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

Date: 2016-11-14

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

Committee identification: ISO/TC 299/WG 6

Secretariat: N/A

Modularity for service robots – Part 201 – Common Information Model for Modules

WD stage

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

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.

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 a common information model for modules based on FDIS 22166-1 Modularity for Service Robot Part 1-General Requirements, which is in FDIS stage and on development. Using the common information model, modules can be easily connected and exchange data among them. The common information model has been designed to make the interoperability, reusability, and automatic composability of modules increase.

Depending on the application, modules can be easily combined using the common information model, and safety requirements of systems or composite modules can be satisfied by utilizing the proper configuration of modules.

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

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

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

The information model includes some methods to get and put the illustrated characteristics. 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 module with software aspects and module with hardware aspects. Thus family applies to composite modules, particular types of robots e,g, mobile servant robots, physical assistant robots, personal carrier robots, and medical robots.

The standard presents robot module safety requirements and open robot inter-operability module guidelines for ensuring effective connectivity and correct functionality enabling published robot system safety requirements to be achieved.

The International Organization for Standardization (ISO) *[and/or] International Electrotechnical Commission (IEC)* draw[s] attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO *[and/or] IEC* take[s] no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO *[and/or] IEC* that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO *[and/or] IEC*. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO *[and/or] IEC* shall not be held responsible for identifying any or all such patent rights.

Modularity for service robots – Part 2-1 – Common information model for modules

1 Scope

This standard complies with the ISO 22166 family of standards providing requirements and guidelines on specifications on modularity for service robots. This Part 201 presents requirements and guidelines for common information models for modules of service robots.

This part presents how the CIM relates to interoperability and reusability and is taking into account of safety and security.

Examples of implementation and test methodologies are presented for modules for validation in typical intended use case applications.

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 DIS 22166-1, Modularity for service robots – Part 1 — General requirements

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

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
Unified Modelling Language
UML
Modelling language that helps the design of a system by visualizing its design.

3.4
module
component or assembly of components with defined interfaces accompanied with property profiles to facilitate system design, integration, interoperability, and re-use

3.5
property
attribute or characteristic of a module

3.6
software module
module whose implementation consists purely of programmed algorithms

3.7
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

3.8
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.

3.9
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.

3.10
Instance
Process in running or object used in a module

3.11
naming rules

3.12

4 Common information model for modules

4.1 General

The common information for modules shall be based on Annex A in FDIS-22166-1 and consist of items shown in Table 4.1. Naming of properties and classes used in information model shall follow the naming rules in Annex A.
Common information model (CIM) for processing the common information in Table 4.1 is illustrated in Figure 5.3. In table 4.1, the symbols 'M' and 'O' denote mandatory and optional meaning, respectively. CIM in Figure 4.1 is described in detail in Clause 5.

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	Hardware Aspects	M	M	-	
7	Software Aspects	M	-	M	
8	Module properties	M	M	M	ModuleProp
9	Inputs	M	M	M	IOVariable
10	Outputs	M	M	M	
11	Function(capabilities)	M	M	M	Service
12	Infrastructure	M	M	M	Infra
13	Safety/security	M	M	M	SafeSecure
14	Modelling	O	O	O	Modelling

Module Name is the name representing the module. Description provides the overview of the module, what the module is, what it does and how it can be used. Manufactures provides contact information for designer, developer, manufacturer of the module. Examples provides typical use cases of the module.

Module ID shall be the unique identifier of the module within a system and described in Annex B.

If the module is composed of two or more HW-SW modules, SW modules and/or HW modules, their module IDs are listed in HW Aspects and SW Aspects, respectively. Otherwise, Hardware aspects and Software aspects do not include anything.

Module properties are values that are generally used in initialization of modules. They can be however used during execution of modules if modification of the related value does not introduce the system-error. Module properties are classified into Mandatory and Optional ones.

Note 1: Environment constraints are also considered as properties, whose examples are operating temperature, operating humidity, and maximum allowed mechanical shock.

Example 1: Each coefficient used in the PID control algorithm is used once in initialization or is changed and used several times during execution of related SW modules.

'Inputs' means signals and/or data that feed into a module. 'Outputs' mean signals and/or data that come out of a module.

Example 2: Outputs of camera module are image data via USB. Inputs and outputs of a composite module 'servo motor with encoder' are motor control value and encoder value via EtherCAT, respectively. Inputs and outputs of a servo control SW module are encoder value and motor control values, respectively.

Note 2: Properties are kinds of input values from the viewpoint of modules but have to be distinguished in that Inputs are related to environment of modules but properties to parameters containing in modules. For examples, Inputs of the servo control SW module are encoder values but its properties are P, I, and D coefficients.

Modules' functions are provided to achieve services and shall be operated via their interfaces. In the software interface-related aspects, this item consists of service name, related-arguments, return value, and descriptions and can be defined similarly to format of application program interface. The arguments consist of zero or more pairs of a data type and a name. In the electrical/electronic interface-related aspects, the data bus type or the signal type and its related information used in the module shall be suggested. In the mechanical interface-related aspects, the mechanical interface and its related information used in the module shall be provided. The electrical/electronic interface-related aspects and the mechanical interface-related aspects are different from the software interface-related aspects in that the latter get directly the related services via provided interfaces and the former doesn't. That is, the former gets the services via physical interfaces or connections but not via interfaces defined in the information model. Hence interfaces for the former shall be provided to check whether the module can be used properly.

Example 3: 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 Annex 2.

Table 4.2 Example of function format

Name	Arguments	Return value	Description
Initialize	int16 val1, float64 val2	uint 8	Initialization using 2 arguments Return value: (0: success, negative value: error type)
	int16 val1, float64 val2, int32 val3	uint 8	Initialization using 3 arguments Return value: (0: success, negative value: error type)

Example 4: an example of function format for electrical/electronic aspect is shown in Table 4.3, where the arguments 'connectorType', 'keying', and 'busProtocol' mean the type of the connector, Male or Female, and the protocol type that the module use. The function is used to check the possibility that the peer module utilizes the electrical aspects of the module. Values for connector type can be USB-A, RJ45, DE-9, etc. A value for keying is one of male and female. Values of busProtocol can be USB, EtherNet, EtherCAT, RS232, etc.

Table 4.3 Example of function format for electrical/electronic adaptability

Name	Arguments	Return value	Description
CheckElecAdaptability	string connectorType, string keying, string busProtocol	boolean	Check electrical/electronic adaptability using 3 arguments Return value: (True: success, False: error)

Example 5: examples of function format for mechanical aspects is provided in Table 4.4. Like the electrical/electronic aspect, the function is used to check the possibility that the peer module utilize the mechanical aspects of the module. Unlike the electrical/electronic aspect, however, the function format for mechanical aspect is able to be more complicated due to huge variety. So, it should be considered as two simplified categories, joint and link.

Table 4.4 Example of function format for mechanical adaptability

Name	Arguments	Return value	Description
LinkAdaptability	vector origin, float mass, vector inertia, string shape, vector size, vector axis, string connection, vector collision	boolean	Check mechanical link adaptability using 8 arguments Return value: (True: success, False: error)
JointAdaptabilty	vector origin, vector axis, float limits, float damping, float friction	boolean	Check mechanical joint adaptability using 5 arguments Return value: (True: success, False: error)

Infrastructure lists infrastructure modules that the module has to use or to connect to.

Examples of infrastructure module may be power-type module, middleware-type module, the databus-type module that the modules are commonly utilizing. Power-type module describes the power supply type and provides the detailed information about the module power consumption. The middleware-type module describes the middleware used for the module, whose examples are ROS, OpenRTM, and OPRoS. Databus-type module has to be utilized common as a kind of system bus.

~~Note. The common information should be converted to the information suitable to the adopted middleware type.~~

~~Note: The power type for module is classified into electrical, hydraulic, and pneumatic type. The detailed information consists of one or more following pairs: (voltage used, ampere consumed).~~

Safety/Security describes the safety-related performance level and the security information provided by the module.

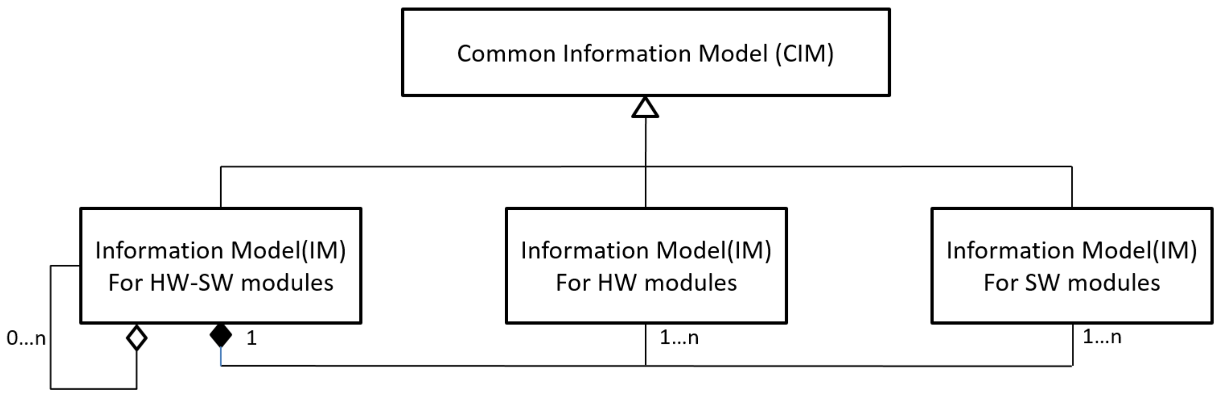
For the Safety/Security of the general service robot, the safety-related performance level is used for the module, which is defined in ISO 13849-1, and the security-related levels are listed for the module, where the security level is 0 ~ 4. The security level 0 means that there is no security measure. The security levels 1~4 are defined in IEC 62443-4-2. For the specific robot types such as a medical robot and a physical assistant robot, other safety-related standard and security standard should be utilized.

Modelling provides 3D model or similar model for simulation of the module.

4.2 Relationship between CIM and specific IMs

The common information model (CIM) can be described using the Unified Modelling Language (UML), which is a kind of object-oriented modelling techniques.

The common information model for modules shall be used in information models for all types of modules, which are HW-SW modules, SW modules, and HW modules. Their relationship is shown in Figure 4.1. The HW-SW modules composes of HW aspects and SW aspects. Most of modules are attached to modules with only hardware aspects, whose examples are mechanical frames, covers, and mechanical joint. The information model of a Robot as a module is illustrated in Figure 4.2. Figure 4.3 shows information model for a mobile robot based on Figure 4.2.



Common Information Model (CIM)
Information Model(IM) For HW-SW modules
Information Model(IM) For HW modules
Information Model(IM) For SW modules
0…n
1
1…n
1…n

Figure 4.1 Relationships among Information Models for modules

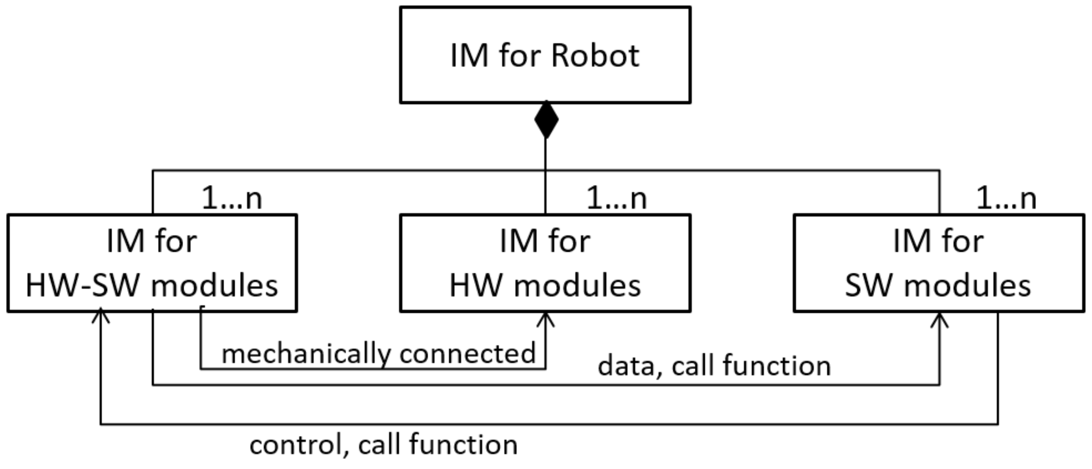


Figure 4.2 Example of Information models for a service robot

Examples: An example for Information model for a mobile robot is illustrated in Figure 4.3.

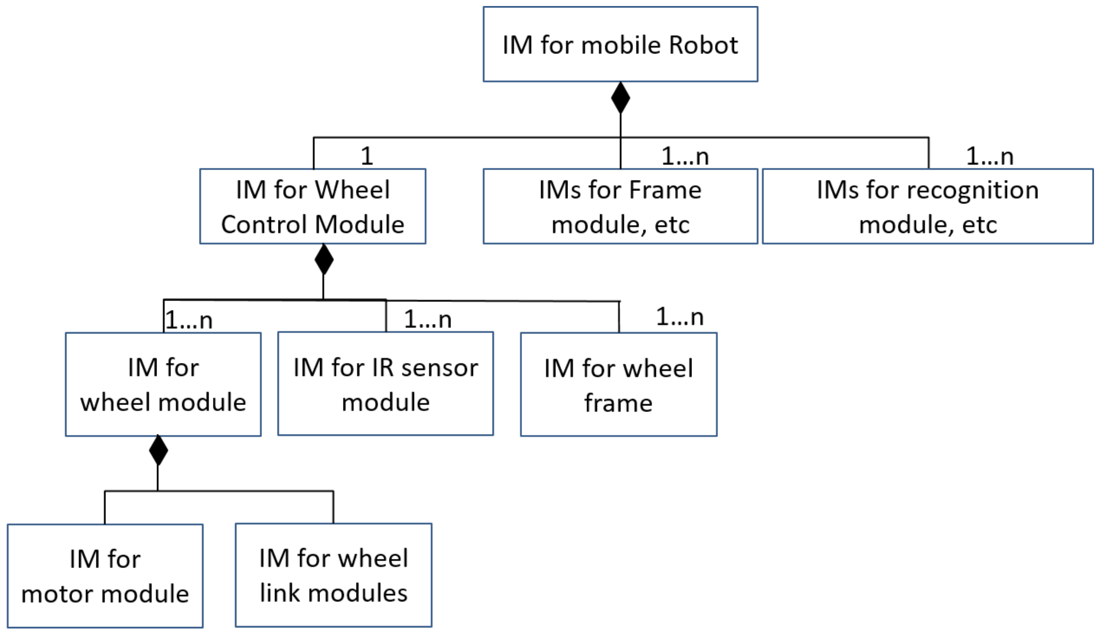


Figure 4.3 Relationship for Information models for a mobile robot

Sorry for my mic problem. Hence I will close this meeting. Is it OK? Next meeting, We will discuss more.

4.3 Class model of common information model

4.3.1 General

The common information model shall be grouped into 9 classes as illustrated in Figure 4.4, where 8 classes are suggested in Table 4.1 and the additional class is related to the status/health-related information of a module. The latter class is mainly used during execution of the module.

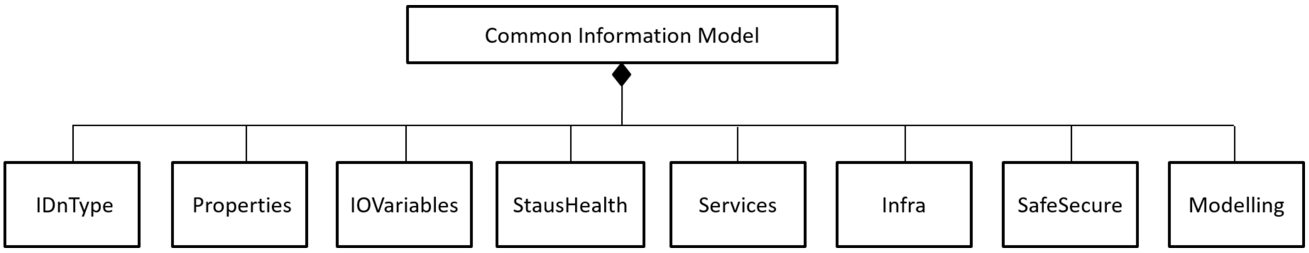


Figure 4.4 Class model of common information model

The class of the Common Information Model (CIM) shall have attributes given in Figure 4.5 and its attributes are based on DIS 22166-1. 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 standard is modified. The symbol oXXXX means the object variable. IDnType, Propertiees, IOVariables, StatusHealth, Services, Infra, SafeSecure, and Modelling in Figure 5.4 mean the class described in Figure 4.5 - Figure 4.15.

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.

Class CIM
- ModuleName : String
- Description : String
- Manufacturer : String
- Examples : String
- InformationModelVersion : String
+ oIDnType : IDnType
+ oProperties : Properties
+ oIOVariables : IOVariables
+ oStatusHealth : StatusHealth
+ oServices : Services
+ oInfra : Infra
+ oSafeSecure : SafeSecure
+ oModelling : Modelling
+ getModuleName() : String
+ putModuleName() : Boolean
+ getDescription() : String
+ putDescription() : Boolean
+ getManufacturer() : String
+ putManufacturer() : Boolean
+ getExamples() : String
+ put Examples() : Boolean

Figure 4.5 Class diagram of CIM class

4.3.2 Class for Module ID and Module Type

Information for Module ID and Module Type shall be defined in the class IDnType. Class IDnType shall have attributes given in Figure 4.6. Values of Attributes in Figure 4.6 are provided via Annex 2. When the same type of modules are used in a module, the modules shall be identified individually. For this purpose, an Instance ID is needed and generated uniquely by the module manager. IID shall be unique if IID is used. Hence genInstanceID() and setInstanceID() shall check whether the used IID is unique. getInstanceID() is used to get IID.

```
Class IDnType
---------------------------------------------------------------
+ moduleID : String
- IID : int32                    // Instnace ID
+ oHWAspects: Vector of 3-tuple ( ModuleType, moduleID, IID)
+ oSWAspects: Vector of 3-tuple ( ModuleType, moduleID, IID)
---------------------------------------------------------------
+ getInstanceID() : int32
+ setInstanceID(ID: int32 ) : Boolean
+ genInstanceID() : int32
```

Figure 4.6 Class diagram of class IDnType

4.3.3 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 4.8, which include information necessary for execution of the module and information about the operating environment or condition of the module. 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 2.

Example: Let’s consider an example provided in Figure 4.4, which is as follows:

```
<Property name=”maxRatedCurrent” type=”float32” unit=”ampere” description = “maximum of rated current for motor” value =15 />
```

From this tag, the following member variable and its data type are generated.
+ maxRatedCurrent = 15 : float32   or public float32 maxRatedCurrent=15

For other modules in order to access values of properties of a module, defined as variables, the following format shall be used: 4 fields are used and each field is concatenated using '_'.

<ModuleType>_ <ModuleID>_<InstanceID>. <Name defined in module>

Module ID and instance ID are provided in Figure 4.6. The Name of property as a variable is provided in Figure 4.7

The operating environment or condition means the condition in which the module with HW aspects is operating and the environment in which the SW module is executed. Examples of the former are the operating range of temperature, of humidity, of voltage, and of current. Examples of the latter are OS types, compiler type, and CPU type (or 8/16/32/64 bit). These values shall be provided using Figure 4.4. The 4 attributes of maxTemp, minTemp, maxHumidity, and minHumidity shall be generated using tag values in Annex 2 if those values are set.

In Figure 4.8, two types of properties are provided, which are the explicitly defined property-related variables and the explicitly undefined property-related variables. In the defined property-related variables, it is divided into hardware-related properties and software-related properties. If the property-related variables not defined, variable names for the properties are provided using Annex 2. To do this, the variable names are defined using the following class and applied to the class Properties.

```
Class Property
```

```
PropertyName : String     // attributes provided in the tag Property
PropertyName = initialValue : data type // if initial value is provided, initialValue is given.
                                          // In thid field, PropertyName is a kind of variable name.
-----------------------------------------------------------------------------------------------------

```
Figure 4.7 Class diagram of class Property

```
                                   Class Properties
-----------------------------------------------------------------------------------------------------
- properties[] : Property     // list of attributes generated according to the tag Property
- maxTemp : float32             // maximum value of temperature in operating environment
- minTemp : float32             // minimum value of temperature in operating environment
- maxHumidity : float32       // maximum value of humidity in operating environment
- minHumidity : float32       // minimum value of humidity in operating environment
- OpOS : Enumeration          // OS type in operation. Table 4.8
- UsedCompilerType : Enumeration       // compiler type to be used, Table 4.9
- OpCPUBit : Enumeration                  // the number of CPU bit to be used. 8, 16, 32, 64
-----------------------------------------------------------------------------------------------------
+ putProperty(propertyName: String, value: data type) : int16
+ getProperty(propertyName: String): data type
+ putMaxTemp(value: float32) : int16
+ getMaxTemp(): float32
+ putMinTemp(value: float32) : int16
+ getMinTemp(): float32
+ putMaxHumidity(value: float32) : int16
+ getMaxHumidity(): float32
+ putMinHumidity(value: float32) : int16
+ getMinHumidity(): float32
+ putOpOS(value: Enumeration) : int16
+ getOpOS():Enumeration
+ put UsedCompilerType(value: Enumeration) : int16
+ get UsedCompilerType():Enumeration
+ put OpCPUBit(value: Enumeration) : int16
+ get OpCPUBit():Enumeration
```
Figure 4.8 Class diagram of class Properties

4.3.4 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 4.9. The class IOVariables shall provide inputs, outputs and variables used in the module, which include information necessary for execution of the module. The detailed attributes such as the member name and the related data type shall be defined using values of the tag IOVariables in Annex 2.

For other modules in order to access values of Inputs, Outputs, and Share Variables used in a module, defined as variables, the following format shall be used: 4 fields are used and each field is concatenated using '_'.

<ModuleType>_ <ModuleID>_<InstanceID>. <Name defined in module>

Module ID and instance ID, and Module Type are provided in Annex or Figure 4.6. The Name of Input, Output, or Shared Variable is provided in Annex 2 or Figure 4.9.

Example: Let’s consider an example provided in Annex 2, which is as follows:

```
<IOVariables>
   <Inputs>
      <input name=”controlValue” type=”float32” unit=ampere description=“current control command to motor”
>
         <value/>
      </input>
```

```
    </Inputs>
    <Outputs>
       <output name=”encoder” type =”uint32” unit = “none” description = “value of absolute encoder” value= 0 />
    </Outputs>
    <InOutputs>     <!-- input and output -->
       <inout name=”state” type=”uint8” unit = “none” description = “ reset or read status of motor”>
           <value />
         </inout>
    </InOutputs>
 </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
+ encoder=0 : float32	or public float32 encoder=0	// value of absolute encoder
+ state : uint8	or public uint8 state	// reset or read status of motor

```
Class IOVarables
-----------------------------------------------------------------------------------------
+ InputName1 : data type1    // attributes generated according to the tag Input
...
+ InputNameN=initialVaule : data typeN    // attributes generated according to the tag Input
+ OutputName1 : data type1    // attributes generated according to the tag Output
...
+ OutputNameN=initialValue : data typeN    // attributes generated according to the tag Output
+ IOName1 : data type1    // attributes generated according to the tag InOutput as variable
...
+ IONameN=initialValue : data typeN    // attributes generated according to the tag InOutput as
variable
-----------------------------------------------------------------------------------------
```

Figure 4.9 Class diagram of class IOVarables

4.3.5 Class for Status and Health condition of Module

Information for Status and Health condition of Module shall be defined in the class StatusHealth in Fiure 4.10. StatusHealth class provides the status and/or health condition of a module which represents the health condition of the module if 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. The method setExecStatus() has to manage the status of execution life cycle. The method setErrorRecovery() is invoked by the safety manager and make the status transit to the inputted inStatus if the function is operating successfully. The variables ExecutionStatus and inStatus shall have one state in states shown in Figure 6 of DIS 22166-1.

Values used in each field will be defined. HelathCond, Error type

```
Class StatusHealth
-----------------------------------------------------------------------
- HealthCond : Enumeration
- ExcutionStatus : Enumeration
- ErrorType : int32
-----------------------------------------------------------------------
+ resetStaus() : Boolean
+ resetHealthCond(); Boolean
+ resetExecutionStatus(); Boolean
+ resetCommStatus(); Boolean
+ getHealthCond() : Enumeration value of Health condition or status of module
+ setHealthCond(inStatus: Enumeration value) : Boolean
+ getExecStatus() : Enumeration value of Execution life cycle status
+ setExecStatus(inStatus: Enumeration value) : Boolean
+ setErrorRecovery(inStatus: Enumeration value) : Boolean
+ getErrorType() : int32
+ resetErrorType() : Boolean
```

Figure 4.10 class diagram of StatusHealth class

4.3.6 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 4.11. The prototypes of methods of the module shall be defined using values of the tag Function in Figure Annex 2.

```
                              Class Services
+ NoOfBasicService : int8           // Number of Basic services provided
+ NoOfOptionalService : int8        // Number of Optional services provided
+ BasicServDes[] : String           // Descriptions of Basic services provided
+ OptionalServDes[] : String        // Descriptions of Basic services provided
-----------------------------------------------------------------------------
// basic(or Mandatory) Services
+ serviceName_b1(ArgName1: dataType1, .., ArgNameN: dataTypeN) : type of return data
..
+ serviceName_bM(ArgName1: dataType1, .., ArgNameN: dataTypeN) : type of return data
// optional(or Special) Services
+ serviceName_s1(ArgName1: dataType1, .., ArgNameN: dataTypeN) : type of return data
..
+ serviceName_sM(ArgName1: dataType1, .., ArgNameN: dataTypeN) : type of return data
```

Figure 4.11 Class diagram of Services class

Example: Let’s consider an example provided in Figure 4.6, which is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Services>
  <Function>
      <name>    initialize </name>              <!-- service/function name        -->
      <TypeBorO> BASIC    </TypeBorO>       <!-- enumeration : BASIC, OPTIONAL -->
      <arguments>
          <1st>
              <Aname> val1 </Aname>            <!-- argument name      -->
              <type> int16 </type>             <!-- data type of argument      -->
        </1st>
        <2nd>
              <Aname> val2 </Aname>
              <type> float64 </type>
       </2nd>
     <arguments>
      <ReturnValue type=”uint8” />      <!--    data type of return value     -->
      <Description>
          Initialization using 2 arguments Return value: (0: success, negative value: error type)
     </Description>
  </Function>
  <Function>
      <name>    initialize </name>
    <TypeBorO> OPTIONAL    </TypeBorO>      <!-- enumeration : BASIC, OPTIONAL -->
      <arguments>
          <1st>
```

```
                    <Aname> val1 </Aname>
                    <type> int16 </type>
             </1st>
             <2nd>
                    <Aname> val2 </Aname>
                    <type> float64 </type>
            </2nd>
             <3rd>
                    <Aname> val3 </Aname>
                    <type> int32 </type>
            </3rd>
        <arguments>
         <ReturnValue type="uint8" />
         <Description>
             Initialization using 3 arguments Return value: (0: success, negative value: error type)
        </Description>
    </Function>
</Services>
```

From these tags, the following member variables and the member methods are generated.

```
+ NoOfBasicService =1 : int8               // Number of Basic services provided
+ NoOfOptionalService=1 : int8            // Number of Optional services provided
+ BasicServDes[] : String                 // Descriptions of Basic services provided
+ OptionalServDes[] : String
// basic(or Mandatory) Services
+ initialize(val1: int16, val2 : float64) : uint8
// optional(or Special) Services
+ initialize(val1: int16, val2 : float64, val3:int32) : uint8
```

4.3.7 Class for Infrastructure of Module

Information for Infrastructure of Module shall be defined in the class Services. The class Infrastructure provides the types of infrastructure that the module supports, as shown in Figure 4.13. The prototypes of methods of the module shall be defined using values of the tag Function in Annex 2. The attributes powerIn and powerOut relates to power input to a module and power output from the module, respectively. The latter shall be described if exist.

Classes ElecPWR and DataBus used in the class Service are shown in Figure 4.12.

```
class ElecPwr {
   int16 count ; // the number of supplied powers
   class ElecComp {
      float32 volt;       // voltage
      float32 current;
   } pwr[count] ;
   void ElecPWR(int16 count)       // constructor
}
Class DataBus {
   String type;      // Physical/MAC protocol type used in module : CAN2.0, EtherCAT, Ethernet, RS485
   float32 speed; // transmission speed, unit : Kbps
   int16 initialize();    // overriding is needed.
   int16 open();    // overriding is needed.
   int16 read();    // overriding is needed.
   int16 write();// overriding is needed.
   int16 close();// overriding is needed.
   int16 control();    // overriding is needed.
}
--------------------------------------------------------------------------------
```

Figure 4.12 Class diagram of class ElecPWR and class DataBus

```
                              Class Services
--------------------------------------------------------------------------------
// Electric power type
+ powerIn : ElecPWR          // Input electric Power into a module.
+ powerOut : ElecPWR         // Output electric Power from a module, if any
// Pneumatic and Hydraulic types will be defined
// dataBuses
+ noBuses : int8        // the number of databuses used in module. Normal value is 1.
+ dataBus[noBuses] : DataBus
// types of DataBase mgt system
+ dbType : String   // DBMS type used in module. e.g oracle, sql, …
// IP code
+ IPcode : String                  // Ingress Protection IEC 60529
--------------------------------------------------------------------------------
```

Figure 4.13 Class diagram of Infrastructure class

4.3.8 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 4.15. Measure for the cybersecurity shall be specified using the security type and the security level (0~5) for the type. A module can include zero or more the measures. The measure for cybersecurity shall be defined using the class shown in Figure 4.14.

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

```
                    Class   CyberSecurity
--------------------------------------------------------------
- securityType : String       // Tag Name in Table 4.10
- securityLevel : Enumeration     // Cyber Security 0,1,2,3,4
--------------------------------------------------------------
```

Figure 4.14 class diagram of CyberSecurity

```
                            Class SafeSecure
--------------------------------------------------------------------------
- SafetyLevel : Enumeration        // SIL
- PhySecurityLevel : Enumeration   // Physical Security
- CybSecurityLevel[] : CyberSecurity
--------------------------------------------------------------------------
+ getSafetyLevel() : Enumeration
+ setSafetyLevel(inSL: Enumeration) :Boolean
+ getPhySecurityLevel() : Enumeration
+ setPhySecurityLevel(inSL: Enumeration) :Boolean
+ getCybSecurityLevel(securityType:String) : Enumeration // if securityType is found, return
                                                         // the corresspoding SL. Othewrwise return 0.
+ setCybSecurityLevel(cybSec: CyberSecurity) :Boolean
```

Figure 4.15 Class diagram of SafetySecurity class

4.3.9 Class for Modelling of Module

The class Modelling shall provide the modelling-related information that the module supports for simulation, as shown in Figure 4.16.

Information for Modelling shall provide information that the module provide for simulation, whose examples are 3D model and Universal Robotic Description Format (URDF) files. These information shall be provided as shown in Annex 2. If there are no URDF file and/or 3D model file, you shall fill the word "NA"(not available) in the corresponding item.

```
                        Class Modelling
+ simulator[] : String      // list of Simulation programs
+ mdf : String          // path of model description file or "NA"
+ 3Dmodel: String // path of 3Dgraphic file model file or "NA"
---------------------------------------------------------------

```

Figure 4.16 Class diagram of Modelling class

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: An Entry Name should start with a capital letter.

Rule 3: An Entry Name shall not contain consecutive redundant words.

Rule 4: An Entry Name shall be less than 10 characters. An abbreviation for the entry name should be defined and used, if necessary.

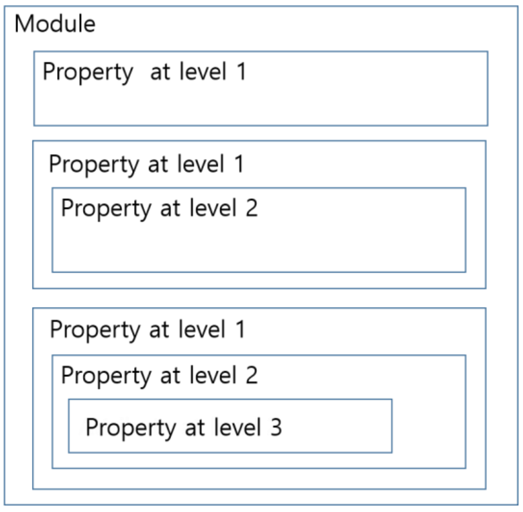
Rule 5: An Entry Name shall be separated by dot. Final element of an Entry Name shall be separated by dot to mark the end of Entry Name.

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

- Module.L1Attr
- Module.L1Attr.L2Attr
- Module.L1Attr.L2Attr.L3Attr
- Module.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 5 elements, which are illustrated in Figure A.2.1 and their types are given in Table A.2.1. The first element is the Vendor ID or VID, which shall be assigned by Universally Unique Identifier(UUID) of IETF RFC4122. The remaining 4 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 A.2.1: 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 ~~is 16 alphanumeric characters and is 16~~ 4 bytes long. The fifth 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 written in hexa code and/or alpha-numeric code.

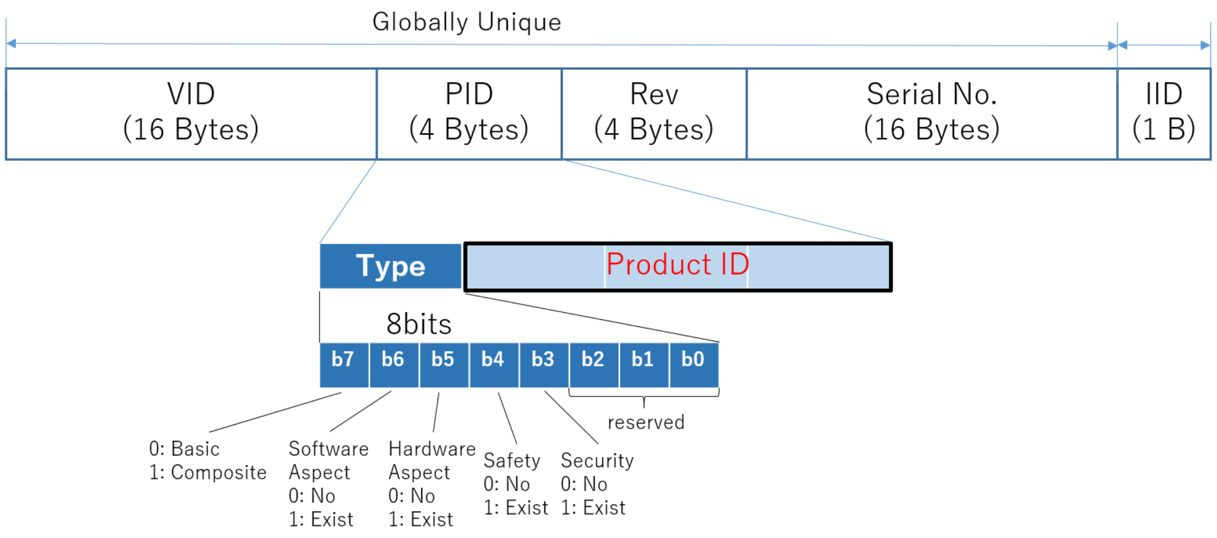


Figure A.2.1 Relationship among Module and property in each level

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	~~8~~ 4	Serial number
IID	unsinged char	1	Instance ID

Table A.2.1 Data types of elements of module ID

Annex C
Representation of common information (Informative)

C.1 General

Common information shall be represented in XML.

C.2 Information for General Information of module

Information for class GenInfo shall be provided as shown in Figure B.1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<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 B.1 Information for group GenInfo in XML

C.3 Information for Module ID

Information for class IDnType is provided as shown in Figure B.2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ID>
      <mid> module id      </mid>                              <!-- string provided by manufacturer -->
      <sequenceNo>       <!-- valid for composite module having the two or more same type modules--
>
           Enumeratio_number                                    <!-- Instance ID provided manually -->
      </ sequenceNo >
</ID >
```

Figure B.2 Information for Tag ID in XML

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 which the attributes in Table B.1 are used. Especially, variables that can input and output simultaneously are marked in symbol 'input/output'. Operating Environment or condition is provided in 'Property' item, if any.

Table B.1 Attribute for Property, Inputs and Outputs

Attribute	property	Inputs	Outputs	Input/Output	note

Name	M	M	M	M	e.g. encoder, ratedCurrent , MaxCurrent, P_Coeff
Data type	M	M	M	M	e.g. int16, float64, int32, int8, uint16
Unit	M	M	M	M	e.g. Ampere, meter, celsius, no unit, enum
Description	O	O	O	O	Explanation
value	O	O	O	O	For initialization

The attribute 'Name' means the name of property or Input or Output used in a module. The attributes 'Data Type' and 'Unit' represents the data type and the unit that property or Input or Output have. The data type chooses one in Table B.2.

Table B.2 types of attribute 'Data Type'

types	Size (in byte)	note
boolean	1	True(1) or False(0)
int8	1	Signed integer of 1 byte
int16	2	Signed integer of 2 byte
int32	4	Signed integer of 4 byte
int64	8	Signed integer of 8 byte
uint8	1	unsigned integer of 1 byte
uint16	2	unsigned integer of 2 byte
uint32	4	unsigned integer of 4 byte
uint64	8	unsigned integer of 8 byte
float32	4	floating point of 4 byte
float64	8	floating point of 8 byte
enum	4	Enumeration type

If the attribute 'Data Type' is 'enum', the new attribute 'enum' shall be used. The attribute '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 attribute 'Unit' has special units of 'none', which means no physical quantity.

- none : no data type

OS Type shall have the enumeration values listed in Table B.3. If the OS type has the value 'OTHER', the provider of the related XML file shall provide the additional information using the format illustrated in Figure 4.4.

Table B.3 enumeration values for OS type

Enumeration Value	Description
WIN32	32 bit windows OS
WIN64	64 bit windows OS

UBU32	32 bit Ubuntu OS
UBU64	64 bit Ubuntu OS
RED32	32 bit Red Hat OS
RED64	64 bit Red Hat OS
DEB32	32 bit Debian OS
DEB64	64 bit Debian OS
VxWorks	Real-time OS
XENOMAI	Real-time OS
PREEMPT	Real-time OS
OTHER	OS type except types listed above

Compiler Type shall have the enumeration values listed in Table B.4. If the compiler type has the value 'OTHER', the provider of the related XML file provides the additional information using the format illustrated in Figure B.5.   -> Make note

Table B.4 enumeration values for compiler type

Enumeration Value	Description
VCC32	32 bit Visual C++ compiler
VCC64	64 bit Visual C++ compiler
GCC32	32 bit GNU compiler in Linux
GCC64	64 bit GNU compiler in Linux
GPP32	32 bit GNU C++ compiler in Linux
GPP64	64 bit GNU C++ compiler in Linux
MGWGPP32	32 bit GNU C++ compiler in Windows
MGWGPP64	64 bit GNU C++ compiler in Windows
OTHER	Compiler type except types listed above

Values of the tag CPUbits shall have the following enumeration values: 8, 16, 32, 64.

The information related to the type of mechanical parts such as physical connection, mass, size and shape is defined in <Properties>, examples of which is shown in Figure B.3. The information related to the physical shape is defined using the tag <geometry>, the values of which are shown in Table B.5. The physical connection is related to the frame including bolting, the relationship between other hardware modules, the power type and data bus type. The mass, inertia, size and shape are related to the physical properties. The locking types in the physical connection are illustrated in Table B.6.

Table B.5 types of Tag <geometry> and their attributes

types	attributes
Cylinder	length, radius
Cuboid	length, height, width
Sphere	radius
Cube	length
TriangularPrism	length, height
HexagonalPrism	length, height

Table B.6 types of tag <connector> in tag <PhysicalConnection> and values

types	values
bolting	M2, M3, M4, M5, etc.

locking	Screw coupling with rtcheting, Bayobet coupling, Push-pull
Bolting with lock	

Information for class Property and IOVariable is provided as shown in Figure B.3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Properties>
  <Property name=”maxRatedCurrent” type=”float32” unit=”ampere” description = “maximum of
rated current for motor”     value =15 />
  <Property name=”maxRatedVoltage” type=”float32” unit=”volt” description = “maximum of rated
voltage for motor” >
       <value> 5 </value>
 </Property>
 <MechanicalProperty="basic module ">
    <Geometry>
        <Cylinder length="0.6" radius="0.2"/>
     </Geometry>
     <Material type="aliminium" />
    <PhysicalConnection>
          <connector="locking">
          <type = “male” />
          <size innerRadius = “10.0” outerRadius=”10.5.” unit = “milimetre” />
     </PhysicalConnection>
     <Connector>       <!-- power connector is included here if any -->
          <Databus connector="RJ45" cable="CAT.6"/>
    </Connector>
    <Inertia>
          <Mass value="10"/>
          <Inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4" iyz="0.0" izz="0.2"/>
     </Inertia>
 </MechanicalProperty>
 <OperatingEnv>         <!-- operating environment or condition is provided -->
     <Temperature>    <!-- pair of maximum and minimum values (HW aspect) -->
          maxValue, minValue         <!-- float type -->
    </Temperature>
    <Humidity>    <!-- pair of maximum and minimum values (HW aspect)-->
          maxValue, minValue         <!-- float type -->
    </ Humidity>
    <OStype> Enumeration Value </OStype>    <!-- OS type, Table B.4 -->
    <Compilertype> Enumeration Value </OStype>    <!-- compiler type to be used for module -->
    <CPUbits> Enumeration Value </CPUbits> <!-- the number of CPU bits to be used for module -->
 </OperatingEnv>
</Properties>
<IOVariables>
   <Inputs>
      <input  name=”controlValue”  type=”float32”  unit=ampere  description  =  “current  control
command to motor” >
          <value/>
      </input>
   </Inputs>
   <Outputs>
      <output name=”encoder” type =”uint32” unit = “none” description = “value of absolute encoder”
value = 0 />
```

```
   </Outputs>
   <InOutputs>    <!-- input and output -->
     <inout name=”state” type=”uint8” unit = “none” description = “ reset or read status of motor”>
        <value />
      </inout>
   </InOutputs>
</IOVariables>
```

Figure B.3 Information for class Property and IOVariable in XML

```
    <OStype> OTHER </OStype>    <!-- OS type including the number of bits -->
    <OtherOS> OS X    </OtherOS>    <!-- this tag exist if OStype has the value ‘OTHER’. -->
```

Figure B.4 Information of the tag OtherOS provided if the value of OStype is OTHER.

C.5 Information for Functions (Capabilities)

Information for Functions (Capabilities) is provided as shown in Figure B.5. 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.

Which services are included in Mechanical Functions?

```
<?xml version="1.0" encoding="UTF-8" ?>
<Services>
  <Function>
      <name>    initialize </name>               <!-- service/function name        -->
      <TypeBorO> BASIC    </TypeBorO>      <!-- enumeration : BASIC, OPTIONAL -->
      <arguments>
           <1st>
                <Aname> val1 </Aname>             <!-- argument name      -->
                <type> int16 </type>               <!-- data type of argument      -->
         </1st>
         <2nd>
                <Aname> val2 </Aname>
                <type> float64 </type>
        </2nd>
     <arguments>
      <ReturnValue type=”uint8” />     <!--    data type of return value    -->
      <Description>
          Initialization using 2 arguments Return value: (0: success, negative value: error type)
     </Description>
   </Function>
  <Function>
      <name>    initialize </name>
    <TypeBorO> OPTIONAL    </TypeBorO>      <!-- enumeration : BASIC, OPTIONAL -->
      <arguments>
           <1st>
                <Aname> val1 </Aname>
                <type> int16 </type>
         </1st>
```

```
            <2nd>
                  <Aname> val2 </Aname>
                  <type> float64 </type>
           </2nd>
            <3rd>
                  <Aname> val3 </Aname>
                  <type> int32 </type>
           </3rd>
        <arguments>
        <ReturnValue type="uint8" />
        <Description>
            Initialization using 3 arguments Return value: (0: success, negative value: error type)
        </Description>
    </Function>
   <Function>
        <name>     read </name>
      <TypeBorO> BASIC    </TypeBorO>      <!-- enumeration : BASIC, OPTIONAL -->
        <arguments>
              <1st>
                  <Aname> fd </Aname>          <!-- file descriptor -->
                  <type> int </type>             <!-- depending on OS and CPU -->
            </1st>
            <2nd>
                  <Aname> buf </Aname>
                  <type> voidp </type>            <!-- void pointer type -->
            </2nd>
            <3rd>
                  <Aname> nbytes </Aname>       <!-- number of bytes to read     -->
                  <type> uint </type>           <!-- depending on OS and CPU -->
           </3nd>
        <arguments>
        <ReturnValue type="uint8" />
        <Description>
            Initialization using 2 arguments Return value: (0: success, negative value: error type)
        </Description>
    </Function>
 </Services>
```

Figure B.5 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. This information is provided as shown in Figure B.6.

Note: 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 and examples of the supported API are initialize(), open(), read(), write(), close(), and control().

```
<?xml version="1.0" encoding="UTF-8" ?>
<Infra>
  <Powers>
    <PowerIn>
       <PowerType> Electric    </PowerType>    <!-- Electric, Pneumatic, Hydraulic    -->
       <consumedPower> 23    </consumedPower>    <!-- total power, unit: Watt-->
       <PowerType>
          <type Volt = 5 Ampere = 5 />       <!-- unit V, A -->
         <type Volt = 12 Ampere = 0.5 />       <!-- unit V, A -->
         <type Volt = 24 Ampere = 0.5 />       <!-- unit V, A -->
       </PowerType>
    <PowerIn>
    <PowerOut>       <!--     If there is no power from module, remove this item.     -->
      <PowerType> Electric    </PowerType>    <!-- Electric, Pneumatic, Hydraulic    -->
       <consumedPower> 9.2    </consumedPower>    <!-- total power, unit: Watt-->
      <PowerType>
         <type Volt = 5 Ampere = 0.4 />       <!-- unit V, A -->
        <type Volt = 12 Ampere = 0.2 />       <!-- unit V, A -->
        <type Volt = 24 Ampere = 0.2 />       <!-- unit V, A -->
     </PowerType>
   <PowerOut>
  </Powers>

  <DataBuses>     <!-- list of data buses used in Module -->
    <Databus>
      <Type> CAN </Type>
      <Speed> 1000 </Speed>      <!-- 1 Mbps, unit Kbps -->
      <API>
         <initialization> initialize() </Initialization>
         <open> open()     </open>
         <close> close()     </close>
         <readBlock> read()     </readBlock>
        <readChar> getc()     </readChar>
        <writeBlock> write()     </writeBlock>
        <writeChar> putc()     </writeChar>
        <control> control() </control>
     </API>
    </Databus>
    <Databus>
        <Type> EtherCAT </Type>
       ....
    </Databus>

    <Databus>
       <Type> RS422 </Type>
      .....
    </Databus>

  </DataBuses>
  <IP>       <!--     Ingress Protection IEC 60529-->
    <IPCode> IP23CH </IPCode>
  </IP>
  </Infra>
```

Figure B.6 Information for class Infra in XML

C.7 Information for Safety and Security

The information Safety and Security provides Safety Performance Level and Security-related information that the module supports. This information shall be provided as shown in Figure B.7.

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). As measures used for cybersecurity are various, types of measures provided by a module shall be assigned by using the tags illustrated in Table 4.10. The measure for the physical security uses the method illustrated in sub-clause 5.6 of ISO 22166-1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SafeSecure>
   <Safety>
      <PL> a </PL>       <!-- choose one in a, b, c, d, e, and none.     -->
   </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 -->
         <TagType 1>      </TagType 1>     <!--     see Table 4.10 -->
           ...
         <TagType N>      </TagType N>     <!--     see Table 4.10 -->
      </CyberSecurity>
   </Security>
</SafeSecure>
```

Figure B.7 Information for class SafeSecure in XML

Table B.7 tag types used for cybersecurity

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

C.8 Information for Modelling

Information for Modelling provides information that the module provide for simulation, whose examples are 3D model and Universal Robotic Description Format (URDF) files. These information is provided as shown in Figure B.8. If there are no URDF file and/or 3D model file, you fills the word "NA"(not available) in the corresponding item.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Modelling>
    <Simulator>    <!--list of Simulation programs used for 3D graphic model and model description --
>
        SimulatorName1 ....    SimulatorNameN    <-- e.g. gazebo or commercial simulation prog. -->
    </Simulator>
    <URDF>        <!-- model description file name for model of module -->
        ../../moduleDescrtionFileName // the path of model description file or NA e.g. urdf file, sdf file
    </URDF>
    <3Dmodel>    <!-- 3D graphic file name for model of module -->
        ../../module3DGraphicFileName // the path of 3Dgraphic file model file or NA
    </3Dmodel>
</Modelling>
```

Figure B.8 Information for Modelling in XML