, Information technology -- Object Management Group Unified Modelling Language (OMG UML) -- Part 1: Infrastructure

ISO/IEC 19516:2020, Information technology — Object management group — Interface definition language (IDL) 4.2

~~IEC 60529 Ed.2.2: 2013, Degrees of protection provided by enclosures (IP Code)~~

~~ISO 11179-5 2015, Information technology — Metadata registries (MDR) — Part 5: Naming principles~~

OMG formal/2008-10-01, Platform Independent Model and Platform Specific Model for Super Distributed Object, Version 1.1

OMG formal/2019-04-03, Protocol DDS Interoperability Wire Protocol (DDSI-RTPS) Version 2.3

DDS :if this word is used in main text, the related reference will be added.

RTC : if this word is used in main text, the related reference will be added.

3 Terms and definitions

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

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

3.2
Common Information model
CIM
Information model that all kinds of modules use commonly.

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

3.4
property
attribute or characteristic of a module

3.5
software module
SW module
module whose implementation consists purely of programmed algorithms

~~**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~~

3.7
module with HW aspects and SW aspects
HW-SW Module
module whose implementation consists of physical parts, software, and communication interface that allows data exchange with other modules.

3.8
Module manager
module that loads the modules and then assigns the module the unique ID if two or more modules have the same module type.

3.9
Instance
process in running or object used in a module

3.10
naming rules
Rules for providing names for data entries

3.11
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 13849-1:2015)

3.12
component
part of something that is discrete and identifiable with respect to combining with other parts to produce something larger

Note 1 to entry: Component can be either software or hardware.

Note 2 to entry: A component is a kind of parts, which are not standardized according to ISO 22166 series.

[source: ISO 22166-1]

3.13
Middleware
software that helps to make transceiving and managing of data easy in module-based distributed applications.

Note 1 to entry: Middleware may be a kind of components. ROS, openRTM, and OPRoS are component types.

3.14
hardware abstraction layer (HAL) (will be removed if not used in main text)
abstraction layer for a component/module that contains hardware aspects, with the abstraction layer providing control of the component/module via a software interface

[source: ISO 22166-1]

3.15
sensing module
sensor module
input module for collecting data about the world around the robot or the state of the robot for use by other modules to support the robot system in performing its task(s)

Note 1 to entry: The module can access to HW-SW module such as lidar, encoder, and camera using HAL, device driver or dedicated driver.

[source: ISO 22166-1]

3.16
actuating module
actuator module
output module whose primary function is to physically move the robot, or alter the world around the robot, in response to instructions from other modules, with the purpose of achieving the robot system's task(s)

Note 1 to entry: The module can access to HW-SW module such as motor using HAL, device driver or dedicated driver.

[source: ISO 22166-1]

Module manager (kamei san)

System management (kamei san)

module configuration management (kamei san)

4 Information model for software modules

4.1 General

The information for software modules (or software information model) shall consist of items shown in Table 4.1, which is based on WD 22166-201 common information model.

Table 4.1. Common information and the corresponding Tag

NO.	ITEMS	Mandatory(M) or Optional(O)			Related Group/Tag name (Abbreviation of each group)
		HW-SW module	HW module	SW module	
1	Module Name	M	M	M	GenInfo
2	Description	O	O	O	
3	Manufactures	M	M	M	
4	Examples	O	O	O	
5	Module ID	M	M	M	IdnType
6	Software Aspects	M	N.A	M	
7	Module properties 1)	M	M	M	ModuleProp
8	Inputs	M	M	M	IOVariable
9	Outputs	M	M	M	
10	Function(capabilities)	M	M	M	Service
11	Infrastructure	M	M	M	Infra
12	Safety/security	M	M	M	SafeSecure
13	Modelling	O	O	O	Modelling

M: Mandatory, O" Optional, N.A: Not Available

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

(mizukawa : suggestion that modelling is provided for SW modelling)

The relationship between CIM and software information model (SIM) is shown in Figure 4.1. That is, SIM is inherited from CIM. Like CIM, SIM can be described using the Unified Modelling Language (UML). SW modules access to HW-SW modules using hardware module with device driver or communication interface. And software modules access to other SW modules using infrastructure such as middleware.

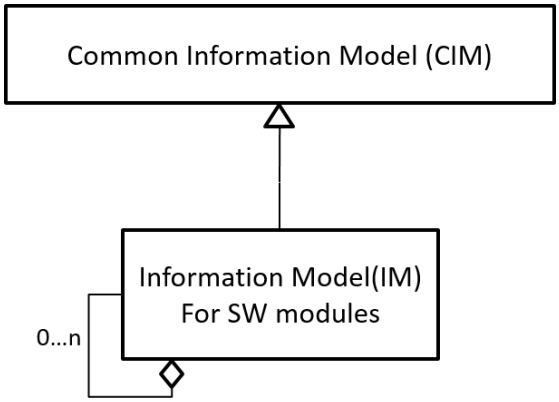


Figure 4.1 Relationship between common information model and software information model

EXAMPLE: An example for information model for a SLAM(Simultaneous Localization and Mapping) module as a kind of composite module is illustrated in Figure 4.2 using multiple sensing modules,

examples of which are Lidar-, camera-, and infrared camera-sensing module. For a localization module, at least one or more sensing modules are utilized. When using a module, the information model provided by the module is used to determine whether to use it. Since the module's interfaces are provided in the module's information model, the modules use them to transmit and receive data.

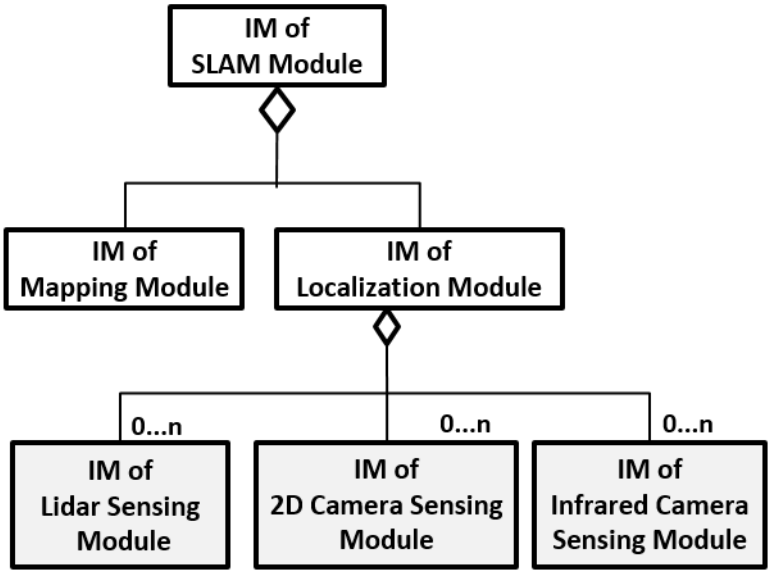


Figure 4.2 Example of Information models for a SLAM software module

4.2 Class model of software information model

4.2.1 General

The software information model shall inherit from common information model specified in WD 22166-201, which is shown in Figure 4.3. SIM is similar to CIM, but the attributes of subclasses can be different from those of CIM.

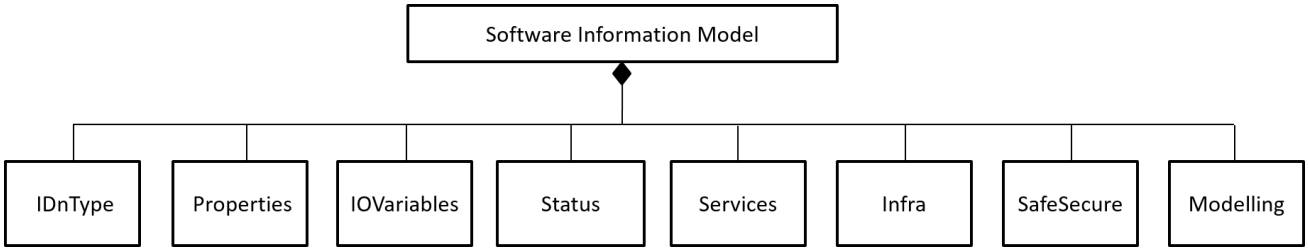


Figure 4.3 Class of software information model

The class of the software information model shall have attributes given in Figure 4.3, which are similar to CIM.

Values of Attributes such as ModuleName, Description, Manufacturer, and Examples are provided via Annex 2. The information model version is the version number of the inofrmation model used when the module is specified. The information model vesion is updated whenever the document is upgraded. IDnType, Propertiees, IOVariables, Status, Services, Infra, SafeSecure, and Modelling in Figure 4.5 mean the class described in 4.3.2-4.3.9.

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

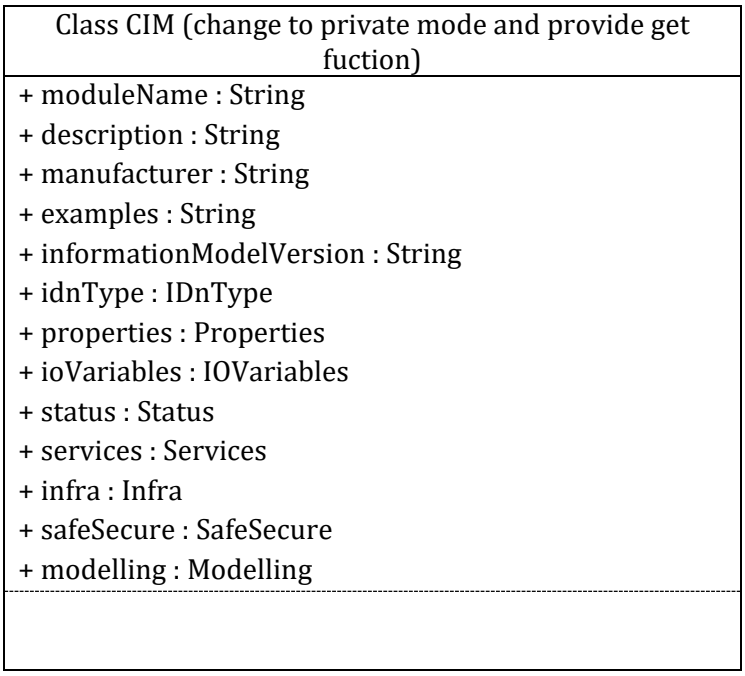


Figure 4.5 Class diagram of CIM class

4.2.2 Class for Module ID

Information for Module ID shall be defined in the class IDnType. Class IDnType shall have attributes given in Figure 4.6. Values of the attribute 'moduleID' and other attributes in Figure 4.6 are provided via Annex B and Annex C. When the same type of modules are used in a module, the modules shall be identified individually. For this purpose, an Instance ID (or IID) is needed and generated uniquely by the module manager. IID shall be unique if IID is used. The method setInstanceID() shall check whether the used IID is unique. The method getInstanceID() is used to get IID.

The attributes 'hwAspects' and 'swAspects' are lists of module IDs for HW aspects and SW aspects, respectively.

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

NOTE 1: Module ID is provided by the manufacturer of a module.
NOTE 2: The moduleID including IID is used in order to access the module. The default IID is 0.

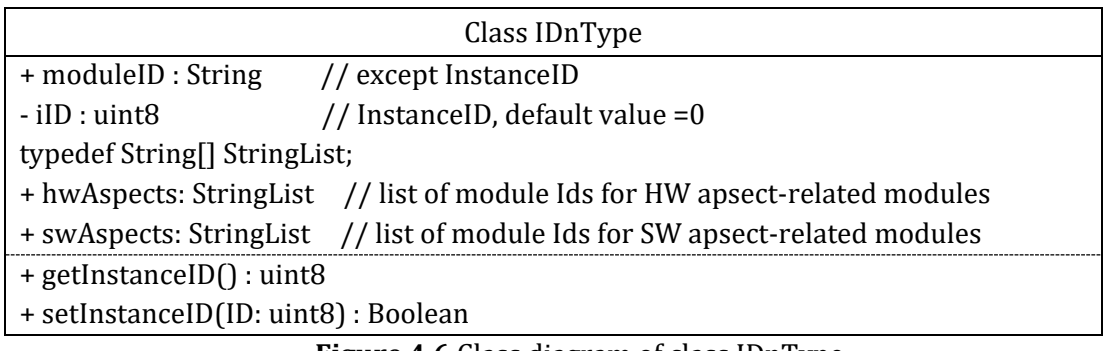


Figure 4.6 Class diagram of class IDnType

4.2.3 Class for Properties

Information for Properties shall be defined in the class Properties. The class Properties shall provide properties of a module as shown in Figure 4.7 ~ Figure 4.9, which include information necessary for execution of the module and information about the operating environment or condition of the module. Figure 4.10 is the class diagrams for Properties and shows the relationship among classes given in Figure 4.7 ~ Figure 4.9. The information necessary for execution of the module is related to values for initialization of modules. 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.

NOTE 1: Values for properties are read-only, provided once by the manufacturer.

For other modules in order to access values of properties of a module, defined as variables, the following format shall be used:

<moduleID>. <propertyName defined in module>,

where moduleID includes the instance ID of the module.

Limit properties which listing is mandatory only to those who can be influenced (set) from the outside or at least to those, which have an expected effect on other modules.

It seems unlikely that all versions of OS and compilers are supported.(Add a version number as additional information)

OStype : Type(Windows, Linux, ..) Version (major, minor), bits

CompilerType : Type(VS, MinGW, Borland, gcc) version, bits

CPUBits (?): needed(?)

Library (?)

ExecutionType: Type (periodic, aperiodicRT, NonRT), deadline or period

EXAMPLE 1: Let's consider following examples, which are given in XML and have to be transformed to like Figure 4.8:

```
<-- example for HW-SW 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 SW module -->
<Property name="Weight" value=150 type="float32" unit="gram" description = "weight of Lidar Sensor" />
<-- example for HW-SW 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 HW 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] />

Class DataProfile	
+ description: string	// explanation of property
+ name: string	// property name
+ type: string	// data type of property
+ unit: string	// unit of property. If no unit, unit is null.

Figure 4.7 Class diagram of class DataProfile

Class Property	
+value ; any	// value of the property

Figure 4.8 Class diagram of class Property

Class Properties	
typedef Property[] PropertyList	
+ property : PropertyList	// list of attributes generated according to the class Property

Figure 4.9 Class diagram of class Properties

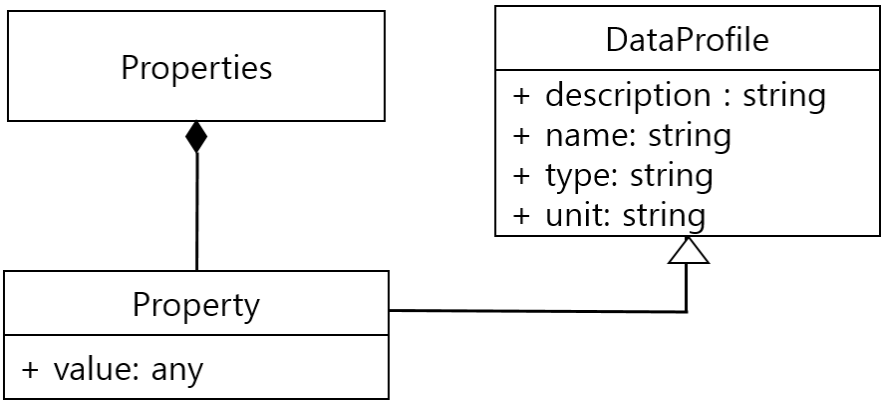


Figure 4.10 Class diagrams for Properties

4.2.4 Class for Input, Output, and Shared Variables of Module

Information for Input, Output, and Shared Variables of Module shall be defined in the class IOVariables in Figure 4.7, and Figure 4.11~Figure 4.13. The class IOVariables shall provide inputs, outputs and variables used in the module, which include information necessary for execution of the module and shall be defined as IN, OUT, INOUT of ioType in Figure 4.11, respectively. The detailed attributes such as the member name and the related data type shall be defined using values of the tag IOVariables in Annex C. Figure 4.14 shows the relationship among class diagrams for class IOVariables, which are provided in Figure 4.7, and Figure 4.11~Figure 4.13.

For other modules in order to access values of Inputs, Outputs, and InOut Variables used in a module, which are defined as variables, the following format shall be used:

<moduleID>. <Name defined in module>,

where moduleID includes the instance ID of the module.

EXAMPLE 1: Let's consider following examples, which are given in XML and have to be transformed to like Figure 4.9 and Figure 4.10:

```
<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

Class VariableProfile
typedef InOut enumeration of {IN, OUT, INOUT } ioType : InOutType // one of {IN, OUT, INOUT }

Figure 4.11 Class diagram of class Variable

Class Variable
+value ; any // value of the variable

Figure 4.12 Class diagram of class Variable

Class IOVariables
+ variable[] : Variable // list of attributes generated according to the tag Variable

Figure 4.13 Class diagram of class IOVariables

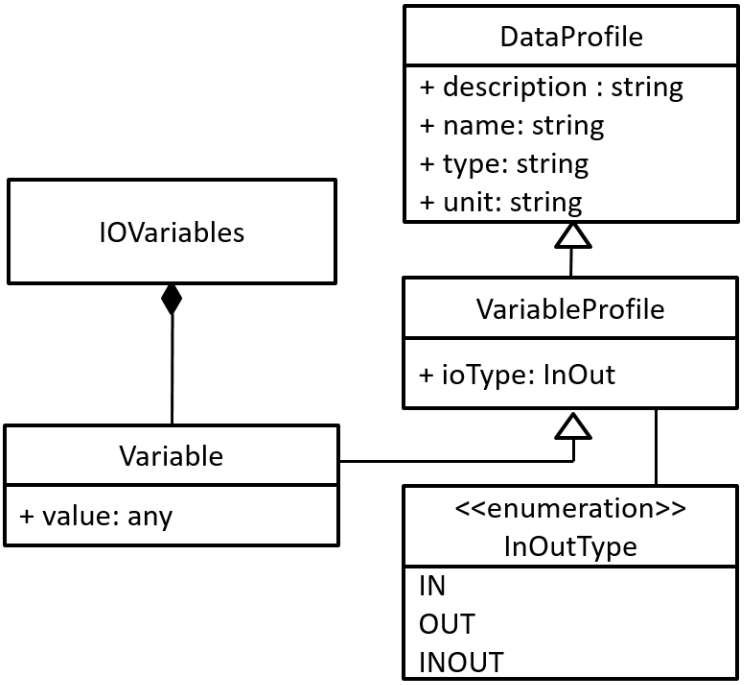


Figure 4.14 Relationship among Class diagrams for class IOVariables

4.2.5 Class for Status of Module

Information for status and health condition of Module shall be defined in the class Status in Figure 4.15. Status class provides the status and/or health condition of a module which represents the health condition of the module if the HW information model exists, the status of execution life cycle if SW information model exists, the status of communication, and the error type occurred in operation.

NOTE 1: Examples of HealthCond is as follows: GH(good health), H1(Health1), H2(Health2), and Fa(Failure). Values of HealthCond will be defined in detail in 203 documents for hardware modules.
NOTE 2: Examples of ErrorType is as follows: OK, BAD_PARAMETER, UNSUPPORTED, OUT_OF_RANGES, PRECONDITION_NOT_MET, and etc. Values of ErrorType will be defined in detail in 202 document and other documents for software modules.
NOTE 3: ExecutionStatus is defined in ISO 22166-1.

Class Status
+ HealthCond : Enumeration // in some modules, HealthCond has no values. + ExecutionStatus : Enumeration // in some modules, ExcutionStatus has no values. + ErrorType : Enumeration // depending on module type, enumeration will be defined

Figure 4.15 class diagram of StatusHealth class

4.2.6 Class for Services of Module

Information for Services of Module shall be defined in the class Services. The class Services provides the methods (or functions) that the module supports, as shown in Figure 4.17. Attributes and methods in Figure 4.17 shall be obtained from CIMServicePackage, defined in IDL, in Figure 4.16 or and obtained from classes defined in Figure 4.18 ~ Figure 4.20. Figure 4.21 shows the relationship among class diagrams for class Services, which are provided in Figure 4.18~Figure 4.20.

NOTE 1: Services in this sub-clause are defined using ISO/IEC 19516:2020 Information technology — Object management group — Interface definition language (IDL) 4.2.

NOTE 2: Basic service in Figure 4.18 means service that is commonly provided by modules and optional service means service that is not common but can be provided additionally by a module.

and optional service

```
interface Service: CIMService
{
      uint8 initialize(int16 val1, float64 val2);                // example interface
      uint8 finalize(int16 val1, float64 val2, int32 val3);   // example interface
      uint8 read(int fd, void* buf, uint nbytes);                // example interface
}

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 interface_typeMO
    {
             string methodName;
            MOType value;     // enumeration {“MANDATORY”, “OPTIONAL”}
    };
    typedef sequence<NameValue> NVList;   // sequence means a list or vector of given data
    typedef sequence<interface_typeMO> MOLIST; // sequence of (methodName, M or O)
    enum PVtype
    {
            PHYSICAL,
            VIRTUAL
    };
    // ServiceProfile
    struct ServiceProfile
    {
              string id;
              PVtype interface_typePV;     // Enumeration PHYSICAL, VIRTUAL
              MOList molist;                // mandatory or optional method defined in CIMService
              NVList properties;            // arguments and their default initialization values for service
              CIMService service;
    };
    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);
    };
};
```

Figure 4.16 CIMServicePackage written in IDL

Class Services	
+ NoOfBasicService : uint8	// Number of Basic services provided
+ NoOfOptionalService : uint8	// Number of Optional services provided

// prototype for basic(or Mandatory) Services
typedef ServiceMethod [] ServiceMethodList;
+ mandService: ServiceMethodList; // mandatory methods(services) provided by module
+ optionalService : ServiceMethodList // optional methods(services) provided by module

Figure 4.17 Class diagram of Services class

Class ServiceProfile
typedef PhysicalVirtual enumeration of { Physical, Virtual } + id: String // name of service profile. One or more service profiles can be provided. + methodList: ServiceMethod[] // list of methods for service + properties: NameValue[] // arguments and their default initialization values for service +pvType: PhysicalVirtual // Physical(Mechanical/Elec) Method or Virtual(SW) Method

Figure 4.18 Class diagram of ServiceProfile class

Class NameValue
+ name : String // name of property + value: any // value of property used for initialization

Figure 4.19 Class diagram of NameValue class

Class ServiceMethod
typedef MOType enumeration of { MANDATORY, OPTIONAL} + methodName: String + argType: ArgSpec[] // list of argument used in a method + retType: String // data type of return value, see Table C.4 + moType: MOType // mandatory or optional method

Figure 4.19 Class diagram of ServiceMethod class

Class ArgSepc
+ type: String // data type of an argument used in a method, see Table C.4 + valueName: String // name of an argument used in a method + inout: InOutType // InOut is define in Figure 4.11

Figure 4.20 Class diagram of ArgSepc class

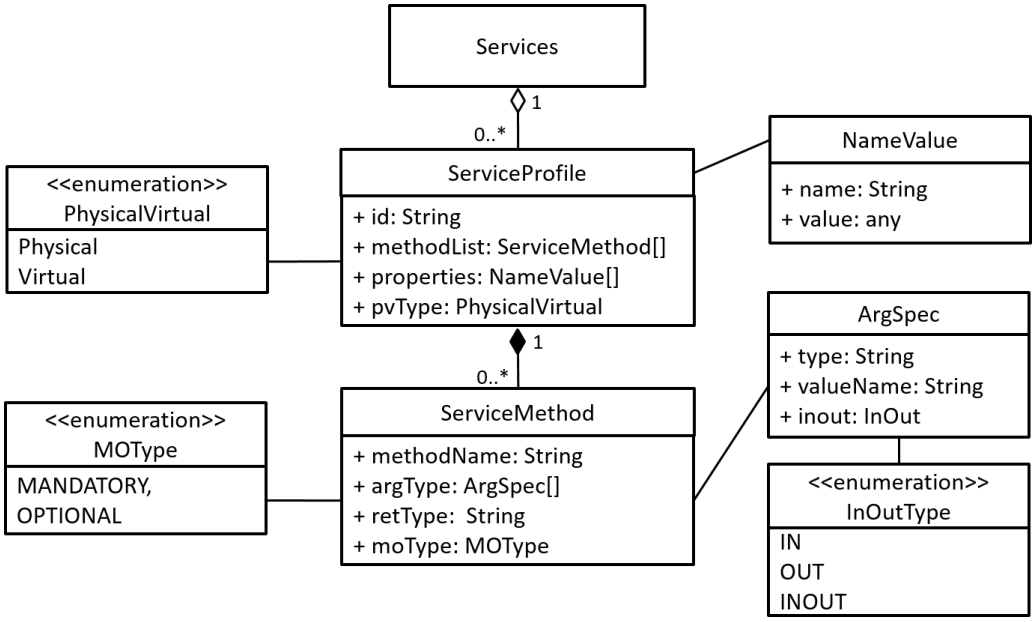


Figure 4.21 Relationship among Class diagrams for class Services

4.2.7 Class for Infrastructure of Module

Information for Infrastructure of Module shall be defined in the class Infrastructure. The class Infrastructure provides the types of infrastructure that the module supports, as shown in Figure 4.24. This sub-clause defines following infrastructure modules: Electric Power, Data Bus, Data Base, and ingress protection(IP).

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

The class DataBus and the class Power used in the class Infrastructure are shown in Figure 4.22 and Figure 4.23, respectively.

NOTE 2: values for typePhyMac in Figure 4.15will be defined in information model for HW modules. Values for typeApp will be defined in in information model for SW modules. values for dbType in Figure 4.16 will defined in in information model for SW modules.

```
typedef String[] Stringlist
Class DataBus {
   String connectotType; // connection type:USB-A, USB-C, USD-C, RG45, DE9
   String typePhyMac;    // Physical/MAC protocol type: USB, CAN2.0, EtherCAT, Ethernet, RS485
   String typeNetTrans; // Network Protocol and Transport Protocol, TCP/IP, CANopen
   Stringlist typeApp;      // session/presentation/application protocol: list of protocol
   float32 speed;           // transmission speed, unit : Kbps
};
```

Figure 4.22 Class diagram of class ElecPWR and class DataBus

```
Class Power {
   float32 ratedPower ;    // rated Power in watt.
   float32 maxPower;      // maximum Power in watt.
};
```

Figure 4.23 Class diagram of class Power

Class Infrastructure

// Electric power type
+ power Power; // see Table 4.23, Pneumatic and Hydraulic types will be defined next.
// dataBuses
typedef DataBus[] DataBusList;
+ noBuses : int8 // the number of databuses used in module. Normal value is 1.
+ dataBus: DataBusList
// types of DataBase mgt system
+ dbType : String // DBMS type used in module. e.g oracle, sql, …
// IP code
+ IPcode : Enumeration // Ingress Protection IEC 60529; two digit

Figure 4.24 Class diagram of Infrastructure class

4.2.8 Class for Safety and Security of Module

The SafeSecure class shall present the safety level and the security level that the module provides, which is shown in Figure 4.27. Figure 4.28 shows the relationship among class diagrams for class SafeSecure, which are provided in Figure 4.25~Figure 4.27.

Measure for the individual safety function shall be specified using the safety function type and the performance level (a~e, none) for the safety function type. A module can include zero or more safety functions. The safety function shall be defined using the class shown in Figure 4.25. Various types of safety functions are shown in Table 4.5. If the module provides a single safety function, the module provides a performance level of the safety function. If the module provides two or more safety functions, the module shall provide an overall performance level of the module using the combination of the performance levels of the provided safety functions of the module or via the verification and validation of the module's overall safety-related function. The safety level for safety function shall be provided using SIL or PL. The security level shall be provided for security function.

Measure for the individual cybersecurity shall be specified using the security type and the security level (0~4) for the type. The various types of security measures are provided in Table 4.6. A module can include zero or more security measures. The measure for individual cybersecurity shall be defined using the class shown in Figure 4.26. If the module provides two or more security measures, the module shall provide an overall security level of 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.

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

NOTE 2: Values for attributes in Figure 4.25 ~ Figure 4.26 are read-only, provided once by the manufacturer.

NOTE 3: The performance level for no safety shall be used as 'none' or 'n'.

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

NOTE 5: Enumeration values of PhySecurityLevel in Figure 4.27 will be defined in information model for Hardware module.

Table 4.5 types of safety functions and their Tags

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

NOTE 4: safety functions that do not list in this table will be added, if needed.

Table 4.6 types of measures for cybersecurity and their Tags

Security Measure Type (IEC 62443-4-2)	Tag Type
Human user identification and authentication	HU_IA
SW 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
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

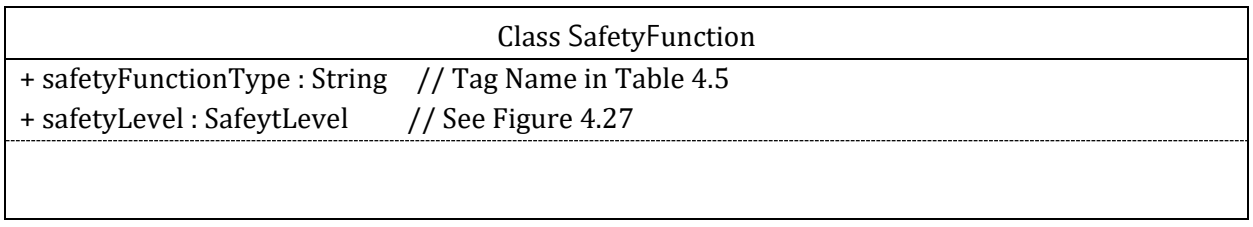
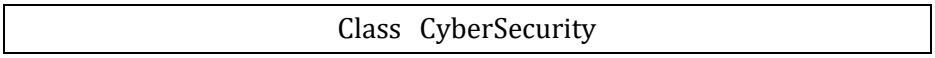


Figure 4.25 class diagram of SafetyFunction



+ securityType : String // Tag Name in Table 4.6 + securityLevel : SecurityLevel // See Figure 4.27

Figure 4.26 class diagram of CyberSecurity

Class SafeSecure

typedef SafeytLevel enumeration of {n, a, b, c, d, e, 0, 1, 2, 3, 4 } // PL or SIL
typedef SecurityLevel enumeration of {0,1,2,3,4 }
+ safetyLevel : SafetyLevel // overall PL of a module a, b, c, d ,e, or n ; 'n' means no safety.
// or SIL 0,1,2,3, or 4; 0 means no safety.
+ phySecurityLevel : Enumeration // overall Physical Security of a module
+ cybSecurityLevel: Enumeration // overall Cyber Security of a module 0,1,2,3,4
typedef SafetyFunction[] SafetyFunctionList;
+ inSafetyLevel: SafetyFunctionList; // PL of individual safety function
typedef CyberSecurity[] CyberSecurityList;
+ inCybSecurityLevel: CyberSecurityList; // individual security function

Figure 4.27 Class diagram of SafetySecurity class

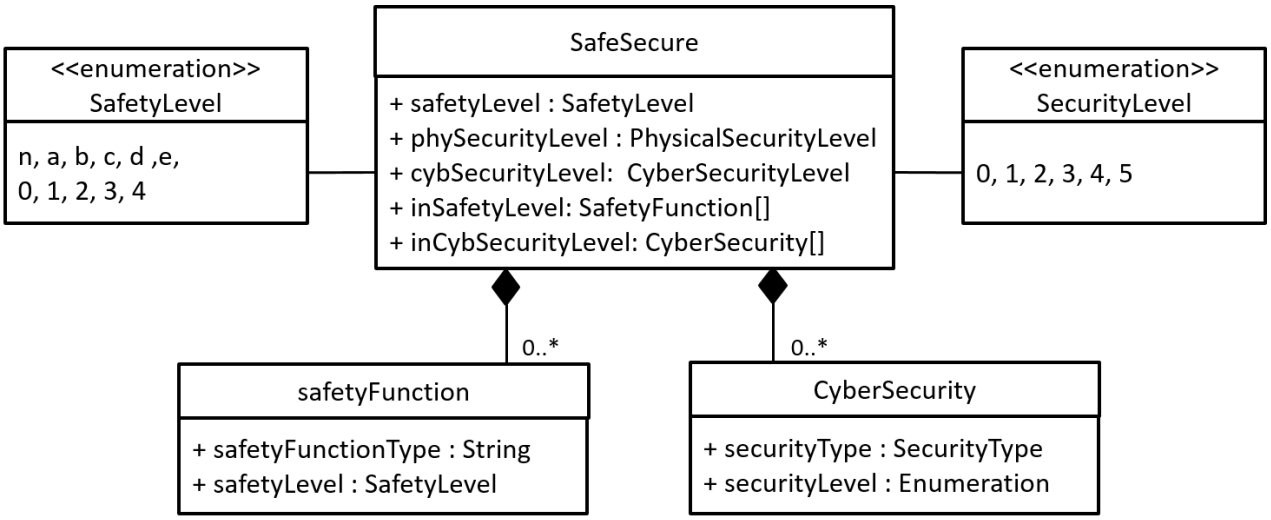


Figure 4.28 Relationship among Class diagrams for class SafeSecure

4.2.9 Class for Modelling of Module

The class Modelling shall provide the modelling-related information that the module supports for simulation, as shown in Figure 4.29 and Figure 30. Figure 4.31 shows the relationship among class diagrams for class Modelling, which are provided in Figure 4.29~Figure 4.30.

Information for Modelling shall provide information that the module provides for simulation, whose examples are 3D model and Universal Robotic Description Format (URDF) files. This information shall be provided as shown in Annex C. Module manufacturers can provide more than one file for simulation. If there are no URDF file and/or 3D model file, you shall fill the word "NA"(not available) in the corresponding item.

Class ModelCase
+ simulator : String // list of Simulation programs that a module support + mdf : String // path of model description file or path of directory including the files or “NA” + 3Dmodel: String // path of 3Dgraphic model file path of directory including the files or “NA”

Figure 4.29 Class diagram of ModelCase class

Class Modelling
Typedef modelCase[] modelCaseList; + simulationModel ModelCaseList;

Figure 4.30 Class diagram of Modelling class

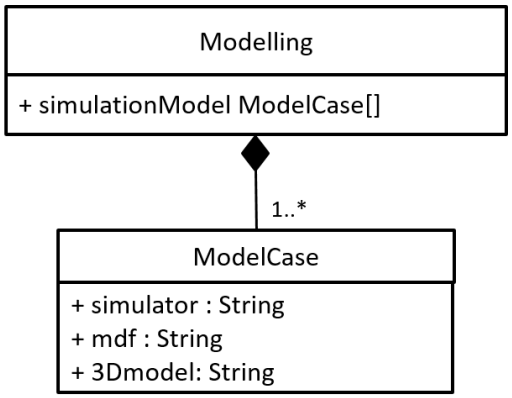


Figure 4.31 Relationship among Class diagrams for class Modelling

Annex A
(Normative)

Naming Rules

A.1 Introduction

These rules are derived from the guidelines and principles described in document ISO 11179-5. These rules have been modified to the usage of information model. In particular, the rules are designed to cover the naming of basic information entities and aggregated information entities.

Basic information entity is used in providing of information and building of information models and its examples are Object class, Attribute of object class, Tag Term, and Property Term. It is defined as Entry Name, which is the name of the entity derived from naming rules.

Aggregated information entity is defined as Name composing of Entry Names of basic information entities.

A.2 Naming rules

Rule 1: The Entry Name shall be unique in the logical data grouping to which the entry belong.

Rule 2: A Class name should 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 is less than 10 characters. And the maximum length of the Entry name is less than 256. An abbreviation for the entry name should be defined and used, if necessary.

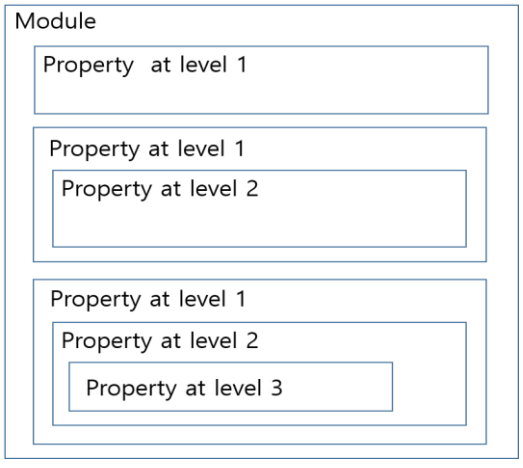
Rule 5: An Entry Name shall be separated by dot. Final element of an Entry Name shall be separated by dot to mark the end of Entry Name.

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 words and each word may start with a capital letter.

EXAMPLE: aggregate information



Possible Format

- moduleID.L1Attr
- moduleID.L1Attr.L2Attr
- moduleID.L1Attr.L2Attr.L3Attr
- moduleID.L1Attr.L2Attr.L3Attr.L4Attr

Figure A.1 Relationship among Module and property in each level

Annex B
Assignment Rule of Module ID (Normative)

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 shall be assigned by Universally Unique Identifier(UUID) rev.5 of IETF RFC4122. The remaining 5 elements shall be locally assigend by the vendor. The second element is the Product ID or PID, which shall consist of some important information of the module of 1 byte and a product ID of 3 bytes. The former shall provide the following 5 types of information, which is shown in Figure B.1: whether the module is composite or basic, whether SW aspects exist in the module, whether HW aspects exist in the module, whether the module provides the safety-related function, and whether the module provides the security-related function. The latter shall be the actual product ID, which is repesented 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 or SerNO, which 4 bytes long, whose value is set to"00000000" for software modules. The fifth element shall be module class ID, which 3 bytes long. The sixth element shall be the instance ID, or IID, of 1 byte long, which 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 the different instance IDs.

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

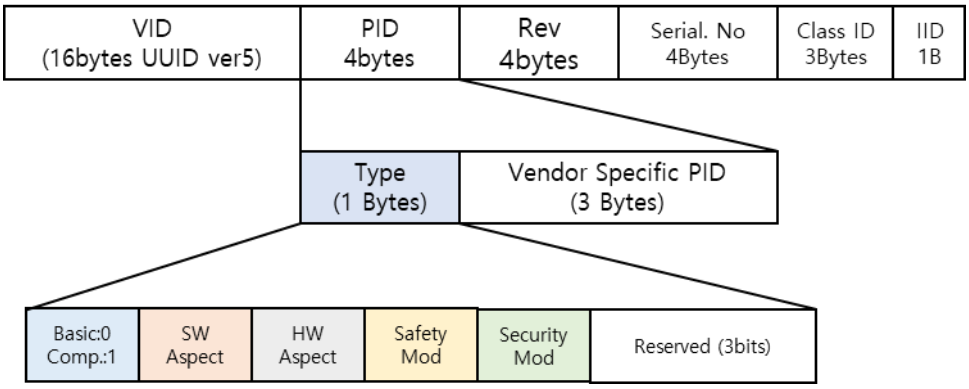


Figure B.1 Relationship among Module and property in each level

Name	Data Type	Length (Byte)	Note
VID	UUID	16	Vendor ID
PID	unsigned char	4	Product ID and properties
Rev	unsigned char	4	Revision number
SerialNo	unsigned long	4	Serial number
ClassID	unsigned long	3	Class ID
IID	unsinged char	1	Instance ID

Table B.1 Data types of elements of module ID

Annex C
Representation of common information (Normative)

C.1 General

This Annex represents information to be used when creating an object for the class defined in Clause 4. The information is able to be represented in XML or JSON. In this document, the information is represented in XML. XML elements for information for Class CIM shall be used as shown in Figure C.1 and descriptions of the elements' names are provided in Table C.1. The sub-elements are described in C.2 ~C.8.

```
<?xml version="1.0" encoding="UTF-8" ?>       <-- version = "1.1" OK     -->
<Module>
  <GenInfo> ....     </GenInfo>                  <!-- See Table C.2 and Figure C.2-->
  < IDnType> .... </ IDnType>                  <!-- See Table C.3 and Figure C.3~ Figure C.4 -->
  <Properties> ...     </Properties>          <!-- See Table C.4 and Figure C.5-->
  <IOVariables> ... </IOVariables>            <!-- See Table C.4 and Figure C.5-->
  <Services> ... </Services>                   <!-- See Table C.6 and Figure C.7-->
  <Infra> ...     </Infra>                        <!-- See Table C.7 and Figure C.8-->
  <SafeSecure> ... </SafeSecure>              <!-- See Table C.8 and Figure C.9-->
  <Modelling> ... </Modelling>                <!-- See Table C.9 and Figure C.10-->
</Module >
```

Figure C.1 Example of information for CIM model of a module in XML

Table C.1 XML element for 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 Information for General Information of module

XML elements for information for class GenInfo shall be used as shown in Table C.2. Examples of information for class GenInfo is provided as shown in Figure C.2.

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
Manufactures	Used for manufacturer's information of a module such as contact information and its name
Examples	Used for typical use case examples of a module

```
<?xml version="1.0" encoding="UTF-8" ?>       <-- version = "1.1" OK     -->
```

```
<GenInfo>
   <ModuleName> Module Name </ModuleName>                <!-- string -->
   <Description> Description of module     </Description>        <!-- string -->
   <Manufactures> manufacturer </Manufactures>                  <!-- string -->
   <Examples> list of use case </Examples>                          <!-- string -->
</GenInfo>
```

Figure C.2 Example of information for group GenInfo in XML

C.3 Information for Module ID

XML elements for information for class IDnType shall be used as shown in Table C.3. Examples of information for class IDnType is provided as shown in Figure C.3 and Figure C.4. Figure C.3 is an example of a basic module and Figure C.4 is an example of a composite module with both HW aspects and SW aspects. If only the element 'HWLIST' exists, the composite module is a kind of HW modules. If only the element 'SWLIST' exists, the composite module is a kind of SW modules. If two elements 'HWLIST' and 'SWLIST' exist at the same time, the composite module is a kind of HW-SW modules

Table C.3 XML elements for GenInfo

Element Name	description
ID	Used for Manufacturer's unique product reference number for a module. See Annex B. This element can have an attribute 'type" for providing of the module type such as a basic module or a composite module. If its value is "Bas" or the attribute is not defined, the module is a kind of basic modules. If its value is "Com", the module is a kind of composite modules.
HWlist	Used for set of modules with HW aspects in the composite module and list moduleIDs of the modules.
SWlist	Used for set of modules with SW aspects in the composite module and list moduleIDs of the modules.

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

Figure C.3 Example of information for Tag ID in XML that used in a basic module

```
<?xml version="1.0" encoding="UTF-8" ?>
< IDnType>       <!-- example of a composite module -->
   <ID type="Com"> moduleId     </ID>                    <!-- ID for the composite module -->
   <HWlist> <!-- in the list, IIDs of modules are not defined     -->
      <!-- in this example, 2 types of modules such as hwModuleID2 and hwModuleID3 are the same -->
      hwModuleID1 hwModuleID2 hwModuleID2 hwModuleID3 hwModuleID3 hwModuleID4
   </HWlist>
   <SWlist>
   <!-- in this example, 1 type of modules such as swModuleID1 are the same -->
      swModuleID1 swModuleID1 swModuleID1 swModuleID2 swModuleID3 swModuleID4
   </SWlist>
</ IDnType>
```

Figure C.4 Example of information for Tag ID in XML that used in a composite module

C.4 Information for Property, Inputs and Outputs

Most of Module Properties, Inputs and Outputs are related to execution of modules and used as parameters and/or variables in SW aspects. For HW aspects, some use cases are necessary to be provided and then examples of information for Inputs and Outputs will be given.

As information for the class Property defined in sub-clause 4.3.3, the name and the value of a property and the data type of the value is provided mandatory, and the unit of the value and the related description can be included for clarity of use.

Information for the class Variable defined in sub-clause 4.3.4 includes the name and the value of a property, the data type of the value, and the input/output type. And the unit of the value and the related description can be included for clarity of use. Especially, variables that can input and output simultaneously are marked in symbol 'input/output'.

XML elements for information of Property, Inputs and Outputs shall be used as shown in Table C.4.

Table C.4 XML elements for Property, Inputs and Outputs

<table>
<tr><th>Element Name</th><th colspan="2">description</th></tr>
<tr><td>Properties</td><td colspan="2">All related properties are defined within this element
Note: if the name space is needed, it will be defined in the -202 and -203.</td></tr>
<tr><td>Property</td><td colspan="2">Used to define an individual property in the element 'Properties'.
This element can have an attribute 'ComplexData" for providing of the array data type and the complex data types such as structure and class. If this attribute has "array" or "class", Property has an array or a class structure. If not defined, Property have a simple data type.</td></tr>
<tr><td>IOVariables</td><td colspan="2">All variables are defined within this element. The type of variables is classified into 3 types of input, output, and input/output.
This element can have an attribute 'Name" as domain name. Then variables defined in the element 'IOVariables' have the same domain.</td></tr>
<tr><td>Inputs</td><td colspan="2">Used for set of input variables</td></tr>
<tr><td>Outputs</td><td colspan="2">Used for set of output variables</td></tr>
<tr><td>InOuts</td><td colspan="2">Used for set of variables that can be simultaneously utilized both input and output variable</td></tr>
<tr><td>Input</td><td>Used for a variable assigned only as input variable in the element 'IOVariables'</td><td rowspan="3">The element can have an attribute 'ComplexData" for provide the array data type and the complex data types such as structure and class. If this attribute has "array" or "class", Variables have an array or a class structure. If not defined, Property have a simple data type.</td></tr>
<tr><td>Output</td><td>Used for a variable assigned only as output variable in the element 'IOVariables'</td></tr>
<tr><td>InOut</td><td>Used for a variables that can be simultaneously utilized both input and output variable in the element 'IOVariables'</td></tr>
<tr><td>name</td><td colspan="2">Name of a property or a variable used in a module
e.g. encoder, ratedCurrent , MaxCurrent, P_Coeff</td></tr>
<tr><td>value</td><td colspan="2">Used for initialization for a property or a variable</td></tr>
<tr><td>type</td><td colspan="2">Data type of a property or a variable; e.g. int16, float64, int32, int8, uint16, class, array. See Table C.5.</td></tr>
</table>

complex	Used to define an array or class of the data type of Table C.5
unit	unit of a property or a variable; e.g. Ampere, meter, Celsius, no unit, etc
description	Explanation of the related property or variable

Table C.4 shows the data types that shall be used in this document. If the element 'Type' is 'enum', the new attribute 'enum' shall be used. The element 'Unit' means the magnitude of physical quantity, whose examples are meter, rpm, sec, bps (bit per sec), ampere, volt, watt, and Celsius. In particular, the element 'Unit' has special units of 'none', which means no physical quantity.

- none : no unit.

Table C.5 values of attribute 'Type'

types	Description
boolean	True(1) or False(0)
int8	Signed integer of 1 byte
int16	Signed integer of 2 byte
int32	Signed integer of 4 byte
int64	Signed integer of 8 byte
uint8	unsigned integer of 1 byte
uint16	unsigned integer of 2 byte
uint32	unsigned integer of 4 byte
uint64	unsigned integer of 8 byte
float32	floating point of 4 byte
float64	floating point of 8 byte
enum	Enumeration type
array	Array type, a kind of complex data
class	Complex data type including structure and class.

Examples of information for class Property and IOVariable is provided as shown in Figure C.4.

```
<?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 -->
<-- 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>
        0, 0, 0, 10, 20, 10
    </value>
```

```
 </Property>       <--     example of array type property -->
<!-- following is example of class: -->
<!--       class cylinder {                    -->
<!--             float32 legnth;               -->
<!--             float32 radius                 -->
<!--        }                                          -->
 <Property name=”cylinder” type = “class”>
     <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=”controlValue” type=”float32” value=5.0 unit=ampere description = “current control
command to motor” / >     <!-- float32 controlValue=5.0     -->
   </Inputs>
   <Outputs>
      <Output name=”encoder” type =”uint32” unit = “none” description = “value of encoder” />
                <!-- uint32 controlValue     -->
   </Outputs>
   <InOut>     <!-- input and output -->
      <Inout name=”state” type=”uint8” unit = “none” description = “ reset or read status of motor”>
          <value = 0 />       <-- uint8 state=0     -->
        </inout>
         <Inout name=”pose” type = “class”>
           <-- class pose { float32 x,y,x ; float32 yaw, roll,pitch; }     -->
             <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”, type=”float32” unit=”degree” description = “pitch” />
       </Inout>       <--     example of class type property -->
   </InOutputs>
</IOVariables>
```

Figure C.5 Example of information for class Property and IOVariable in XML

C.5 Information for Services of Module

XML elements for information of Class Services shall be used as shown in Table C.6. An example of information for Services (or Functions/Capabilities) is provided as shown in Figure C.6 and Figure C.7. 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 argument used as the additional return value is the pointer in C/C++ or reference type in Java and Python.

Table C.6 XML elements for class Service

Element Name	description
Services	All properties for Servcies are defined within this element. This element describes the information about the Services (or Functions/Capabilities) of a module. The element can have an attribute ‘type” for providing the related interface definition file such as IDL.

```
typedef sequence<Octet> BufferType;
interface Service: CIMService
{
        uint8 initialize(int64 val1, float32 val2); // one of methods/services to be provided by module
        uint8 finalize(Long val1, Float val2, Long val3);
        uint8 read(Long fd, BufferType buf, UShort nbytes);
}
```

Figure C.6 IDL example for Services

```
<?xml version="1.0" encoding="UTF-8" ?>
<Services type = “IDL”>
      Path of IDL file.            <!-- see Figure 4.16 -->
</Services>
```

Figure C.7 Example of information for class Services in XML

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, whose examples are Ethernet, EtherCAT, CAN, USB and RS422. IP relates to IP code that the module provides according to IEC 60529. XML elements for information of infrastructure shall be used as shown in Table C.7. An example of information for class Infrastructure is provided as shown in Figure C.8.

NOTE 1 : Data bus for infrastructure means the bus on which modules are commonly used in the system.

Information for Databuses provides 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 Infrastructure are defined within this element. This element describes the information about the Infrastructure of a module.
Powers	Used to define Power that a module uses or provides. Power types are as follows: Electric power, Pneumatic power, and Hydraulic power. In ISO22166-203, the detail description will be provided.
ElecPower	Used to define Electric Power. Its attributes are value and unit. Value means the amount of power in unit, where unit is defined like “watt” or “W”,, or “mW”.
DataBuses	Used for defining of the data buses that a module use.
Databus	Used for defining of one data bus that a module use.
ConnectorType	Used for defining of connector type for data bus that a module use.
TypePhyMac	Used for defining of types of Physical and MAC protocol that a module uses. e.g. CAN, EtherCAT, Ethernet, RS232, RS485. The values will be defined in detail in ISO 22166-203.
TypeNetTrans	Used for defining of types of Network and Transport protocol that a module uses. e.g. IPTCP, CANopen, The values will be defined in detail in ISO 22166-202 or -203

TypeApp	Used for defining of types of Physical and Mac layer that a module uses. e.g. Modbus, POD, OBD, SYNC, The values will be defined in detail in ISO 22166-203.
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 a code of Ingress Protection of IEC 60529.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Infra>
  <Powers>
    <ElecPower value = 43 unit = “watt” />
  </Powers>
  <DataBuses>    <!-- list of data buses used in Module -->
   <Databus>
      <ConnectorType> DE9 </ConnectorType>       <!-- D-Sub 9-->
      <TypePhyMac> CAN </Type>
      <TypeNetTrans> CANopen </TypeNetTrans>
      <TypeApp> OBD NMT SDO PDO SYNC </TypeApp>    <!-- CANopen -->
      <Speed value = 1000 unit=”kbps” />    <!-- 1 Mbps, unit Kbps -->
   </Databus>
   <Databus>
      <ConnectorType> RJ45 </ConnectorType>       <!-- Jack for EtherNet -->
      <TypePhyMac> EtherCAT </Type>
      <TypeNetTrans> IPTCP </TypeNetTrans>
      <TypeApp> Modbus </TypeApp>
      <Speed value = 10000 unit=”kbps” />    <!-- 1 Mbps, unit Kbps -->
   </Databus>
  </DataBuses>
   <DBType> SQL DBMS </DBType> <!-- DBMS type    -->
  <IP> IP23CH    </IP>      <!--    Ingress Protection IEC 60529-->
  </Infra>
```

Figure C.8 Example of information for class Infra in XML

C.7 Information for Safety and Security

The information Safety and Security provides Safety Performance Level for each safety functions and Security-related information for each security function that the module supports. XML elements for information of Safety and Security shall be used as shown in Table C.8. An example of information for class SafeSecure is provided as shown in Figure C.9.

The tag PL shall have following enumeration values: [n, a, b, c, d, e] where n denotes no performance level and a~e denote PLa~PLe(ISO 13849-1). Cybersecurity has following enumeration values: [0, 1, 2, 3, 4] where 0 denotes no security measures and 1~4 denotes the security level(SL) 1 ~ SL 4(see IEC 62443-4-2). The measure for the physical security uses the method illustrated in sub-clause 5.6 of ISO 22166-1. The safety level for safety function shall be provided using SIL or PL.

Table C.8 XML elements for Class SafeSecure

Element Name	description
SafeSecure	All related attributes are defined within this element.

	This element describes the information about the safety and the security of a module.
Safety	Used to define the performance level of safety function that a module provides or the performance level of overall safety of a module if the module provides two or more safety functions.
Overall	Used for providing the overall performance level of safety or cyber security of a module
SafetyType	Used for providing the performance level of specified individual safety function. Using the attribute 'type', the individual safety function is specified and its values are shown in the column 'Tag type' in defined in Table 4.6.
Security	Used to define the security level of security function that a module provides or the level of overall security of a module if the module provides two or more security functions.
PhysicalSecurity	Used to define the physical security level of a module. The values are as follows: None, LatchSensor, LockwithKey, LockwithActuator
CyberSecurity	Used to define the security level of cybersecurity function
SecType	Used for providing the security level of specified individual security function. Using the attribute 'type', the individual security function is specified and its values are shown in the column 'Tag type' defined in Table 4.6.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SafeSecure>
   <Safety>
       <Overall> d </Overall>              <!-- choose one PL among a, b, c, d, e, and none.     -->
       <SafetyType type = "ESTOP ">     d </SafetyType> <!-- individual safety measure. see Table 4.5 --
>
      <SafetyType type = "SRSC " value = 'e' />
   </Safety>
   <Security>
      <PhysicalSecurity>
        <!-- None, LatchSensor, LockwithKey, LockwithActuator , see sub-clause 5.6 in ISO 22166-1 -->
         None
      </PhysicalSecurity>
      <CyberSecurity>       <!-- list security functions provided by module -->
          <Overall>       2     </Overall>     <!-- overall security level of a module -->
          <SecType type = "HU_IA">     2     </ SecType > <!-- individual security fct. see Table 4.10 -->
         < SecType type = "ACNT_MGT" value = 2     />     <!--     see Table 4.10 -->
      </CyberSecurity>
   </Security>
</SafeSecure>
```

Figure C.9 Example of information for class SafeSecure in XML

C.8 Information for Modelling

Information for Modelling provides information that the module provide for simulation, whose examples include a simulator type to be used, 3D model and 3D model Description Format Universal Robotic Description Format (URDF) files. XML elements for information of Safety and Security shall be used as shown in Table C.9. An example of information for class Modelling is provided as shown in Figure C.10.

Table C.9 Attribute for Simulation

Element Name	description
Modelling	All properties are defined within this element. This element describes the information about simulation modelling of a module. Note that a module can have one or more simulation model.
SimulationModel	Used for defining of a simulation model that a module provides. The simulation model consists of simulator (or simulation program), Model description format, and 3D model format.
Simulator	Used for defining of simulator (or simulation program). E.g. Gazebo, Webots, RoboDK, SimSpark, OpenRave
ModelFile	Used for defining of model description format(MDF) and 3D model format(3DF). The attribute 'type' can have one of following values. If type is one of MDFs such as URDF and SDF, the value is URL or path of MDF file that a module provides. Note that description format represents a module model, environment, visualization, and control. If type is one of 3DFs such as STL, OBJ, 3DS, DAE, and FBX, the value is URL or path of 3DF file that a module provides.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Modelling>
 <SimulationModel>    <!—the 1-st simulation model to be used -->
   <Simulator> Gazebo </Simulator>       <!-- Simulation programs. optional -->
   <ModelFile type=”URDF>    <!-- model description file type and its file path for model of module --
   >
      ../../modelDescrtionFileName <!-- the path of model description file    -->
   </ModelFile>
   <ModelFile type=”STL”>    <!-- 3D graphic model type and its file path for model of module -->
         ../../module3DGraphicFileName <!--the path of 3Dgraphic file model file    -->
   </ModelFile>
 <SimulationModel>
</Modelling>
```

Figure C.10 Example of information for Modelling in XML