Document Title	Specification of Health Management for Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	851

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-03

Document Change History			
Date	**Release**	**Changed by**	**Description**
2018-03-29	18-03	AUTOSAR Administration	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

This document is the software specification of the Platform Health Management functional cluster within the Adaptive Platform [1].

The specification implements the requirements specified in [2, RS Health Management].

It also implements the general functionality described in the Foundation documents [3, RS Health Monitoring] and [4, SWS Health Monitoring].

`Health Monitoring` is required by [5, ISO 26262] (under the terms control flow monitoring, external monitoring facility, watchdog, logical monitoring, temporal monitoring, program sequence monitoring) and this specification is supposed to address all relevant requirements from this standard.

2 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to the specification or implementation of Health Monitoring that are not included in the [6, AUTOSAR glossary].

Abbreviation:	Description:
CM	AUTOSAR Adaptive Communication Management
DM	AUTOSAR Adaptive Diagnostic Management
PHM	Platform Health Management
SE	Supervised Entity

Acronym:	Description:
Alive Counter	An independent data resource in context of a Checkpoint to track and handle its amount of Alive Indications.
Alive Indication	An indication of a Supervised Entity to signal its aliveness by calling a checkpoint used for Alive Supervision.
Alive Supervision	Mechanism to check the timing constraints of cyclic Supervised Entities to be within the configured min and max limits.
Checkpoint	A point in the control flow of a Supervised Entity where the activity is reported.
Deadline End Checkpoint	A Checkpoint for which Deadline Supervision is configured and which is a ending point for a particular Transition. It is possible that a Checkpoint is both a Deadline Start Checkpoint and Deadline End Checkpoint - if Deadline Supervision is chained.
Deadline Start Checkpoint	A Checkpoint for which Deadline Supervision is configured and which is a starting point for a particular Transition.
Deadline Supervision	Mechanism to check that the timing constraints for execution of the transition from a to a corresponding are within the configured min and max limits.
Expired Supervision Cycle	A Supervision Cycle where the Alive Supervision has failed its two escalation steps (Alive Counter fails the expected amount of Alive Indications (including tolerances) more often than the allowed amount of failed reference cycles).
Failed Supervision Reference Cycle	A Supervision Reference Cycle that ends with a detected deviation (including tolerances) between the Alive Counter and the expected amount of Alive Indications.
Global Supervision Status	Status that summarizes the Local Supervision Status of all Supervised Entities of a software subsystem.

Graph	A set of Checkpoints connected through Transitions, where at least one of Checkpoints is an Initial Checkpoint and there is a path (through Transitions) between any two Checkpoints of the Graph.
Health Channel	Channel providing information about the health status of a (sub)system. This might be the Global Supervision Status of an application, the result any test routine or the status reported by a (sub)system (e.g. voltage monitoring, OS kernel, ECU status, ...).
Health Channel Supervision	Kind of supervision that checks if the health indicators registered by the supervised software are within the tolerances/limits.
Health Monitoring	Supervision of the software behaviour for correct timing and sequence.
Health Status	A set of states that are relevant to the supervised software (e.g. the Global Supervision Status of an application, a Voltage State, an application state, the result of a RAM monitoring algorithm).
Logical Supervision	Kind of online supervision of software that checks if the software (Supervised Entity or set of Supervised Entities) is executed in the sequence defined by the programmer (by the developed code).
Local Supervision Status	Status that represents the current result of Alive Supervision, Deadline Supervision and Logical Supervision of a single Supervised Entity.
Platform Health Management	`Health Monitoring` for the Adaptive Platform
Supervised Entity	A software entity which is included in the supervision. A Supervised Entity denotes a collection of Checkpoints within an application. There may be zero, one or more Supervised Entities in an application. A Supervised Entity may be instantiated multiple times, in which case each instance is independently supervised.
Supervised Entity Identifier	An Identifier that identifies uniquely a Supervised Entity within an Application.
Supervision Counter	An independent data resource in context of a Supervised Entity which is updated during each supervision cycle and which is used by the `Alive Supervision` algorithm to perform the check against counted Alive Indications.
Supervision Cycle	The time period in which the cyclic `Alive Supervision` is performed.

Supervised Entity	A software entity which is included in the supervision. A Supervised Entity denotes a collection of Checkpoints within a software component. There may be zero, one or more Supervised Entities in a Software Component. A Supervised Entity may be instantiated multiple times, in which case each instance is independently supervised.
Supervision Mode	An overall state of a microcontroller or virtual machine. Modes are mutually exclusive and all Supervised Entities are in the same Supervision Mode. A mode can be e.g. Startup, Shutdown, Low power.
Supervision Reference Cycle	The amount of Supervision Cycles to be used as reference by the `Alive Supervision` to perform the check of counted Alive Indications (individually for each Supervised Entity).

Table 2.1: Acronyms

3 Related documentation

3.1 Input documents & related standards and norms

[1] Explanation of Adaptive Platform Design
AUTOSAR_EXP_PlatformDesign

[2] Requirements on Health Management for Adaptive Platform
AUTOSAR_RS_HealthManagement

[3] Requirements on Health Monitoring
AUTOSAR_RS_HealthMonitoring

[4] Specification of Health Monitoring
AUTOSAR_SWS_HealthMonitoring

[5] ISO 26262 (Part 1-10) – Road vehicles – Functional Safety, First edition
http://www.iso.org

[6] Glossary
AUTOSAR_TR_Glossary

[7] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[8] Methodology for Adaptive Platform
AUTOSAR_TR_AdaptiveMethodology

[9] Guidelines for the use of the C++14 language in critical and safety-related systems
AUTOSAR_RS_CPP14Guidelines

3.2 Related specification

See section 3.1.

4 Constraints and assumptions

4.1 Limitations

[SWS_PHM_00110] Daisy chaining (i.e. forwarding Supervision Status, Checkpoint or Health channel information to an entity external to PHM or another PHM instance) is currently not supported in this document release. ⌈⌋ *(RS_PHM_00108)*

4.2 Applicability to car domains

No restriction

5 Dependencies to other modules

5.1 Platform dependencies

The Platform Health Management functional cluster is dependent on the Execution Management Interface [7]. The Execution Management Interfaces are used by Platform Health Management to request terminating or restarting an Application execution or to request changing machine state, function group state or application state.

The Platform Health Management functional cluster is dependent also on the Watchdog Interface, although this interface is not standardized as it is implementor specific.

6 Requirements Tracing

The following tables reference the requirements specified in [2] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_HM_09240]	Health Monitoring shall support multiple occurrences of the same Supervised Entity.	[SWS_PHM_00457] [SWS_PHM_01116] [SWS_PHM_01120] [SWS_PHM_01121] [SWS_PHM_01123] [SWS_PHM_01133]
[RS_HM_09241]	Health Monitoring shall support multiple instances of Checkpoints in a Supervised Entity occurrence.	[SWS_PHM_01116] [SWS_PHM_01120] [SWS_PHM_01121] [SWS_PHM_01133]
[RS_HM_09254]	Health Monitoring shall provide an interface to Supervised Entities to report the currently reached Checkpoint.	[SWS_PHM_00321] [SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_00458] [SWS_PHM_01010] [SWS_PHM_01123] [SWS_PHM_01124] [SWS_PHM_01125] [SWS_PHM_01127] [SWS_PHM_01131] [SWS_PHM_01132]
[RS_HM_09257]	Health Monitoring shall provide an interface to Supervised Entities for report their health status.	[SWS_PHM_00321] [SWS_PHM_00457] [SWS_PHM_00458] [SWS_PHM_01010] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01122] [SWS_PHM_01124] [SWS_PHM_01126] [SWS_PHM_01128] [SWS_PHM_01131]
[RS_PHM_00001]	The `Platform Health Management` shall provide a standardized header file structure for each service.	[SWS_PHM_01002] [SWS_PHM_01013] [SWS_PHM_01020] [SWS_PHM_01101] [SWS_PHM_01114] [SWS_PHM_01115]
[RS_PHM_00002]	The service header files shall define the namespace for the respective service.	[SWS_PHM_01005] [SWS_PHM_01018] [SWS_PHM_01113]

Requirement	Description	Satisfied by
[RS_PHM_00003]	The `Platform Health Management` shall define how language specific data types are derived from modeled data types.	[SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_01116] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01120] [SWS_PHM_01121] [SWS_PHM_01122] [SWS_PHM_01132] [SWS_PHM_01133]
[RS_PHM_00101]	`Platform Health Management` shall provide a standardized C++ interface for the reporting of `Checkpoints` .	[SWS_PHM_00321] [SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_00458] [SWS_PHM_01010] [SWS_PHM_01123] [SWS_PHM_01124] [SWS_PHM_01125] [SWS_PHM_01127] [SWS_PHM_01131] [SWS_PHM_01132]
[RS_PHM_00102]	`Platform Health Management` shall provide a standardized C++ interface for the reporting of `Health Channel` .	[SWS_PHM_00321] [SWS_PHM_00457] [SWS_PHM_00458] [SWS_PHM_01010] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01122] [SWS_PHM_01124] [SWS_PHM_01126] [SWS_PHM_01128] [SWS_PHM_01131]
[RS_PHM_00108]	`Platform Health Management` shall provide a standardized interface between `Platform Health Management` components used in a daisy chain.	[SWS_PHM_00110] [SWS_PHM_NA]
[RS_PHM_00109]	`Platform Health Management` shall provide the `daisy chaining` interface over `ara::com` .	[SWS_PHM_NA]

7 Functional specification

The Platform Health Management supervises the Applications and could trigger a Recovery Action in case any supervision entity fails. The Recovery Actions are defined by the integrator based on the software architecture requirements for the Platform Health Management and configured in the Application Manifest and/or PHM Manifest. The Execution Management is responsible for the state dependent management of Application start/stop.

`Platform Health Management` invokes functional cluster internal interfaces (RestartProcess API) provided by the Execution Management to restart a specific Process.

`Platform Health Management` invokes functional cluster internal interfaces (OverrideState API) provided by the Execution Management to force the Execution Management to switch to specific Function Group States and/or to a specific Machine State.

`Platform Health Management` invokes functional cluster internal interfaces (ReportApplicationState API) provided by the Execution Management to evaluate the application state. The application state is used to determine the `Supervision Mode`.

The interfaces of `Health Management` to other `Functional Clusters` (OverrideState and RestartProcess) are only informative and not standardized.

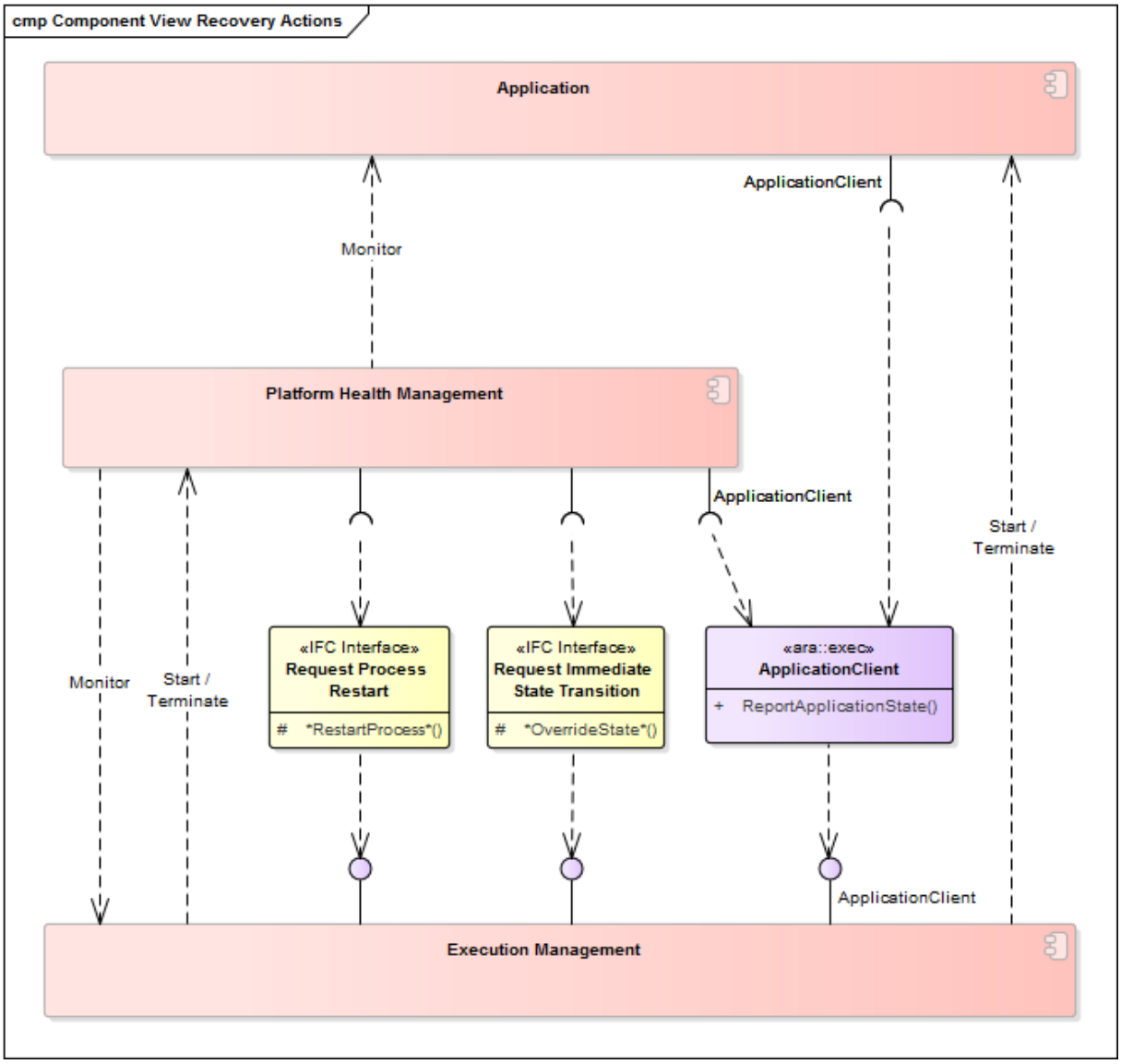

Figure 7.1: `Platform Health Management` and the environment

8 Health Management API specification

8.1 C++ language binding

Note that in this release (2018-03) the C++ language binding uses generated types that are made available to the application (e.g. enumerations with checkpoints), which is generated by AUTOSAR toolchain based on the AUTOSAR manifest. It is possible that this approach will be modified in upcoming 2018-10 AUTOSAR release.

8.1.1 API Header files

This section describes the header files of the `ara::phm` API.

The input for the generated header files of `Platform Health Management` are the AUTOSAR metamodel classes within the `PlatformHealthManagementContribution` description, as defined in the AUTOSAR Adaptive Methodology Specification [8].

8.1.1.1 Generated header file(s)

The generated header files provide the generated types for `Supervised Entitys` and `Health Channels` to use the health management.

8.1.1.1.1 `Supervised Entity`

For each `Supervised Entity`, a separate namespace with a contents is generated.

Namespaces are used to separate the definition of services from each other to prevent name conflicts and they allow to use reasonably short names. It is recommended to define the namespace unique, e.g. by using the company domain name.

[SWS_PHM_01005] Namespace of generated header files for a `Supervised Entity` ⌈ Based on the `symbol` attributes of the ordered `SymbolProps` aggregated by `PhmSupervisedEntityInterface` in role `namespace`, the C++ namespace of a `Supervised Entity` shall be:

```
namespace ara {
namespace phm {

namespace supervised_entities {

namespace <PhmSupervisedEntityInterface.namespace[0].symbol> {
namespace <PhmSupervisedEntityInterface.namespace[1].symbol> {
namespace <...> {
namespace <PhmSupervisedEntityInterface.namespace[n].symbol> {

namespace <PhmSupervisedEntityInterface.shortName> {
```

```
...
} // namespace <PhmSupervisedEntityInterface.shortName>

} // namespace <PhmSupervisedEntityInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <PhmSupervisedEntityInterface.namespace[1].symbol>
} // namespace <PhmSupervisedEntityInterface.namespace[0].symbol>

} // namespace supervised_entities

} // namespace phm
} // namespace ara
```

with all namespace names converted to lower-case letters. ⌋*(RS_PHM_00002)*

So example namespace could be e.g.

`ara::phm::supervised_entities::oem:body::headlights::low_beam`

with low_beam being the name of the `Supervised Entity` and body, headlights and low_beam are namespaces used to organize uniquely identify the `Supervised Entity`.

[SWS_PHM_01020] Folder structure for `Supervised Entity` files ⌈ The generated header files defined by [SWS_PHM_01002] shall be located within the folder:

`<folder>/ara/phm/supervised_entities/<namespace[0]>/.../<namespace[n]>/`

where:
`<folder>` is the start folder for the `ara::phm` header files specific for a project or platform vendor,
`<namespace[0]>` ... `<namespace[n]>` are the namespace names as defined in [SWS_PHM_01005]. ⌋*(RS_PHM_00001)*

[SWS_PHM_01002] Generated header files for Supervised Entities ⌈ The health management shall provide one *`Supervised Entity` header file* for each `PhmSupervisedEntityInterface` defined in the input by using the file name `<name>.h`, where `<name>` is the `PhmSupervisedEntityInterface.shortName` ⌋*(RS_PHM_00001)*

So effectively, for each `Supervised Entity`, there is a separate generated file. There can be several `Supervised Entitys` in the same namespace, which results with several files in the same folder.

8.1.1.1.2 `Health Channel`

The generation of files/namespaces for `Health Channels` is similar to the one of `Supervised Entity`.

[SWS_PHM_01113] Namespace of generated header files for a `Health Channel` ⌈ Based on the `symbol` attributes of the ordered `SymbolProps` aggregated

by PhmHealthChannelInterface in role namespace, the C++ namespace of the Health Channel shall be:

```
namespace ara {
namespace phm {
namespace health_channels {

namespace <PhmHealthChannelInterface.namespace[0].symbol> {
namespace <PhmHealthChannelInterface.namespace[1].symbol> {
namespace <...> {
namespace <PhmHealthChannelInterface.namespace[n].symbol> {

namespace <PhmHealthChannelInterface.shortName> {
...
} // namespace <PhmHealthChannelInterface.shortName>

} // namespace <PhmHealthChannelInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <PhmHealthChannelInterface.namespace[1].symbol>
} // namespace <PhmHealthChannelInterface.namespace[0].symbol>

} // namespace health_channels

} // namespace phm
} // namespace ara
```

with all namespace names converted to lower-case letters. ⌋(*RS_PHM_00002*)

So example namespace could be e.g.

`ara::phm::health_channels::oem::drivetrain::wheels:pressure`

with `pressure` being the name of the Health Channel and oem, drivetrain and wheels are namespaces used to organize uniquely identify the Health Channel.

[SWS_PHM_01114] Folder structure for Supervised Entity files ⌈ The generated header files defined by [SWS_PHM_01002] shall be located within the folder:

`<folder>/ara/phm/health_channels/<namespace[0]>/.../<namespace[n]>/`

where:
`<folder>` is the start folder for the `ara::phm` header files specific for a project or platform vendor,
`<namespace[0]>` ... `<namespace[n]>` are the namespace names as defined in [SWS_PHM_01113]. ⌋(*RS_PHM_00001*)

[SWS_PHM_01115] Generated header files for Health Channels ⌈ The health management shall provide one *Health Channel header file* for each HealthChannel defined in the input by using the file name `<name>.h`, where `<name>` is the HealthChannel.shortName ⌋(*RS_PHM_00001*)

So effectively, for each Health Channel, there is a separate generated file. There can be several Health Channels in the same namespace, which results with several files in the same folder.

8.1.1.2 Non-generated header files

The `Non-generated header files` include the types that provide the `ara::phm` API. Such type definitions are used in the standardized interfaces defined in chapter 8.1.3.

There are following classes:

1. `PHM` - existing in one instance per application, providing supervisions executed locally and providing the communication with remote `PHM` components.
2. `SupervisedEntity` - a class to report `Checkpoints`.
3. `HealthChannel` - a class to report `Health Statuss`.

[SWS_PHM_01101] Folder structure for `Non-generated files` ⌈ The *Non-generated header file*s shall be located within the folder:

`<folder>/ara/phm/`

where:
`<folder>` is the start folder for the ara::phm header files specific for a project or platform vendor. ⌋*(RS_PHM_00001)*

[SWS_PHM_01018] Non-generated header file namespace ⌈ The C++ namespace for the data type definitions included by the *Non-generated header file* shall be:

```
namespace ara {
namespace phm {
...
} // namespace phm
} // namespace ara
```

⌋*(RS_PHM_00002)*

[SWS_PHM_01013] Non-generated header file existence ⌈ The health management shall provide the following `Non-generated header files`:

1. `PHM.hpp` and `PHM.cpp`
2. `SupervisedEntity.hpp`
3. `HealthChannel.hpp`

⌋*(RS_PHM_00001)*

Note that in the current demonstrator `SupervisedEntity.cpp` and `HealthChannel.cpp` are not needed as they are implemented as class templates.

It is not mandatory that all data type definitions are located directly in the *Non-generated header file*. `Health Management` implementation can also distribute the definitions into different header files, but at least all those header files need to be included into the *Non-generated header file*.

8.1.2 API Types

This chapter describes the standardized types provided by the `ara::phm` API, both the ones generated from the description based on the AUTOSAR Metamodel and the specific ones that are non-generated.

8.1.2.1 Generated Types

The types described in this chapter will exist only if there is a related `PhmSupervisedEntityInterface` or `PhmHealthChannelInterface` configured by the user, i.e. they are fully dependent on the input configuration. These types are intended to be used for the unique, configuration-dependent identification of `Supervised Entitys` and `Health Channels`.

An Enumeration is not a plain primitive data type, but a structural description defined with a set of custom identifiers known as *enumerators* representing the possible values. In C++, an enumeration is a first-class object and can take any of these enumerators as a value.

8.1.2.1.1 Generated code for `PhmSupervisedEntityInterface`

The following three items are generated for each `Supervised Entity`, within the namespace:

1. An enumeration with the `Checkpoints`
2. A type identifying this `Supervised Entity`
3. A type identifying each `Supervised Entity` prototype

[SWS_PHM_00424] Enumeration for `Supervised Entity` ⌈ For each `PhmSupervisedEntityInterface`, there shall exist the corresponding type declaration as:

```
enum class Checkpoints : EnumUnderlyingType {
  <enumerator-list>
};
```

where:

`<enumerator-list>` are the enumerators as defined by [SWS_PHM_00425].

`EnumUnderlyingType` defines the standardized underlying type for the Id.

⌋*(RS_PHM_00003, RS_PHM_00101, RS_HM_09254)*

[SWS_PHM_00425] Definition of enumerators of `Supervised Entitys` ⌈ For each `PhmCheckpoint` contained in the `PhmSupervisedEntityInterface`, there shall exist the corresponding enumeration nested in the declaration defined by [SWS_PHM_00424] as:

```
  <enumeratorLiteral> = <initializer><suffix>,
```

where:

<enumeratorLiteral> is PhmCheckpoint.shortName

<initializer> is the PhmCheckpoint.id

<suffix> shall be "U".

⌋(*RS_PHM_00003*, *RS_PHM_00101*, *RS_HM_09254*)

For example, this can generate:

```
enum class Checkpoints : EnumUnderlyingType
{
        Initializing = 0U,
        StartupTest = 1U,
        InitializingFinished = 2U
};
```

[SWS_PHM_01116] Definition of an identifier for a Supervised Entitys ⌈ For each Supervised Entity there shall exist a corresponding declaration as:

```
template <PrototypeType PrototypeId>
using SE = Identifier<<supervisedEntityId><suffix>,
                      PrototypeId,
                      Checkpoints>;
```

where:

<supervisedEntityId> is PhmSupervisedEntityInterface.id

<suffix> shall be "U"

PrototypeType defines the standardized underlying type for a prototype

Identifer is a class template provided by Platform Health Management.

⌋(*RS_PHM_00003*, *RS_HM_09240*, *RS_HM_09241*)

For example, this can generate (with 100U being the Supervised Entity ID):

```
template <PrototypeType PrototypeId>
using SE = Identifier<100U, PrototypeId, Checkpoints>;
```

[SWS_PHM_01133] Definition of an identifier for a Supervised Entity Prototypes ⌈ For each Supervised Entity Prototype there shall exist a list of corresponding declarations as:

```
using Prototype<prototypeId> = SE<<prototypeId><suffix>>;
```

where:

<prototypeId> is list of numbers in the range from 0 to PhmSupervisedEntityInterface.numberOfPrototypes - 1.

<suffix> shall be "U".

⌋(*RS_PHM_00003*, *RS_HM_09240*, *RS_HM_09241*)

For example, this can generate, for a Supervised Entity that has 2 prototypes:

```
using Prototype0 = SE<0U>;
using Prototype1 = SE<1U>;
```

8.1.2.1.2 Enumeration for PhmHealthChannelInterface

The generation for Health Channels is similar to the one of Supervised Entitys.

Items are generated for each Health Channel, within the namespace:

1. An enumeration with the Health Statuses
2. A type identifying this Health Channel
3. A type identifying each possible Health Channel prototype.

[SWS_PHM_01118] Enumeration for Health Channel ⌈ For each PhmHealthChannelInterface, there shall exist the corresponding type declaration as:

```
enum class HealthStatuses : EnumUnderlyingType {
        <enumerator-list>
};
```

where:

<enumerator-list> are the enumerators as defined by [SWS_PHM_01119]

EnumUnderlyingType defines the standardized underlying type for the Id.

⌋(*RS_PHM_00003*, *RS_PHM_00102*, *RS_HM_09257*)

[SWS_PHM_01119] Definition of enumerators of Health Channels ⌈ For each PhmHealthChannelStatus contained in the PhmHealthChannelInterface, there shall exist the corresponding enumeration nested in the declaration defined by [SWS_PHM_00424] as:

```
<enumeratorLiteral> = <initializer><suffix>,
```

where:

<enumeratorLiteral> is PhmHealthChannelStatus.shortName

<initializer> is the PhmHealthChannelStatus.id

<suffix> shall be "U".

⌋(*RS_PHM_00003*, *RS_PHM_00102*, *RS_HM_09257*)

For example, this can generate:

```
enum class HealthStatuses : EnumUnderlyingType
{
        Low = 0U,
        High = 1U,
        Ok = 2U,
        VeryLow = 3,
        VeryHigh = 4
};
```

[SWS_PHM_01120] Definition of an identifier for a `Health Channels` ⌈ For each `Health Channel` there shall exist a corresponding declaration as:

```
template <PrototypeType PrototypeId>
using HC = Identifier<<HealthChannelId><suffix>, PrototypeId, HealthStatuses>;
```

where:

<HealthChannelId> is `PhmHealthChannelInterface.id`

<suffix> shall be "U"

`PrototypeType` defines the standardized underlying type for a prototype

`Identifer` is a class template provided by `Platform Health Management`.

⌋*(RS_PHM_00003, RS_HM_09240, RS_HM_09241)*

For example, this can generate:

```
template <PrototypeType PrototypeId>
using HC = Identifier<102U, PrototypeId, HealthStatuses>;
```

[SWS_PHM_01121] Definition of an identifier for a `Health Channel Prototypes` ⌈ For each `Health Channel Prototype` there shall exist a list of corresponding declarations as:

```
using Prototype<prototypeId> = SE<<prototypeId><suffix>>;
```

where:

<prototypeId> is list of numbers in the range from 0 to `PhmHealthChannelInterface.numberOfPrototypes - 1`.

<suffix> shall be "U".

⌋*(RS_PHM_00003, RS_HM_09240, RS_HM_09241)*

For example, this can generate, for a `Health Channel` that has 4 prototypes:

```
using Prototype0 = HC<0>;
using Prototype1 = HC<1>;
using Prototype2 = HC<2>;
using Prototype3 = HC<3>;
```

8.1.2.2 Non-generated types

This section defines the types that are non-generated.

8.1.2.2.1 Data types

[SWS_PHM_00321] Underlying data types ⌈ Health Management shall provide the following data types - InterfaceType, PrototypeType, InstanceType, EnumUnderlyingType:

```
using InterfaceType       = std::uint16_t;
using PrototypeType       = std::uint8_t;
using InstanceType        = std::int32_t;
using EnumUnderlyingType  = std::uint8_t;
```

⌋(*RS_PHM_00101*, *RS_PHM_00102*, *RS_HM_09254*, *RS_HM_09257*)

This means that a globally unique serialized representation of a Checkpoint or of a Health Status takes 4 bytes.

8.1.2.2.2 Identifier

[SWS_PHM_01131] Identifier Class Template ⌈ Health Management shall provide a Identifier class, which represents uniquely an prototype of a Supervised Entity Prototype/Health Channel Prototype and it identifies its enumeration type.

```
template <InterfaceType Id, PrototypeType PrototypeId, typename Enum>
struct Identifier
{

        /// definition of the supervised entity Id / health channel Id
        constexpr static InterfaceType id = Id;

        /// definition of the prototype Id,
        constexpr static PrototypeType prototypeId = PrototypeId;

        /// definition of all checkpoints/health statuses of this SE
        using EnumType = Enum;
};
```

⌋(*RS_PHM_00101*, *RS_PHM_00102*, *RS_HM_09254*, *RS_HM_09257*)

Identifier is used by the generated classes SupervisedEntity and HealthChannel.

8.1.2.2.3 SupervisedEntity

[SWS_PHM_01132] SupervisedEntity Class Template ⌈ Health Management shall provide a SupervisedEntity class template which shall inherit from PHM and which shall provide a method to report Checkpoints.

```
template <InterfaceType InterfaceId, PrototypeType PrototypeId, typename Enum>
class SupervisedEntity<Identifier<InterfaceId, PrototypeId, Enum>>
    : private PHM
{
        public:

        explicit SupervisedEntity(PHM& phm) : PHM{phm} {}

        void ReportCheckpoint(Enum t);
};
```

⌋(*RS_PHM_00003*, *RS_PHM_00101*, *RS_HM_09254*)

8.1.2.2.4 HealthChannel

[SWS_PHM_01122] HealthChannel Class Template ⌈ Health Management shall provide a HealthChannel class template which shall inherit from PHM and which shall provide a method to report HealthStatuss.

```
template <InterfaceType InterfaceId, PrototypeType PrototypeId, typename Enum>
class HealthChannel<Identifier<InterfaceId, PrototypeId, Enum>>
     : private PHM
{
        public:

        explicit HealthChannel(PHM& phm) : PHM{phm} {}

        void ReportHealthStatus(Enum t);
};
```

⌋(*RS_PHM_00003*, *RS_PHM_00102*, *RS_HM_09257*)

8.1.2.2.5 PHM

[SWS_PHM_01010] PHM Class ⌈ The Health Management shall provide a C++ class named PHM, which shall be responsible for the establishment of the communication with the PHM Daemons and the establishment of the supervision executed locally and which shall contain a copy-constructor and two protected methods (used by SupervisedEntity and HealthChannel).

```
class PHM
{
    public:
    PHM() {
        // implementation-specific
    }

    PHM(PHM&  phm) {

        // implementation-specific, shallow-copy

    }

    // remaining special member functions and destructor according to C++14 coding guidelines.
    // It is implementation specific if they are delete, default or have custom implementation.
    // Probably the move constructor does not make sense.

    protected:
    void ReportCheckpoint(InterfaceType supervisedEntityId, PrototypeType prototypeId, InstanceType instanceId, EnumUnderlyingType checkpointId);

    void ReportHealthStatus(InterfaceType healthChannelId, PrototypeType prototypeId, InstanceType instanceId, EnumUnderlyingType healthStatusId);
};

```

⌋(*RS_PHM_00101*, *RS_PHM_00102*, *RS_HM_09254*, *RS_HM_09257*)

8.1.2.3 `Daisy Chaining` Related Types (Non-generated)

Daisy chaining is not supported in this AUTOSAR release.

8.1.2.4 Error and Exception Types

The ara::phm API make use of C++ exceptions to notify the user of the API about any errors occurred. ara::phm API does hereby strictly follow [9, AUTOSAR CPP14 guidelines] regarding exception usage. I.e. there is a clean separation of exception types into `Unchecked Exceptions` and `Checked Exceptions`, which ara::phm API builds upon.

The former ones (i.e., `Unchecked Exceptions`) can basically occur in *any* ara::phm API call, are not formally modeled in the Manifest, and are fully implementation specific.

The latter ones (i.e., `Unchecked Exceptions`) are not used by `Health Management API`.

8.1.2.5 E2E Related Data Types

The usage of `E2E communication protection` for `Health Management` is not standardized.

8.1.3 API Reference

8.1.3.1 `SupervisedEntity` API

[SWS_PHM_01123] Creation of a `SupervisedEntity` ⌈ The Health Management shall provide constructor for class `SupervisedEntity` accepting the reference to `PHM`.

```
SupervisedEntity(PHM& phm): PHM{phm}
```

⌋*(RS_PHM_00101, RS_HM_09254, RS_HM_09240)*

[SWS_PHM_01127] ReportCheckpoint ⌈ The Health Management shall provide a method ReportCheckpoint, provided by `SupervisedEntity`.

```
void ReportCheckpoint(Enum t);
```

Where Enum is defined by the class template `SupervisedEntity` ⌋*(RS_PHM_00101, RS_HM_09254)*

8.1.3.2 `HealthChannel` API

[SWS_PHM_00457] Creation of a `HealthChannel` ⌈ The Health Management shall provide constructor for class `HealthChannel` accepting the reference to `PHM`.

```
HealthChannel(PHM& phm): PHM{phm}
```

⌋*(RS_PHM_00102, RS_HM_09257, RS_HM_09240)*

[SWS_PHM_01128] ReportHealthStatus ⌈ The Health Management shall provide a method ReportHealthStatus, provided by `HealthChannel`.

```
void ReportHealthStatus(Enum t);
```

Where Enum is defined by the class template `HealthChannel` ⌋*(RS_PHM_00102, RS_HM_09257)*

8.1.3.3 `PHM` API

[SWS_PHM_00458] Creation of PHM service interface ⌈ The Health Management shall provide a default constructor for class `PHM`.

```
PHM()
```

⌋*(RS_PHM_00101, RS_PHM_00102, RS_HM_09254, RS_HM_09257)*

[SWS_PHM_01124] Copy constructor for the use by `SupervisedEntity` and by `HealthChannel` ⌈ The Health Management shall provide a copy default constructor for class `PHM`.

```
PHM(PHM& phm)
```

⌋(*RS_PHM_00101*, *RS_PHM_00102*, *RS_HM_09254*, *RS_HM_09257*)

[SWS_PHM_01125] ReportCheckpoint ⌈ The Health Management shall provide a protected method ReportCheckpoint, provided by PHM, used by SupervisedEntity.

```
void ReportCheckpoint(InterfaceType supervisedEntityId, PrototypeType prototypeI
                      EnumUnderlyingType checkpointId);
```

⌋(*RS_PHM_00101*, *RS_HM_09254*)

[SWS_PHM_01126] ReportHealthStatus ⌈ The Health Management shall provide a protected method ReportHealthStatus, provided by PHM, used by HealthChannel.

```
void ReportHealthStatus(InterfaceType healthChannelId, PrototypeType prototypeId
                        EnumUnderlyingType healthStatusId);
```

⌋(*RS_PHM_00102*, *RS_HM_09257*)

8.1.3.4 Forward supervision state (daisy-chain)

This feature is not supported by this AUTOSAR release.

A Not applicable requirements

[SWS_PHM_NA] ⌈ These requirements are not applicable as they are not within the scope of this release. ⌋*(RS_PHM_00108, RS_PHM_00109)*

B Example implementation of `ara::phm`

This chapter provides an example implementation of `ara::phm` API. This chapter is informative. It can be used as a user manual, as an implementation hint or as an initial demonstrator.

B.1 Application

The following listing shows an example adaptive application. It has:

1. Engine `Supervised Entity` that is a single instance
2. Wheel `Supervised Entity` that is in four instances
3. WheelPressure `Health Channel` that is in four instances

There are no explicit integer identifiers in the application code (for supervised entity, instance, enum), this is cleanly encapsulated by the API.

```
#include "ara/phm/HealthChannel.hpp"
#include "ara/phm/PHM.hpp"
#include "ara/phm/SupervisedEntity.hpp"

// generated files with the Supervised Entities and Health Channels.
#include "ara/phm/health_channels/TyrePressure.hpp"
#include "ara/phm/supervised_entities/Engine.hpp"
#include "ara/phm/supervised_entities/Wheel.hpp"

// this file is just for the purpose of the demonstration, they are not needed in production code
#include <typeinfo>

// namespace with non-generated phm code
using namespace ara::phm;

// namespaces with the generated code
using namespace ara::phm::supervised_entities;
using namespace ara::phm::health_channels;

int main()
{

    std::cout << std::endl
              << "PHM Demo" << std::endl
              << "for each supervised entity prototype, e.g. engine::Prototype0, there is "
              << "a type with 3 attributes, available for the application: " << std::endl;
    std::cout << "Id of engine Supervised Entity: " << engine::Prototype0::interfaceId << std::endl;
    std::cout << "Id of engine0 Supervised Entity Prototype: " << static_cast<int>(engine::Prototype0::prototypeId)
              << std::endl;
```

```
    std::cout << "Enum type for engine: " << typeid(engine::Prototype0::EnumType).name() << std::endl;

    std::cout << std::endl << "Creating phm" << std::endl;
    PHM phm{};

    // example 1: single prototype of SE (engine0) with 3 checkpoints
    std::cout << std::endl << "example 1: single prototype of SE (engine) with 3 checkpoints" << std::endl;
    SupervisedEntity<engine::Prototype0> engine0{phm};

    std::cout << "- prototype 0" << std::endl;
    engine0.ReportCheckpoint(engine::Checkpoints::Initializing);
    engine0.ReportCheckpoint(engine::Checkpoints::StartupTest);
    engine0.ReportCheckpoint(engine::Checkpoints::InitializingFinished);

    // example 2: four prototypes of the same SE, each with 2 checkpoints
    std::cout << std::endl << "example 2: four prototypes of the same SE (wheel), each with 4 checkpoints" << std::endl;
    SupervisedEntity<wheel::Prototype0> wheel0{phm};
    SupervisedEntity<wheel::Prototype1> wheel1{phm};
    SupervisedEntity<wheel::Prototype2> wheel2{phm};
    SupervisedEntity<wheel::Prototype3> wheel3{phm};

    std::cout << "- prototype 0" << std::endl;
    wheel0.ReportCheckpoint(wheel::Checkpoints::Started);
    wheel0.ReportCheckpoint(wheel::Checkpoints::Finished);

    std::cout << "- prototype 1" << std::endl;
    wheel1.ReportCheckpoint(wheel::Checkpoints::Started);
    wheel1.ReportCheckpoint(wheel::Checkpoints::Finished);

    std::cout << "- prototype 2" << std::endl;
    wheel2.ReportCheckpoint(wheel::Checkpoints::Started);
    wheel2.ReportCheckpoint(wheel::Checkpoints::Finished);

    std::cout << "- prototype 3" << std::endl;
    wheel3.ReportCheckpoint(wheel::Checkpoints::Started);
    wheel3.ReportCheckpoint(wheel::Checkpoints::Finished);

    // example 3: four prototypes of the type wheel pressure health status
    std::cout << std::endl << "example 3: four prototypes of the type (wheel pressure health status)" << std::endl;
    HealthChannel<tyre_pressure::Prototype0> tyre0{phm};
    HealthChannel<tyre_pressure::Prototype1> tyre1{phm};
    HealthChannel<tyre_pressure::Prototype2> tyre2{phm};
    HealthChannel<tyre_pressure::Prototype3> tyre3{phm};


    std::cout << "- prototype 0 - with 2 health statuses reported" << std::endl;
    tyre0.ReportHealthStatus(tyre_pressure::HealthStatuses::Low);
    tyre0.ReportHealthStatus(tyre_pressure::HealthStatuses::Ok);

    std::cout << "- prototype 1" << std::endl;
    tyre1.ReportHealthStatus(tyre_pressure::HealthStatuses::Ok);

    std::cout << "- prototype 2" << std::endl;
    tyre2.ReportHealthStatus(tyre_pressure::HealthStatuses::High);

    std::cout << "- prototype 3" << std::endl;
    tyre3.ReportHealthStatus(tyre_pressure::HealthStatuses::VeryLow);

    return 0;
}
```

This example application generates the following text output:

```

PHM Demo
```

```
for each supervised entity prototype, e.g. engine::Prototype0, there is a type with 3 attributes, available for the application:
Id of engine Supervised Entity: 100
Id of engine0 Supervised Entity Prototype: 0
Enum type for engine: N3ara3phm19supervised_entities6engine11CheckpointsE

Creating phm

example 1: single prototype of SE (engine) with 3 checkpoints
- prototype 0
  Received checkpoint. Supervised entity:100 Prototype:0 Instance:6921 Checkpoint:0
  Received checkpoint. Supervised entity:100 Prototype:0 Instance:6921 Checkpoint:1
  Received checkpoint. Supervised entity:100 Prototype:0 Instance:6921 Checkpoint:2

example 2: four prototypes of the same SE (wheel), each with 4 checkpoints
- prototype 0
  Received checkpoint. Supervised entity:101 Prototype:0 Instance:6921 Checkpoint:0
  Received checkpoint. Supervised entity:101 Prototype:0 Instance:6921 Checkpoint:1
- prototype 1
  Received checkpoint. Supervised entity:101 Prototype:1 Instance:6921 Checkpoint:0
  Received checkpoint. Supervised entity:101 Prototype:1 Instance:6921 Checkpoint:1
- prototype 2
  Received checkpoint. Supervised entity:101 Prototype:2 Instance:6921 Checkpoint:0
  Received checkpoint. Supervised entity:101 Prototype:2 Instance:6921 Checkpoint:1
- prototype 3
  Received checkpoint. Supervised entity:101 Prototype:3 Instance:6921 Checkpoint:0
  Received checkpoint. Supervised entity:101 Prototype:3 Instance:6921 Checkpoint:1

example 3: four prototypes of the type (wheel pressure health status)
- prototype 0 - with 2 health statuses reported
  Received health status. Health channel:102 Prototype:0 Instance:6921 Health status:0
  Received health status. Health channel:102 Prototype:0 Instance:6921 Health status:2
- prototype 1
  Received health status. Health channel:102 Prototype:1 Instance:6921 Health status:2
- prototype 2
  Received health status. Health channel:102 Prototype:2 Instance:6921 Health status:1
- prototype 3
  Received health status. Health channel:102 Prototype:3 Instance:6921 Health status:3
```

B.2 PHM Generated code

The following information is generated out of the configuration files:

1. namespace of `Supervised Entity` or `Health Channel`
2. a separate type for each `Supervised Entity` or `Health Channel`
3. a separate enumeration for the list of possible `Checkpoints` or `Health Statuses`
4. a separate type for each instance of `Supervised Entity` or `Health Channel`.

The following two files show the generated types for the example application for `Supervised Entitys`:

Engine:

```
#ifndef _ARA_PHM_SUPERVISED_ENTITIES_ENGINE_HPP
#define _ARA_PHM_SUPERVISED_ENTITIES_ENGINE_HPP

#include "ara/phm/PHM.hpp"
```

```
namespace ara
{
namespace phm
{

namespace supervised_entities
{

namespace engine
{

// definition of all health statuses of this SE
enum class Checkpoints : EnumUnderlyingType
{
    Initializing = 0U,
    StartupTest = 1U,
    InitializingFinished = 2U
};

template <PrototypeType PrototypeId>
using SE = Identifier<100U, PrototypeId, Checkpoints>;

// definition of the supervised entity prototype - with prototype ID
using Prototype0 = SE<0U>;
}  // namespace engine
}  // namespace supervised_entities
}  // namespace phm
}  // namespace ara

#endif // _ARA_PHM_SUPERVISED_ENTITIES_ENGINE_HPP
```

Wheel:

```
#ifndef _ARA_PHM_SUPERVISED_ENTITIES_WHEEL_HPP
#define _ARA_PHM_SUPERVISED_ENTITIES_WHEEL_HPP

#include "ara/phm/PHM.hpp"

namespace ara
{
namespace phm
{

namespace supervised_entities
{

namespace wheel
{

// definition of all checkpoints of this SE
enum class Checkpoints : EnumUnderlyingType
{
    Started = 0U,
    Finished = 1U
};

template <PrototypeType PrototypeId>
using SE = Identifier<101U, PrototypeId, Checkpoints>;

using Prototype0 = SE<0>;
using Prototype1 = SE<1>;
using Prototype2 = SE<2>;
using Prototype3 = SE<3>;
}  // namespace wheel
}  // namespace supervised_entities
}  // namespace phm
}  // namespace ara

#endif // _ARA_PHM_SUPERVISED_ENTITIES_WHEEL_HPP
```

A similar code is generated for Health Channels:

```
#ifndef _ARA_PHM_HEALTH_CHANNELS_TYREPRESSURE_HPP
#define _ARA_PHM_HEALTH_CHANNELS_TYREPRESSURE_HPP

#include "ara/phm/PHM.hpp"

namespace ara
{
namespace phm
{

namespace health_channels
{

namespace tyre_pressure
{

// definition of all possible health statuses
enum class HealthStatuses : EnumUnderlyingType
{
    Low = 0U,
    High = 1U,
    Ok = 2U,
    VeryLow = 3,
    VeryHigh = 4
};

// definition of the supervised entity - with the SE ID
template <PrototypeType PrototypeId>
using HC = Identifier<102U, PrototypeId, HealthStatuses>;

// definition of the supervised entity prototype - with prototype ID
using Prototype0 = HC<0>;
using Prototype1 = HC<1>;
using Prototype2 = HC<2>;
using Prototype3 = HC<3>;
}  // namespace tyre_pressure
}  // namespace health_channels
}  // namespace phm
}  // namespace ara

#endif // _ARA_PHM_HEALTH_CHANNELS_TYREPRESSURE_HPP
```

B.3 PHM Non-generated code

Class PHM provides supervision checks executed locally and it provides a communication with remote PHM daemons. It sees Checkpoints/Health Statuses as a tuples of 3 integers (id, instance id, serialized enum value), taking together 4 bytes.

PHM operates fully based on the xml/json configuration.

PHM.hpp (simplified):

```
#ifndef _ARA_PHM_PHM_HPP
#define _ARA_PHM_PHM_HPP

#include <cstdint>
#include <iostream>
#include <type_traits>
#include <unistd.h>

// non-generated code
namespace ara
{
```

```
namespace phm
{

using InterfaceType = uint16_t;
using PrototypeType = uint8_t;
using InstanceType = int32_t;
using EnumUnderlyingType = uint8_t;

class PHM
{
  public:
    PHM() : instanceId{getpid()} {}

    PHM(PHM& phm) : instanceId{phm.instanceId} {}

    ~PHM() = default;

  protected:
    void ReportCheckpoint(InterfaceType supervisedEntityId,
                          PrototypeType prototypeId,
                          InstanceType instanceId,
                          EnumUnderlyingType checkpointId);

    void ReportHealthStatus(InterfaceType healthChannelId,
                            PrototypeType prototypeId,
                            InstanceType instanceId,
                            EnumUnderlyingType healthStatusId);

    InstanceType GetInstanceId() { return instanceId; };

  private:
    InstanceType instanceId;
};

// An identifier for each Supervised Entity prototype or Health Channel prototype
template <InterfaceType InterfaceId, PrototypeType PrototypeId, typename Enum>
struct Identifier
{

    /// definition of the supervised entity Id / health channel Id
    constexpr static InterfaceType interfaceId = InterfaceId;

    /// definition of the prototype Id,
    constexpr static PrototypeType prototypeId = PrototypeId;

    /// definition of all checkpoints/health statuses of this SE
    using EnumType = Enum;
};

template <typename T>
struct DependentFalse : std::false_type
{
};
}  // namespace phm
}  // namespace ara

#endif // _ARA_PHM_PHM_HPP
```

PHM.cpp (simplified - the methods only print out the identifiers):

```
#include "ara/phm/PHM.hpp"

namespace ara
{
namespace phm
{

void PHM::ReportCheckpoint(InterfaceType supervisedEntityId,
                           PrototypeType prototypeId,
                           InstanceType instanceId,
```

```
                            EnumUnderlyingType checkpointId)
{

    std::cout << "  Received checkpoint. "
              << "Supervised entity:" << +supervisedEntityId << " Prototype:" << static_cast<
    int>(prototypeId)
              << " Instance:" << static_cast<int>(instanceId)
              << " Checkpoint:" << static_cast<int>(checkpointId) << std::endl;
}

void PHM::ReportHealthStatus(InterfaceType healthChannelId,
                             PrototypeType prototypeId,
                             InstanceType instanceId,
                             EnumUnderlyingType healthStatusId)
{

    std::cout << "  Received health status. "
              << "Health channel:" << +healthChannelId << " Prototype:" << static_cast<int>(
    prototypeId)
              << " Instance:" << static_cast<int>(instanceId)
              << " Health status:" << static_cast<int>(healthStatusId) << std::endl;
}
}  // namespace phm
}  // namespace ara
```

The class PHM is used by classes SupervisedEntity and HealthChannel, which are template classes over the generated types. Moreover, they also inherit from PHM to have a access it its protected methods (it is a has-a relationship realized with private inheritance).

SupervisedEntity.hpp:

```
#ifndef _ARA_PHM_SUPERVISEDENTITY_HPP
#define _ARA_PHM_SUPERVISEDENTITY_HPP

#include <cstdint>
#include <iostream>
#include <type_traits>

#include "ara/phm/PHM.hpp"

using namespace ara::phm;

namespace ara
{
namespace phm
{

template <typename T>
class SupervisedEntity
{
    static_assert(DependentFalse<T>::value, "SupervisedEntity must be created using Identifier
     template");
};

template <InterfaceType Id, PrototypeType PrototypeId, typename Enum>
class SupervisedEntity<Identifier<Id, PrototypeId, Enum>> : private PHM
{
  public:
    explicit SupervisedEntity(PHM& phm) : PHM{phm} {}

    void ReportCheckpoint(Enum t);
};

template <InterfaceType Id, PrototypeType PrototypeId, typename Enum>
void SupervisedEntity<Identifier<Id, PrototypeId, Enum>>::ReportCheckpoint(Enum t)
{
    auto checkpointId = static_cast<std::underlying_type_t<Enum>>(t);
```

```
36
37      PHM::ReportCheckpoint(Id, PrototypeId, GetInstanceId(), checkpointId);
38  }
39  }  // namespace phm
40  }  // namespace ara
41
42  #endif
```

HealthChannel.hpp (right now looking similar, but we assume that new use cases will introduce differences to SupervisedEntity):

```
1   #ifndef _ARA_PHM_HEALTHCHANNEL_HPP
2   #define _ARA_PHM_HEALTHCHANNEL_HPP
3
4   #include <cstdint>
5   #include <iostream>
6   #include <type_traits>
7
8   #include <ara/phm/PHM.hpp>
9
10  namespace ara
11  {
12  namespace phm
13  {
14
15  template <typename T>
16  class HealthChannel
17  {
18      static_assert(DependentFalse<T>::value, "HealthChannel must be created using Identifier
        template");
19  };
20
21  template <InterfaceType Id, PrototypeType PrototypeId, typename Enum>
22  class HealthChannel<Identifier<Id, PrototypeId, Enum>> : private PHM
23  {
24    public:
25      explicit HealthChannel(PHM& phm) : PHM{phm} {}
26
27      void ReportHealthStatus(Enum t);
28  };
29
30  template <InterfaceType Id, PrototypeType PrototypeId, typename Enum>
31  void HealthChannel<Identifier<Id, PrototypeId, Enum>>::ReportHealthStatus(Enum t)
32  {
33      auto healthStatusId = static_cast<std::underlying_type_t<Enum>>(t);
34
35      PHM::ReportHealthStatus(Id, PrototypeId, GetInstanceId(), healthStatusId);
36  }
37  }  // namespace phm
38  }  // namespace ara
39  #endif
```

C Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	***HealthChannel* (abstract)**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::PlatformHealthManagement			
Note	This element defines the source of a health channel. **Tags:** atp.ManifestKind=ApplicationManifest; atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	HealthChannelExternalStatus, HealthChannelSupervision			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table C.1: HealthChannel

Class	***ImplementationProps* (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.			
Base	*ARObject*, *Referrable*			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps, SymbolicNameProps			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
symbol	CIdentifier	1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table C.2: ImplementationProps

Class	**PhmCheckpoint**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the ability to implement a checkpoint for interaction with the Platform Health Management Supervised Entity. **Tags:** atp.Status=draft			
Base	*ARObject*, *AtpFeature*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
checkpointId	PositiveInteger	1	attr	Defines the numeric value which is used to indicate the reporting of this Checkpoint to the Phm. **Tags:** atp.Status=draft

Table C.3: PhmCheckpoint

Class	**PhmHealthChannelInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the ability to implement a PortInterface for interaction with the Platform Health Management Health Channel. **Tags:** atp.Status=draft; atp.recommendedPackage=PlatformHealthManagement Interfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
healthChannelId	PositiveInteger	1	attr	Defines the numeric value which is used to indicate the reporting of this Health Channel to the Phm. **Tags:** atp.Status=draft
status	PhmHealthChannelStatus	*	aggr	Defines the possible set of status information available to the health channel. **Tags:** atp.Status=draft

Table C.4: PhmHealthChannelInterface

Class	**PhmHealthChannelStatus**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	The PhmHealthChannelStatus specifies one possible status of the health channel. **Tags:** atp.Status=draft			
Base	*ARObject*, *AtpFeature*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
statusId	PositiveInteger	1	attr	Defines the numeric value which is used to indicate the indication of this status the Phm. **Tags:** atp.Status=draft

Table C.5: PhmHealthChannelStatus

Class	**PhmSupervisedEntityInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the ability to implement a PortInterface for interaction with the Platform Health Management Supervised Entity. **Tags:** atp.Status=draft; atp.recommendedPackage=PlatformHealthManagement Interfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***

Attribute	Type	Mul.	Kind	Note
checkpoint	PhmCheckpoint	*	aggr	Defines the set of checkpoints which can be reported on this supervised entity. **Tags:** atp.Status=draft
supervised EntityId	PositiveInteger	1	attr	Defines the numeric value which is used to interact with this Supervised Entity when calling the Phm. **Tags:** atp.Status=draft

Table C.6: PhmSupervisedEntityInterface

Class	PlatformHealthManagementContribution			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::PlatformHealthManagement			
Note	This element defines a contribution to the Platform Health Management. **Tags:** atp.ManifestKind=ApplicationManifest; atp.Status=draft; atp.recommendedPackage=PlatformHealthManagementContributions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
action	Action	*	aggr	Collection of Actions and ActionLists in the context of a PlatformHealthManagementContribution. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft xml.sequenceOffset=50
arbitration	Arbitration	*	aggr	Collection of Arbitrations in the context of a PlatformHealthManagementContribution. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft xml.sequenceOffset=40
checkpoint	SupervisionCheckpoint	*	aggr	Collection of checkpoints in the context of a PlatformHealthManagementContribution. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft xml.sequenceOffset=10
globalSupervision	GlobalSupervision	*	aggr	Collection of GlobalSupervisions in the context of a PlatformHealthManagementContribution. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft xml.sequenceOffset=30
healthChannel	HealthChannel	*	aggr	Collection of HealthChannels in the context of a PlatformHealthManagementContribution. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft xml.sequenceOffset=30

Attribute	Type	Mul.	Kind	Note
process	Process	0..1	ref	Reference to the Process this PhmContribution shall be applied to. **Tags:** atp.Status=draft xml.sequenceOffset=90

Table C.7: PlatformHealthManagementContribution

Class	*PlatformHealthManagementInterface* (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the abstract ability to define a PortInterface for the interaction with Platform Health Management. **Tags:** atp.Status=draft			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	PhmHealthChannelInterface, PhmSupervisedEntityInterface			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table C.8: PlatformHealthManagementInterface

Class	*Referrable* (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClient ServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, Diagnostic DebounceAlgorithmProps, *DiagnosticEnvModeElement*, EthernetPriority Regeneration, EventHandler, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *Multilanguage Referrable*, PncMappingIdent, *SingleLanguageReferrable*, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table C.9: Referrable

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to attach with the symbol attribute a symbolic name that is conform to C language requirements to another meta-class, e.g. AtomicSwComponentType, that is a potential subject to a name clash on the level of RTE source code.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table C.10: SymbolProps