January 2016

Hardware Abstraction Layer for Robotic Technology (HAL4RT)

Version: 1.0 – Beta 1

OMG Document Number: dtc/2016-01-01
Standard document URL: http://www.omg.org/spec/HAL4RT/1.0

This OMG document replaces the submission document (robotics/15-12-14, Alpha). It is an OMG Adopted Beta specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by January 8, 2016.

You may view the pending issues for this specification from the OMG revision issues web page http://www.omg.org/issues/.

The FTF Recommendation and Report for this specification will be published on December 16, 2016. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

OMG's ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/report_issue.htm.)

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG™s specifications implement the Model Driven Architecture (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG™s specifications include: UML® (Unified Modeling Language); CORBA® (Common Object Request Broker Architecture); CWM (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

OMG Headquarters
109 Highland Ave,
Needham, MA 02494 USA
USA

Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier` - 10 pt. Bold: Programming language elements.

Helvetica/Arial - 10 pt.: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1. Scope

This specification defines the Platform-Independent Model (PIM) of a Hardware Abstraction Layer for robotic systems that is capable to support at least the following devices:

- Sensors. Besides the actual, normalized measurement, sensor kind and unit of measure should be provided.
- Actuators. Commands to perform motions, and motion feedback information should be provided.

In addition this specification defines the Platform specific Model (PSM) in language C based on the HAL PIM.

This specification aims to enable engineers such as device makers, device users, and software users to build robotic software without any concern about the differences among the targeted devices, by standardizing the API of these devices.

Target readers of this specification include:

- Software engineers who use the HAL4RT to develop middleware and software.
- Device vendors and its engineers who develop devices and components which conforms to the HAL4RT.
- Engineers who are interested in robot and software development.

2. Conformance

An HAL4RT implementation conforms to this specification if and only if it implements the C PSM as specified in sub clause 7.3. The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,””SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMEND,” “MAY,” and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

3. References

3.1. Normative References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[UML]	Object Management Group, OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, 2015
[RTC]	Robotic Technology Component specification, http://www.omg.org/spec/RTC/1.1/
[RoIS]	Robotic Interaction Service specification, http://www.omg.org/spec/RoIS/1.0/

[SMART]	Smart Transducers specification, http://www.omg.org/spec/SMART/1.0/
[ISO/IEC-9899]	International Organization for Standardization, Programming languages – C, 1999

4. Terms and Definitions

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

Robotic Technology Component (RTC)

A logical representation of a hardware and/or software entity that provides well-known functionality and services.

Robotic Technology (RT)

Robotic Technology (RT) is a general term of the technology originating in robotics, and it means not only the standalone robot but technical element which constitutes robots.

Extensive Markup Language (XML)

A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

XML Metadata Interchange (XMI)

An OMG standard for exchanging metadata information via XML.

5. Symbols

There are no special symbols or terms.

6. Additional Information

6.1. Acknowledgements

The following organization submitted this specification:

- Japan Embedded Systems Technology Association (JASA)
  6-7 Nihonbashi Odenmacho, Chuo-ku, Tokyo, 103-0011 Japan
  Contact: Kenichi Nakamura (nakamura@upwind-technology.com)

The following organizations contributed to this specification:

- Central Information Center, Co., Ltd.

- CORE CORPORATION
- Dai-ichi Seiko Co., Ltd.
- ECS Co., Ltd.
- Hitachi, Ltd.
- Keio University
- NDD Corporation
- ORIENTAL MOTOR Co., Ltd.
- ROBOTEC, Inc.
- TDI Product Solution Co., Ltd.
- Tokyo Metropolitan Industrial Technology Institute
- Tokyo Metropolitan University
- Upwind Technology, Inc.

The following organizations supported this specification:

- Honda R&D, Co., Ltd.
- Japan Robot Association
- National Institute of Advanced Industrial Science and Technology
- Shibaura Institute of Technology

This page intentionally left blank.

7. Hardware Abstraction Layer for Robotic Technology (HAL4RT)

7.1. General

Hardware Abstraction Layer for Robotic Technology (HAL4RT) is an open standard for the implementation of robotics and control software systems.

Specifically, HAL4RT is an API (Application Program Interface) for the layer between on the first hand an application software of a middleware and on the other hand the drivers for devices such as sensor inputs, motor control commands and so on.

This standardized API increases the portability and reusability of these device drivers.

7.2. Format and Conventions

Class and Interface

Classes and interfaces described in this PIM are documented using tables of the following format:

Table 7.1 : <Class / Interface Name>

Description : <description>				
Derived From: <parent class>				
Attributes				
<attribute name>	<attribute type>	<obligation>	<occurrence>	<description>
…	…	…	…	…
Operations				
<operation name>	<description>			
<direction>	<parameter name>	<parameter type>		<description>
…	…	…		…

Note that derived attributes or operations are not described explicitly. Also, as the type of return code for every operation in this specification is Returncode, which is defined in Section7.3, Return Codes, this is omitted in the description table.

The ‘obligation’ and ‘occurrence’ are defined as follows.

Obligation

M (mandatory): This attribute shall always be supplied.

O (optional): This attribute may be supplied.

C (conditional): This attribute shall be supplied under a condition. The condition is given as a part of the attribute description.

Occurrence

The occurrence column indicates the maximum number of occurrences of the attribute values that are permissible. The followings denote special meanings.

N: No upper limit in the number of occurrences.

ord: The appearance of the attribute values shall be ordered.

unq: The appeared attribute values shall be unique.

Enumeration

Enumerations are documented as follows:

Table 7.2 : <enumeration name>

<constant name>	<description>
…	…

7.3. Return Codes

At the PIM level we have modeled errors as operation return codes typed ***ReturnCode***. Each PSM may map these to either return codes or exceptions. The complete list of return codes is indicated below.

Table 7.3: ReturnCode enumeration

HAL_OK	The operation completed successfully.
HAL_NO_CONNECTED	The target device is not connected.
HAL_NO_MEMORY	The target of the operation ran out of the memory needed to complete the operation.
HAL_NULL_PARAMETER	The parameter is not supported.
HAL_NOT_IMPLEMENTED	The operation is unsupported by the implementation (e.g., it belongs to a compliance point that is not implemented).
HAL_BAD_PARAMETER	The operation failed because an illegal argument was passed to it.

7.4. Platform Independent Model (PIM)

7.4.1 Overview

This section specifies the PIM for service interfaces and data models. HAL4RT has two layers: "Surface layer" and "Device layer".

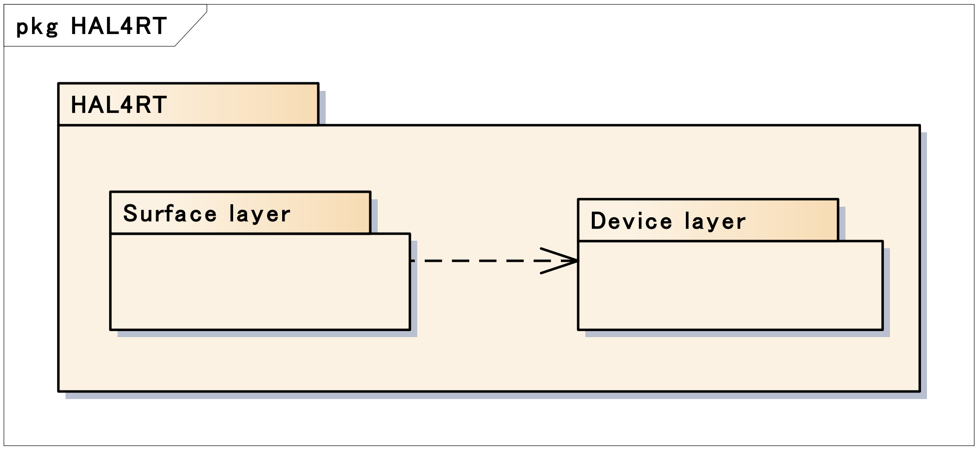

Figure 7.1 - Schematic structure of HAL4RT

The Surface layer provides standardized API (Application Program Interface) to application software and middleware. The Surface layer will so enable software developers to build application software and middleware without any concern about the differences among device interfaces she or he uses.

The Device layer consists of HAL4RT components. Application (including middleware) developers do not need to be aware of the presence of the Device layer.

Device suppliers and manufacturers provide HAL4RT component to their customer along with their actuators or sensors. The Device layer serves to bridge the gap between the API of the Surface Layer and the actual operation of the device.

[OPTIONAL] In addition, the Surface layer is not required because HAL4RT aims “light and compact” specification.

7.4.2 Interaction

7.4.2.1 System Interface

System Interface is the interface to notify the status of the device to the application program.

For examples, the following sequence diagram shows the behavior when the application program receives the status of the actuator by using system interface.

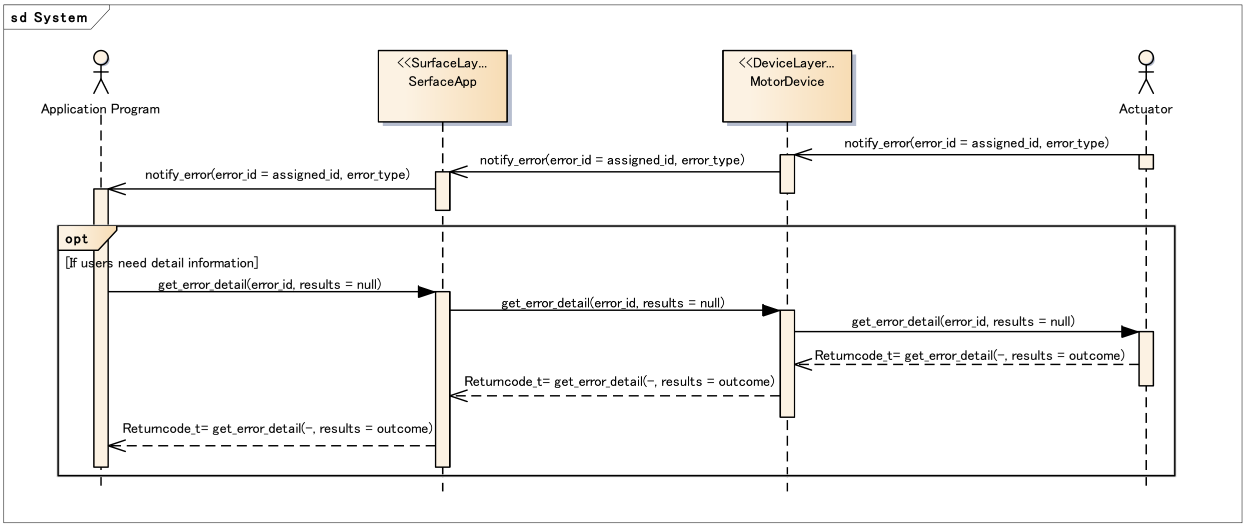

Figure 7.2 – Sequence diagram of System Interface (System Error)

When the error occurred in the actuator, the application receives the asynchronous error notification by using notify_error(). notify_error() notify the error_id and the error_type. The application program can know the detail of the error by using get_error_detail() and the error_id.

7.4.2.2 Command Interface

Command interface is the interface to execute the commands of the device from the application program.

For the examples, the following sequence diagram show the behavior to send the position command to the actuator, and to receive the position information from the actuator by using Command interface.

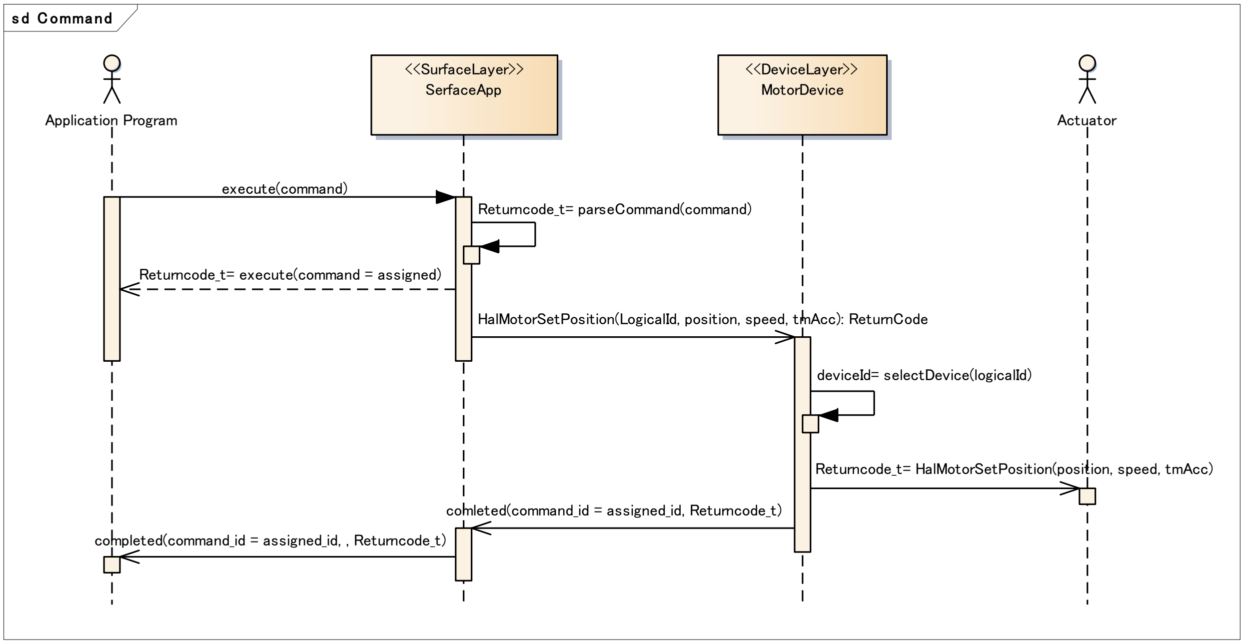

Figure 7.3 – Sequence Diagram of Command Interface (Motor position control)

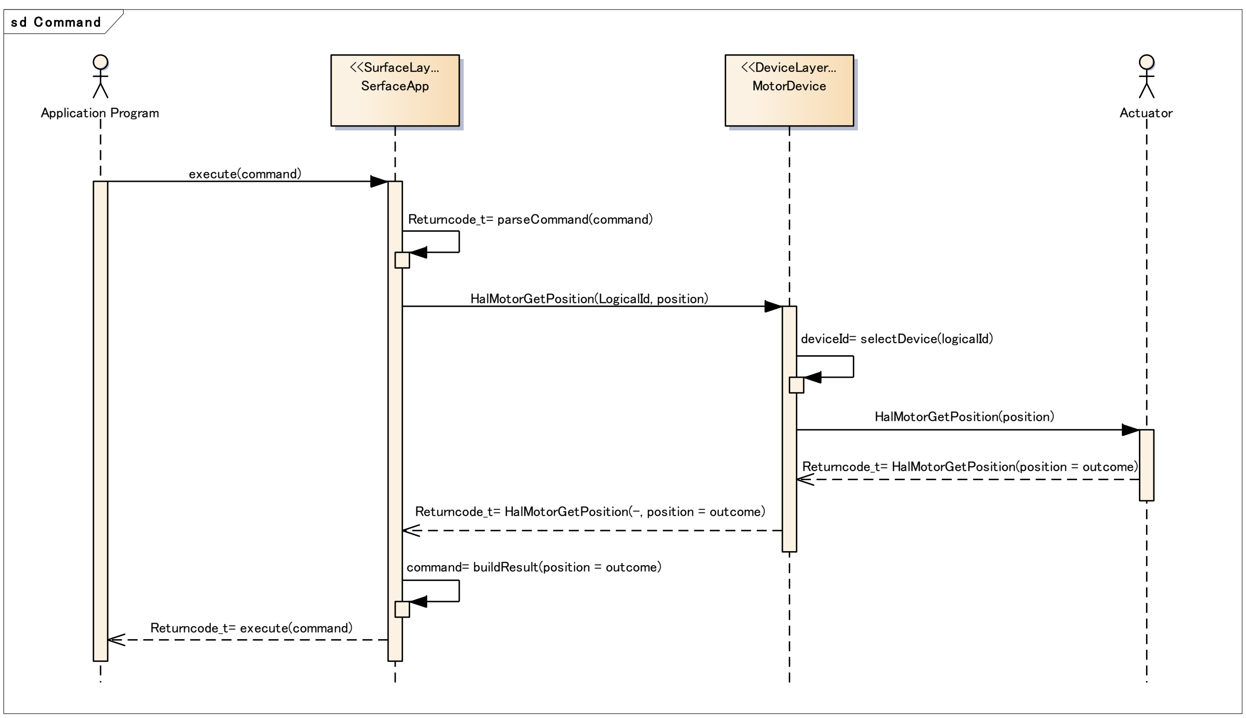

Figure 7.4 – Sequence Diagram of Command Interface (Get current motor position)

The application program specifies the command of the target device by using execute() in the surface layer. The surface layer parse the command and select the element in the device layer.

When the command is asynchronous, the surface layer returns command_id to the application program. After the command finished execution in the device, the device layer call completed() and notifies to the application program.

On the other hand, when the command is synchronous, the surface layer calls processing in the device layer. The device layer selects the target device, and calls the command. The surface layer returns the return value to the application program.

7.4.2.3 Event Interface

Event Interface is the interface to receive the notification of changing state of the device. This interface uses subscribe/unsubscribe to register/unregister the event.

The following sequence diagram shows the example.

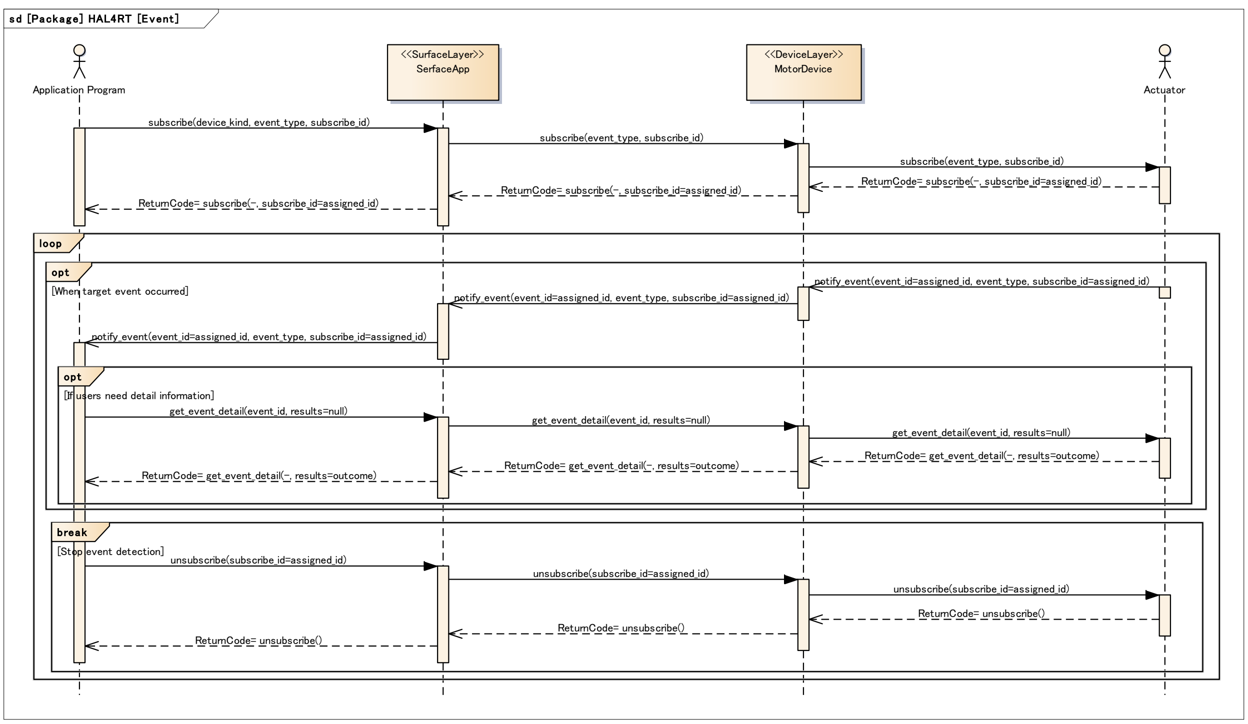

Figure 7.5 – Sequence Diagram of Event Interface

The application program registers the event by using subscribe(). The device returns the result and subscribe_id.

When the event occurs, the device notifies the application program and sends event_id by using notify_event(). The application program can find out the detail of the event using get_event_detail() and event_id.

The application calls unsubscribe() with subscribe_id to cancel the notification of the event.

7.4.3 Surface layer

The surface layer is the specification of the Application Programming Interface for the application programmer who uses HAL4RT devices. The surface layer parses the command from the application program, gets the device kind, and generates the logical id.

The following diagram is the PIM of the surface layer.

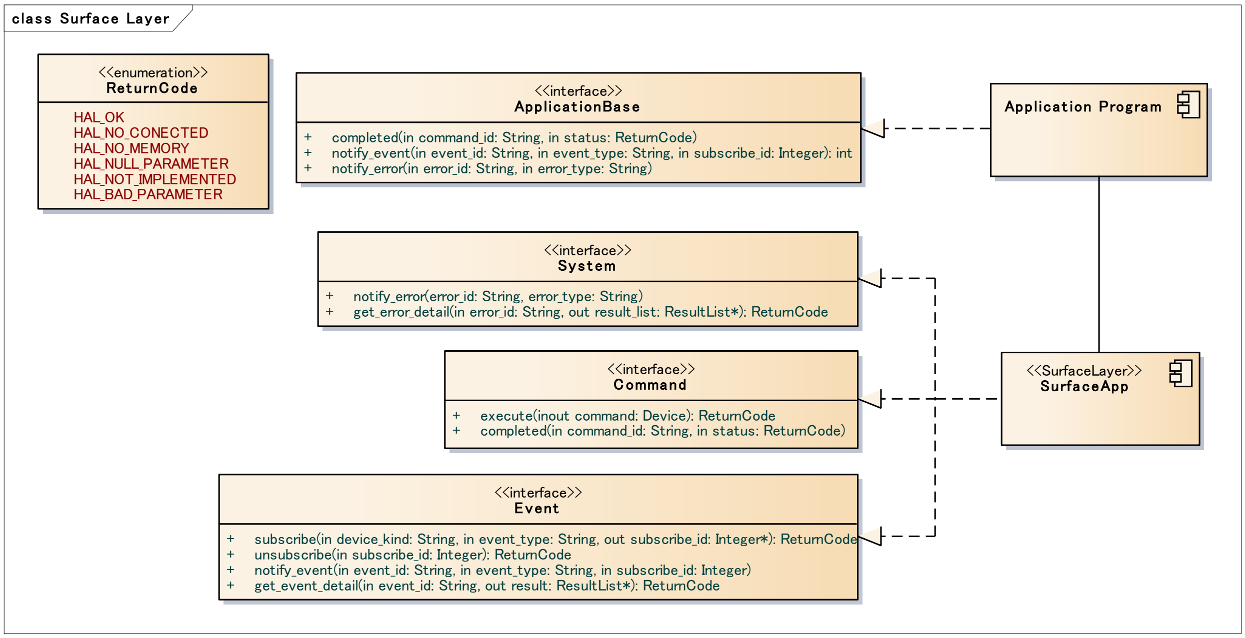

Figure 7.6 – Surface Layer

7.4.3.1 Interface for Surface Layer

Table 7.4: System Interface

<table>
<tr><td colspan="4">Description: The device notifies the error to the application program. Or, the application program gets the detail of the error.</td></tr>
<tr><td colspan="4">Derived From: None</td></tr>
<tr><td colspan="4">Operations</td></tr>
<tr><td colspan="2">notify_error</td><td colspan="2">Notify the error to the application program.</td></tr>
<tr><td>in</td><td>error_id</td><td>String</td><td>Id information to distinguish the error.</td></tr>
<tr><td>in</td><td>error_type</td><td>String</td><td>Information to distinguish the kind of the error.</td></tr>
<tr><td colspan="2">get_error_detail</td><td colspan="2">Get the detail of the error.</td></tr>
<tr><td>in</td><td>error_id</td><td>String</td><td>Id information to distinguish the error that was notified by notify_error.</td></tr>
<tr><td>out</td><td>result_list</td><td>ResultList</td><td>The detail information of the error.</td></tr>
</table>

Table 7.5: Command Interface

Description: The application program calls the command of the device.			
Derived From: None			
Operations			
execute	Execute the command of the device.		
inout	command	Device	The kind of the target device, the detail of the command, the result of the command.
completed	Notify the completion of the command.		
in	command_id	String	ID information of the command.
in	status	ReturnCode	The result of the command.

Table 7.6: Event Interface

Description: The device notifies the event to the application program.			
Derived From: None			
Operations			
subscribe	Register the event.		
in	device_kind	String	The kind of the device.
in	event_type	String	The type of the event.
out	subscribe_id	Integer	ID information of the registered event.
unsubscribe	Unregister the event.		
in	subscribe_id	Integer	ID information of the registered event.
get_event_detail	Get the detail of the event.		
in	event_id	String	ID information of the event.
out	results	ResultList	The detail information of the event.
notify_event	Notify the event.		
in	event_id	String	ID information of the event.
in	event_type	String	The type of the event.
in	subscribe_id	String	ID information of the registered event.

7.4.3.2 Interface for Application

The detail of the interface for the application program.

Table 7.7: Application Base Interface

Description: The application program implements this interface to receive the information from the device.			
Derived From: None			
Operations			
completed	Notify completion of the process in the device.		
in	command_id	String	The command ID.
in	status	ReturnCode	The result of command.
notify_event	Notify the event in the device.		
in	event_id	String	ID information of the event.
in	event_type	String	The type of the event.
in	subscribe_id	Integer	ID information of the registered event.
notify_error	Notify the error in the device.		
in	error_id	String	ID information of the error.
in	error_type	String	The type of the error.

7.4.4 Message Data Structure

When the application program executes the command of the device, it uses execute() in the command interface. The following diagram shows the data structure of the message that is the argument of execute().

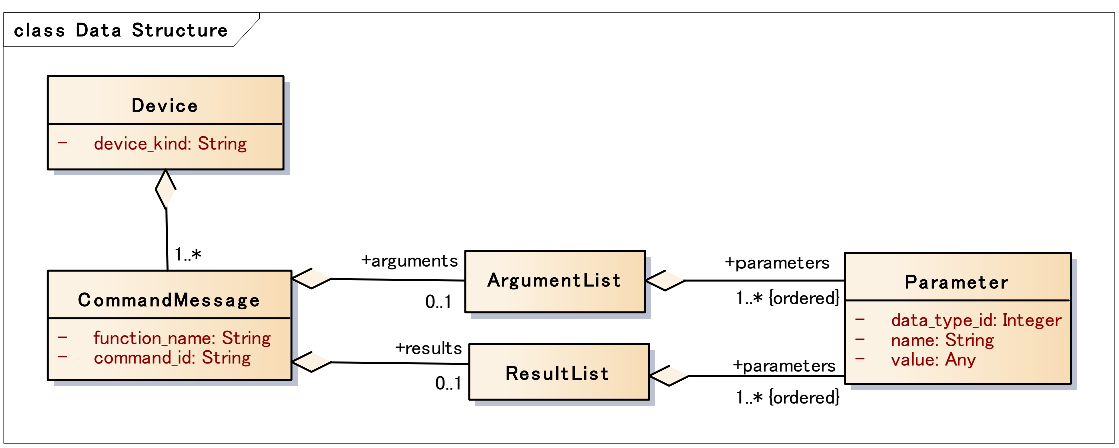

Figure 7.7 – Data Structure of Message

The following tables are the detailed definition of each class.

Table 7.8: Device

Description: Data type to distinguish the device.				
Derived From : None				
Attributes				
device_kind	String	M	1	The kind of the device. For examples, Motor, Encoder, Gyro sensor, Acceleration sensor, Force sensor, Torque sensor, Temperature sensor, Humidity sensor, GPS sensor, Direction sensor etc.

Table 7.9: Command Message

Description: Data type to specify the API and to keep the result.				
Derived From : None				
Attributes				
function_name	String	M	1	The name of API.
command_id	String	M	1	The ID information to distinguish the CommandMessage. The element of the surface layer select the ID and notify the application program. In the asynchronous command, this is used to notify the result.
arguments	ArgumentList	O	1	Parameter information for the command.
results	ResultList	O	1	Parameter information for the result of the command.

Table 7.10: ArgumentList

Description: Data type to keep the parameter.				
Derived From : None				
Attributes				
parameters	Parameter	M	Nord	Parameter information.

Table 7.11: ResultList

Description: Data type to keep the result.				
Derived From : None				
Attributes				
parameters	Parameter	M	Nord	The information of the result.

Table 7.12: Parameter

Description: Data type to send the parameter and to keep the result.				
Derived From : None				
Attributes				
data_type_id	Integer	M	1	The ID of data type.
name	String	M	1	Parameter name.
value	Any	M	1	Parameter value.

7.4.5 Device layer

The device layer is the specification of API for the device manufacturer. In the same kind of device, it hides the differences between manufacturers, models etc.. It defines the common API for each device type. It converts the actual device ID to the logical ID. The following diagram shows PIM of the device layer.

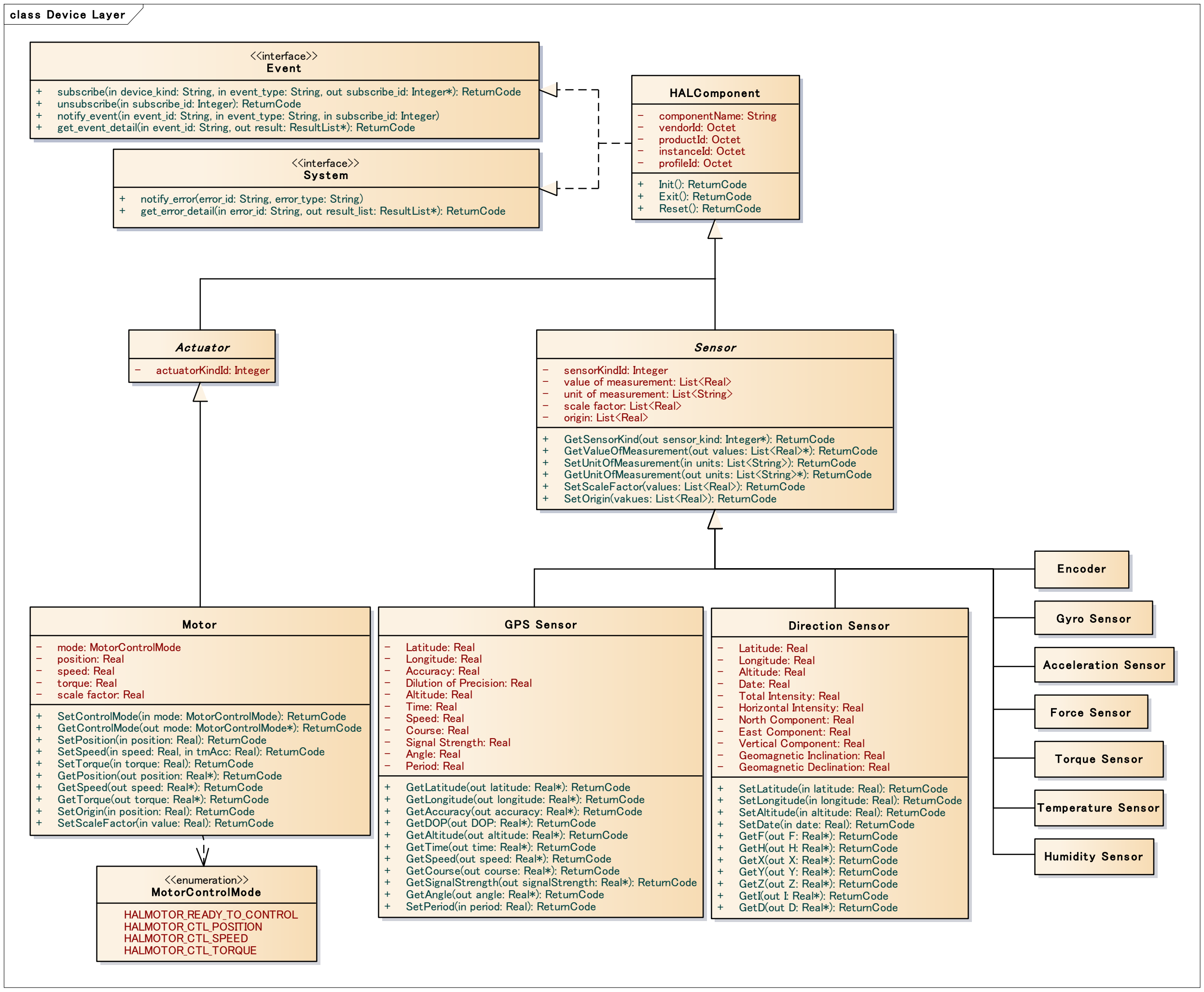

Figure 7.8 – Device Layer

Products conforming to HAL4RT, for each affected device must implement the entire API. If the nature of the product, an API that does not function exists, it will implement all of the API. (A motor which is corresponding to only the speed control in, or if it can not support the torque control API · Position Control API) And, non-support API will be implemented as you plan to return the "API unimplemented error (HAL_NOT_IMPLEMENTED)" as the API of the return value.

Event Interface and System Interface is an interface that is required only when you are using a surface layer. If the surface layer is omitted, it is not necessary to implement these interfaces.

7.4.5.1 HAL Component

HAL Component is an element to keep the common definition for all components in the device layer.

Derived From: None				
Attributes				
componentName	String	M	1	Component name.
vendorId	Octet [RTC]	M	1	Manufacturer ID. Manufacturer provides a component for the device.
productId	Octet [RTC]	M	1	Product ID defined by manufacturer.
instanceId	Octet [RTC]	M	1	Instance ID for some same devices.
profileId	Octet [RTC]	M	1	ID for the kind of the device.
Operations				
Init	Initialize HAL Component			
Exit	Finish HALComponent.			
Reset	Reset the error status and recover.			

componentName

Each HAL component is identified by a Profile ID (See the section 7.2.2), a Vendor ID (See the section 7.2.3) and a Product ID (See the section 7.2.4), and has a unique HAL component name.

Hal + *Profile name* + *Vender Name* + *Product Name*

vendorId

The Vendor names and Vendor IDs identify the device suppliers and manufacturers. A Vendor ID is defined as a 32bit unsigned integer type, with value between 0x00000000 and 0xFFFFFFFF. The Vendor names and Vendor IDs are maintained by the OMG. The Vendor name is a 2 to 16-character string beginning with an uppercase letter.

productId

The Product names and Product IDs identify the products of the device suppliers and manufacturers. A Product ID is defined as a 32bit unsigned integer type, with values between 0x00000000 and 0xFFFFFFFF. The Product names and Product IDs are defined by device suppliers and manufacturers

instanceId

An Instance ID identifies a specific device. An Instance ID is defined as a 32bit unsigned integer type, with values between 0x00000000 and 0xFFFFFFFF. The Instance IDs are defined by the application developers.

profileId

A Profile ID identifies the kind of the device.

Device kind ID Registry

A register to manage Device Kind ID, Device Name, componentName, vendorId, productId, instanceId and profileId.

For examples,

Device Kind ID, Device Name

0x00000001, Motor

0x00000002, Encoder

0x00000003, GyroSensor

0x00000004, TorqueSensor

Vendor ID, Product ID, Component Name

0x00000001, 0x00000001, AAA

0x00000002, 0x00000001, BBB

Device Characteristics Profile

A profile that will have the kind of the device, command name, command ID, and parameters.

Device Characteristics Profile Definition

A register to have componentName, vendorId, productId, instanceId and profileId.

State Machine

The following diagram shows State Machine of HAL Component. State Machine of the device is defined in "Active".

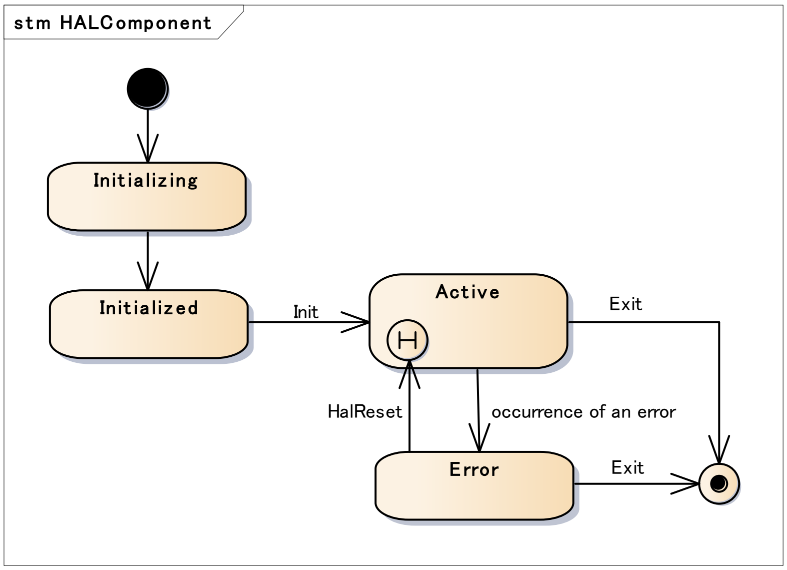

Figure 7.9 – State Machine of HAL Component

The detail of each state is below.

- Initializing: The status to initialize the device. The application program cannot call the methods of HAL Component.
- Initialized: The status after initializing of the device. The application program can call only Init.
- Active: The state that HAL Component is working. The application program can call all API except Init and Reset.
- Error: The state that HAL Component is stopping. The application program can call Reset and Exit.

7.4.5.2 Actuator

Actuator is an element that has the common definition of the actuator.

Derived From: HAL Component				
Attributes				
actuatorKind	Integer	M	1	ID that shows the kind of the actuator.
Operations：None				

7.4.5.3 MotorControlMode

MotorControlMode is an enumeration to distinguish the control mode of the motor.

HALMOTOR_READY_TO_CONTROL	No power mode.
HALMOTOR_CTL_POSITION	Position control mode.
HALMOTOR_CTL_SPEED	Speed control mode.
HALMOTOR_CTL_TORQUE	Torque control mode.

7.4.5.4 Motor

Motor is an abstract element that shows 1 DOF (degree of freedom) motor including linear and rotary. HAL4RT doesn't support Multiple DOF motors.

Derived From: Actuator				
Attributes				
mode	MotorControlMode	M	1	The control mode of the motor.
position	Real	M	1	The current position/angle.
speed	Real	M	1	The current velocity/angular velocity.
torque	Real	M	1	The current force/torque.
scale factor	Real	M	1	Unit(the scale of position/angle of output shaft) Rotating system：The value per round. The initial value is 2 pi. Linear system：The value per meter. The initial value is1.0.

Operations				
SetControlMode		Set control mode of the motor.		
in	mode	MotorControlMode	M	Control mode.
GetControlMode		Get control mode of the motor.		
out	mode	MotorControlMode	M	The current control mode.
SetPosition		Move to the target position/angle. If the control mode is not position control mode, change to position control mode.		
in	position	Real	M	The target position. Unit: scale factor.
SetSpeed		Increase/Decrease velocity/angular velocity in the specified acceleration/deceleration time. If the control mode is not speed control mode, change to speed control mode.		
in	speed	Real	M	The target speed. Unit:scale factor/second.
in	tmAcc	Real	M	Acceleration/Deceleration time to the target speed. Unit:second.
SetTorque		Output the specified torque. If the control mode is not torque control mode, change to torque control mode.		
in	torque	Real	M	The target torque. Unit:[Nm] or [N].
GetPosition		Get the current position/angle.		
out	position	Real	M	The current position/angle. Unit:scale factor.
GetSpeed		Get the current velocity/angular velocity.		
out	speed	Real	M	The current speed. Unit:scale factor/second
GetTorque		Get the current torque.		
out	torque	Real	M	The current torque. Unit:[Nm] or [N]
SetOrigin		Set the current position as the origin.		
in	position	Real	M	The target position. Unit:[rad] or [m]

State Machine

The following diagram shows State Machine of Motor. It shows the State Machine in the Active state of HAL4RT Component.

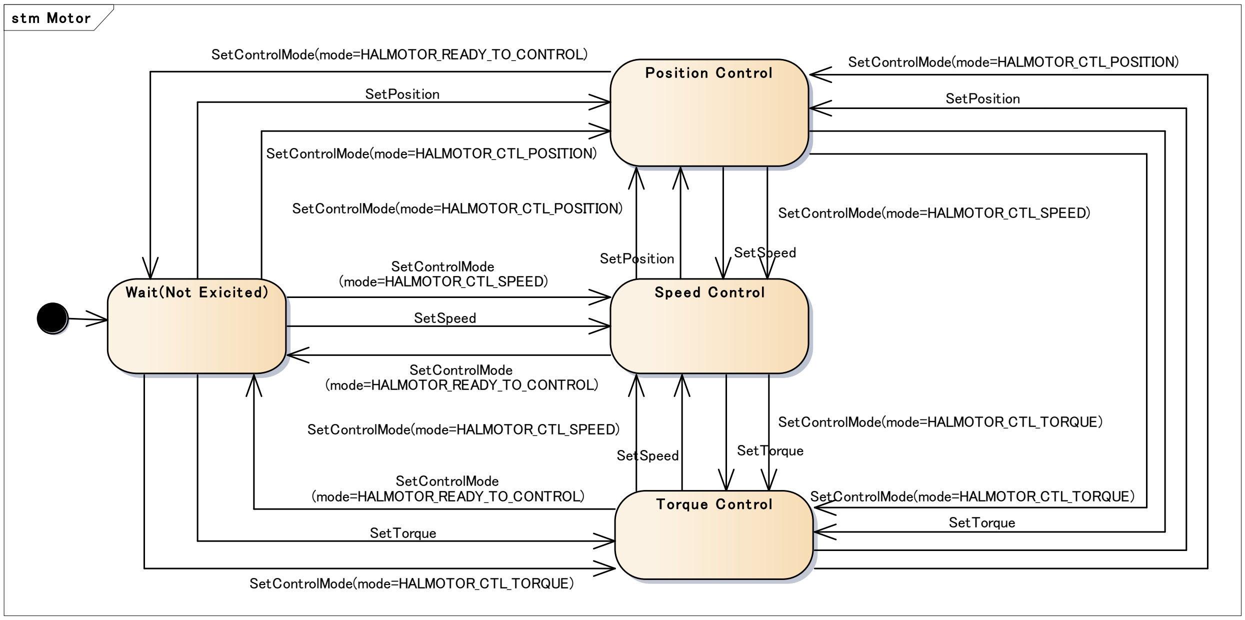

Figure 7.10 – State Machine of Motor

The details of each state are below.

- Wait (Not Excited): Standby state that motor body is not energized.
- Position control: The state running position control. If SetPosition is called, change to this state automatically.
- Speed Control: The state running speed control. If SetSpeed is called, change to this state automatically.
- Torque Control: The state running torque control. If SEtTorque is called, change to this state automatically.

[OPTIONAL] It is not necessary to implement all control modes.

7.4.5.5 Sensor

Sensor is an abstract element that has the common definition of the sensor.

Derived From: HAL Component				
Attributes				
sensorKindId	Integer	M	1	ID information to distinguish the kind of the sensor.
value of measurement	Real	M	Nord	Value of measurement of the sensor.
unit of measurement	String	M	Nord	Unit of measurement.
scale factor	Real	M	Nord	Factor to convert the measurement content of the sensor to the specified measurement information.
origin	Real	M	Nord	The origin of each measurement
Operations				
GetSensorKind	Get the kind of the sensor.			
out	sensor_kind	Integer	M	The kind of the sensor.

GetValueOfMeasurement		Get the value of measurement of the sensor.		
out	values	List<Real>	M	Value of measurement of the sensors.
SetUnitOfMeasurement		Set the unit of sensor measurement.		
in	units	List<String>	M	Unit of sensor measurement.
GetUnitOfMeasurement		Get the unit of sensor measurement.		
out	units	List<String>	M	Unit of sensor measurement.
SetScaleFactor		Set the scale factor of the value of measurement of the sensor.		
in	values	List<Real>	M	Scale factor of the value of measurement of the sensor.
GetScaleFactor		Get the scale factor of the value of measurement of the sensor.		
out	values	List<Real>	M	Scale factor of the value of measurement of the sensor.
SetOrigin		Set the origin of the sensor.		
in	values	List<Real>	M	Origin of the sensor.

7.4.5.6 Encoder

Encoder is a sensor to measure the position / angle.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.7 Gyro Sensor

Gyro Sensor is a sensor to measure angular velocity.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.8 Acceleration Sensor

Acceleration Sensor is a sensor to measure acceleration.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.9 Force Sensor

Force Sensor is a sensor to measure force.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.10 Torque Sensor

Torque Sensor is a sensor to measure torque.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.11 Temperature Sensor

Temperature Sensor is a sensor to measure temperature.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.12 Humidity Sensor

Humidity Sensor is a sensor to measure humidity.

Derived From: Sensor
Attributes: None
Operations: None

7.4.5.13 GPS Sensor

GPS Sensor is a sensor to measure latitude and longitude, or heading and distance to GPS position.

Derived From: Sensor				
Attributes				
latitude	Real	M	1	Latitude
longitude	Real	M	1	Longitude
accuracy	Real	M	1	Accuracy
dilution of precision	Real	M	1	Dilution of precision (DOP)

altitude	Real	M	1	Altitude
time	Real	M	1	Time
speed	Real	M	1	Speed
course	Real	M	1	Course
signal strength	Real	M	1	Signal strength
angle	Real	M	1	Angle
period	Real	M	1	Period
Operations				
GetLatitude	Get the latitude, in degrees.			
out	latitude	Real	M	Latitude
GetLongitude	Get the longitude, in degrees.			
out	longitude	Real	M	Longitude
GetAccuracy	Get the estimated accuracy of this location, in meters.			
out	accuracy	Real	M	Accuracy
GetDOP	Get the value of DOP.			
out	DOP	Real	M	Dilution of precision (DOP)
GetAltitude	Get the altitude if available, in meters above the WGS 84 reference ellipsoid.			
out	altitude	Real	M	Altitude
GetTime	Return the UTC time of this fix, in milliseconds since January 1, 1970.			
out	time	Real	M	Time
GetSpeed	Get the speed if it is available, in meters/second over ground.			
out	speed	Real	M	Speed. Unit : [m/s]
GetCourse	Get the course if it is available, in 000.0–359.9 degrees.			
out	course	Real	M	Course. Unit : [degree]
GetSignalStrength	Get signal strength.			
out	signal strength	Real	M	Signal strength
GetAngle	Get angle.			
out	angle	Real	M	Angle
SetPeriod	Set period.			
in	period	Real	M	Period

7.4.5.14 Direction Sensor

Direction Sensor is a sensor to measure direction.

Derived From: Sensor				
Attributes				
latitude	Real	M	1	Latitude. -90.00 to +90.00 degrees
longitude	Real	M	1	Longitude. -180.00 to +180.00 degrees
altitude	Real	M	1	Altitude. referenced to the WGS 84 ellipsoid OR the Mean Sea Level (MSL)
date	Real	M	1	Date. 2015.0 to 2020.0
total intensity	Real	M	1	F - Total Intensity of the geomagnetic field
horizontal intensity	Real	M	1	H - Horizontal Intensity of the geomagnetic field
north component	Real	M	1	X - North Component of the geomagnetic field
east component	Real	M	1	Y - East Component of the geomagnetic field
vertical component	Real	M	1	Z - Vertical Component of the geomagnetic field
geomagnetic inclination	Real	M	1	I (DIP) - Geomagnetic Inclination
geomagnetic declination	Real	M	1	D (DEC) - Geomagnetic Declination (Magnetic Variation)

Operations				
SetLatitude		Set latitude.		
in	latitude	Real	M	Latitude
SetLongitude		Set longitude.		
in	longitude	Real	M	Longitude
SetAltitude		Set altitude.		
in	Altitude	Real	M	Altitude
SetDate		Set date.		
in	date	Real	M	Date
GetF		Get F.		
out	F	Real	M	F - Total Intensity of the geomagnetic field
GetH		Get H.		
out	H	Real	M	H - Horizontal Intensity of the geomagnetic field
GetX		Get X.		
out	X	Real	M	X - North Component of the geomagnetic field
GetY		Get Y.		
out	Y	Real	M	Y - East Component of the geomagnetic field
GetZ		Get Z.		
out	Z	Real	M	Z - Vertical Component of the geomagnetic field
GetI		Get I.		

out	I	Real	M	I (DIP) - Geomagnetic Inclination
GetD		Get D.		
out	period	Real	M	D (DEC) - Geomagnetic Declination (Magnetic Variation)

7.5 Platform Specific Model (PSM)

This section specifies the PSM for HAL4RT. HAL4RT offers only one PSM, which is based on the ISO/IEC 9899:1999 Programming Language C (also known as C99).

7.5.1 UML-to-C Transformation

7.5.1.1 Type Definition

7.5.1.1.1 char [ISO/IEC-9899]

String is mapped to char.

7.5.1.1.2 Octet [RTC]

Octet is mapped to int32_t.

7.5.1.1.3 double [ISO/IEC-9899]

Real is mapped to double.

7.5.2 C PSM

```
/* HAL4RT_Surface.h */

#ifndef HAL4RT_SURFACE_H
#define HAL4RT_SURFACE_H

int32_t completed();
int32_t notify_event();
int32_t notify_error();
int32_t get_error_detail();
int32_t execute();
int32_t completed();
int32_t subscribe();
int32_t unsubscribe();
int32_t get_event_data();

typedef struct ApplicationBase {
  int32_t completed();
  int32_t notify_event();
  int32_t notify_error();
} APPLICATIONBASE;

typedef struct System {
  int32_t notify_error();
  int32_t get_error_detail();
```

```
} SYSTEM;

typedef struct Command {
  int32_t execute();
  int32_t completed();
} COMMAND;

typedef struct Event {
  int32_t subscribe();
  int32_t unsubscribe();
  int32_t notify_event();
  int32_t get_event_data();
} EVENT;


/* HAL4RT_Device.h */

#ifndef HAL4RT_DEVICE_H
#define HAL4RT_DEVICE_H

typedef struct HALComponent {
  char ComponentName[32];
  uint32_t Vendor_ID;
  uint32_t Product_ID;
  uint32_t Instance_ID;
  uint32_t Profile_ID;
} HALCOMPONENT;

typedef struct Actuator {
  uint32_t ActuatorKindId;
} ACTUATOR

typedef struct Sensor {
  uint32_t SensorKindId;
  int32_t ValueOfMeasurement[32];
  int32_t ScaleFactor[32];
  int32_t Origin[32];
  int32_t GetSensorKind();
  int32_t GetValueOfMeasurement();
  int32_t SetUintOfMeasurement();
  int32_t GetUnitOfMeasurement();
  int32_t SetScaleFactor();
  int32_t SetOrigin();
} SENSOR

#endif HAL4RT_DEVICE_H
```

7.5.3 XML PSM

The below is XML schema to express Message Data Structure.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="CommandMessage" type="CommandMessage"/>
	<xs:complexType name="CommandMessage">
		<xs:sequence>
			<xs:element name="function_name" type="xs:string" minOccurs="1" maxOccurs="1"/>
			<xs:element name="command_id" type="xs:string" minOccurs="1" maxOccurs="1"/>
			<xs:element name="arguments" type="ArgumentList" minOccurs="0" maxOccurs="1"/>
			<xs:element name="results" type="ResultList" minOccurs="0" maxOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="ArgumentList" type="ArgumentList"/>
	<xs:complexType name="ArgumentList">
		<xs:sequence>
			<xs:element name="                    parameters″ type=″Parameter″ minOcc
maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="Parameter" type="Parameter"/>
	<xs:complexType name="Parameter">
		<xs:sequence>
			<xs:element name="data_type_id" type="xs:integer" minOccurs="1" maxOccurs="1"/>
			<xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
			<xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="ResultList" type="ResultList"/>
	<xs:complexType name="ResultList">
		<xs:sequence>
			<xs:element name="parameters" type="Parameter" minOccurs="1"
maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="Device" type="Device"/>
	<xs:complexType name="Device">
		<xs:sequence>
			<xs:element name="device_kind" type="xs:string" minOccurs="1" maxOccurs="1"/>
			<xs:element name="CommandMessage" type="CommandMessage" minOccurs="1"
maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
</xs:schema>
```

This page intentionally left blank.