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

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Health Channels are set to obsolete • Removed retry after failed notification to State Management • Removed GetLocalSupervisionStatus() and GetGlobalSupervisionStatus() APIs from SupervisedEntity class • Added Determination of Supervision Status from Foundation SWS_HealthMonitoring • Added Mode Dependent Configuration Concept • Alignment of Enumeration Literal Indices of SupervisionStatus with Classic Platform WdgM types • Introduction of PhmErrorDomain • Introduction of WatchdogInterface
2020-11-30	R20-11	AUTOSAR Release Management	• Changed role of PHM to a monitor who notifies State Management, thus rework of logic and interfaces. • Integration of Identity and Access Management for PHM • Moving specification of Health Channel Supervision from Foundation to Adaptive Platform • Reintroduced Enum for Checkpoints and Health Status

2019-11-28	R19-11	AUTOSAR Release Management	• Added recovery action via application • Usage of `ara::core` types in PHM APIs • Set data types to uint32_t by default • Editorial rework of chapters 7 and 8 • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Modified the API for Supervised Entity and Health Channel • Modified the interface with the Execution Manager
2018-10-31	18-10	AUTOSAR Release Management	• Described the interfaces with functional clusters execution management and state management
2018-03-29	18-03	AUTOSAR Release Management	• 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 Platform Health Management].

It also implements the general functionality described in the Foundation documents [3, RS Health Monitoring] and [4, ASWS Health Monitoring]. In addition to the functionality specified in [4], this document also defines Health Channel Supervision.

Health Monitoring is required by [5, ISO 26262:2018] (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:
E2E	AUTOSAR End to End communication protection mechanism
PHM	Platform Health Management
SE	Supervised Entity

Acronym:	Description:
Alive Supervision	Mechanism to check the timing constraints of cyclic Supervised Entitys to be within the configured min and max limits.
ara::com	Communication middleware for the AUTOSAR Adaptive Platform
AUTOSAR Adaptive Platform	see [6] AUTOSAR Glossary
Checkpoint	A point in the control flow of a Supervised Entity where the activity is reported.
Daisy chaining	Chaining multiple instances of Health Monitoring
Deadline Supervision	Mechanism to check that the timing constraints for execution of the transition from a Deadline Start Checkpoint to a corresponding Deadline End Checkpoint are within the configured min and max limits.
Function Group	A Function Group is a set of coherent Processs, which need to be controlled consistently. Depending on the state of the Function Group, Processes are started or terminated.
Global Supervision Status	Status that summarizes the Local Supervision Status of all Supervised Entitys of a software subsystem.
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	Check 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
Process	A Process is a loaded instance of an executable to be executed on a machine.
Supervised Entity	A whole or part of a SwComponentType which is included in the supervision. A Supervised Entity denotes a collection of Checkpoints within the corresponding SwComponentType. A SwComponentType can include zero, one or more Supervised Entities. A Supervised Entity may be instantiated multiple times, in which case each instance is independently supervised.
Supervision Mode	State of a machine or Function Group in which Supervised Entity Instances are to be monitored with a specific set of configuration parameters. Supervision parameters differ from one mode to other as the behavior (timing or sequence) of Supervised Entity changes from one mode to other. Modes are mutually exclusive. A mode can be "Normal", "Degradation".

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 Platform Health Management
AUTOSAR_RS_PlatformHealthManagement

[3] Requirements on Health Monitoring
AUTOSAR_RS_HealthMonitoring

[4] Specification of Health Monitoring
AUTOSAR_ASWS_HealthMonitoring

[5] ISO 26262:2018 (all parts) – Road vehicles – Functional Safety
http://www.iso.org

[6] Glossary
AUTOSAR_TR_Glossary

[7] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[8] Specification of Adaptive Platform Core
AUTOSAR_SWS_AdaptivePlatformCore

[9] Specification of State Management
AUTOSAR_SWS_StateManagement

[10] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[11] Guidelines for using Adaptive Platform interfaces
AUTOSAR_EXP_AdaptivePlatformInterfacesGuidelines

[12] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

3.2 Further applicable specification

AUTOSAR provides a general specification [7, SWS_BSWGeneral] which is also applicable for `Platform Health Management`. The specification SWS General shall be considered as additional and required specification for implementation of `Platform Health Management`.

AUTOSAR provides a core specification [8] which is also applicable for `Platform Health Management`. The chapter "General requirements for all FunctionalClusters"

of this specification shall be considered as an additional and required specification for implementation of Platform Health Management.

4 Constraints and assumptions

4.1 Known limitations

- 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.
- Interface with the Diagnostic Manager is not specified in this release.
- Health Channels (HealthChannelExternalStatus) are set to obsolete. They are expected to be introduced at State Management in the next release.
- The configuration attribute for the alive notification cycle time (with respect to PHM sending AliveNotification to watchdog interface) is not specified for this release.
- A change in the value of Supervision (Alive/Deadline/Logical) configuration parameters between two Function Group states wherein the process being supervised continues to execute on switching between these states is not considered. The Supervision continues as per configuration in the Supervision Mode corresponding to old Function Group state.
- Similar to above limitation, dynamic change between Supervision exclusion (disable) and Supervision inclusion (enable) on Function Group state change wherein the process under consideration continues to execute on change in Function Group state is not supported. Supervision exclusion or inclusion can be applied starting with the Function Group state in which execution of the process begins and the same is applied until termination of the process.
- Currently specified mechanism of Notifying State Management on Global Supervision Status reaching state kStopped is insufficient in case of multiple failures. It could happen that the Global Supervision Status remains in state kStopped without further notification to State Management about successive failures. Thereby the recovery might be hindered.
- "PowerMode" dependent Supervision configuration is not supported in this release. See [9] for information on "PowerMode".
- Exact point in time at which Alive Supervision is to be started and stopped is not yet specified.

4.2 Applicability to car domains

No restriction

5 Dependencies to other Functional Clusters

5.1 Platform dependencies

The interfaces within AUTOSAR Adaptive Platform are not standardized.

5.1.1 Dependencies on Execution Management

The Platform Health Management functional cluster is dependent on the Execution Management Interface [10].

The Platform Health Management functional cluster might need some Process information from Execution Management Interface [10]. The exact form of the information is vendor specific and therefore not standardized by AUTOSAR. However it is expected to include process states and function group states.

5.1.2 Dependencies on State Management

The Platform Health Management functional cluster has an interface also with the State Management: If a failure is detected within a Supervised Entity or via Health Channel, Platform Health Management notifies State Management on this failure.

5.1.3 Dependencies on Watchdog Interface

The Platform Health Management functional cluster is dependent also on the Watchdog Interface.

5.1.4 Dependencies on other Functional Clusters

It is possible for all functional clusters to use the Supervision mechanisms provided by the Platform Health Management by using Checkpoints and the Health Channels as the other Applications.

5.2 Protocol layer dependencies

None.

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_AP_00119]	Return values / application errors.	[SWS_PHM_01240] [SWS_PHM_01241]
[RS_HM_09159]	Health Monitoring shall be able to report supervision errors.	[SWS_PHM_00101] [SWS_PHM_00102] [SWS_PHM_00104] [SWS_PHM_01147] [SWS_PHM_01148]
[RS_HM_09226]	Health Monitoring shall be able to wrongly trigger the serviced watchdogs.	[SWS_PHM_00104] [SWS_PHM_00105] [SWS_PHM_00106]
[RS_HM_09237]	Health Monitoring shall provide an interface to Supervised Entities informing them about their Supervision State.	[SWS_PHM_00100] [SWS_PHM_01136] [SWS_PHM_01137] [SWS_PHM_01146]
[RS_HM_09249]	Health Monitoring shall support building safety-related systems.	[SWS_PHM_00010] [SWS_PHM_00100] [SWS_PHM_00101] [SWS_PHM_00102] [SWS_PHM_00104] [SWS_PHM_00105] [SWS_PHM_00106]
[RS_HM_09254]	Health Monitoring shall provide an interface to Supervised Entities to report the currently reached Checkpoint.	[SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_00426] [SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01132] [SWS_PHM_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01227] [SWS_PHM_01228] [SWS_PHM_01229]
[RS_IAM_00002]	Position of Policy Enforcement	[SWS_PHM_01229] [SWS_PHM_01330]
[RS_IAM_00010]	Adaptive applications shall only be able to use AUTOSAR Resources when authorized	[SWS_PHM_01229] [SWS_PHM_01330]

Requirement	Description	Satisfied by
[RS_PHM_00001]	The `Platform Health Management` shall provide a standardized header file structure for each service.	[SWS_PHM_00457] [SWS_PHM_01002] [SWS_PHM_01020] [SWS_PHM_01114] [SWS_PHM_01115] [SWS_PHM_01122] [SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01128] [SWS_PHM_01132] [SWS_PHM_01146] [SWS_PHM_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01221] [SWS_PHM_01222] [SWS_PHM_01223] [SWS_PHM_01224] [SWS_PHM_01225]
[RS_PHM_00002]	The service header files shall define the namespace for the respective service.	[SWS_PHM_00457] [SWS_PHM_01005] [SWS_PHM_01113] [SWS_PHM_01122] [SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01128] [SWS_PHM_01132] [SWS_PHM_01146] [SWS_PHM_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01221] [SWS_PHM_01222] [SWS_PHM_01223] [SWS_PHM_01224] [SWS_PHM_01225]

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_00426] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01122] [SWS_PHM_01129] [SWS_PHM_01132] [SWS_PHM_01138] [SWS_PHM_01139] [SWS_PHM_01140] [SWS_PHM_01141] [SWS_PHM_01142] [SWS_PHM_01143] [SWS_PHM_01144] [SWS_PHM_01145] [SWS_PHM_01149] [SWS_PHM_01150] [SWS_PHM_01151] [SWS_PHM_01152] [SWS_PHM_01231] [SWS_PHM_01232] [SWS_PHM_01233] [SWS_PHM_01234] [SWS_PHM_01235] [SWS_PHM_01236] [SWS_PHM_01237] [SWS_PHM_01238] [SWS_PHM_01239]
[RS_PHM_00101]	Platform Health Management shall provide a standardized C++ interface for the reporting of Checkpoints.	[SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_00426] [SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01132] [SWS_PHM_01146] [SWS_PHM_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01227] [SWS_PHM_01228] [SWS_PHM_01229]

Requirement	Description	Satisfied by
[RS_PHM_00102]	Platform Health Management shall provide a standardized C++ interface for the reporting of Health Channel.	[SWS_PHM_00457] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01122] [SWS_PHM_01128] [SWS_PHM_01129] [SWS_PHM_01221] [SWS_PHM_01222] [SWS_PHM_01223] [SWS_PHM_01224] [SWS_PHM_01225] [SWS_PHM_01328] [SWS_PHM_01329] [SWS_PHM_01330]
[RS_PHM_00104]	Platform Health Management shall derive the Supervision Mode from Function Group State(s).	[SWS_PHM_00211] [SWS_PHM_00212] [SWS_PHM_00213] [SWS_PHM_00214] [SWS_PHM_00240] [SWS_PHM_00241] [SWS_PHM_00242] [SWS_PHM_00243] [SWS_PHM_00244] [SWS_PHM_00245]
[RS_PHM_00108]	Platform Health Management shall provide a standardized interface between Platform Health Management components used in a daisy chain.	[SWS_PHM_NA]
[RS_PHM_00109]	Platform Health Management shall provide the Daisy chaining interface over ara::com.	[SWS_PHM_NA]

Requirement	Description	Satisfied by
[RS_PHM_00111]	Platform Health Management shall determine Supervision status	[SWS_PHM_00201] [SWS_PHM_00202] [SWS_PHM_00203] [SWS_PHM_00204] [SWS_PHM_00205] [SWS_PHM_00206] [SWS_PHM_00207] [SWS_PHM_00208] [SWS_PHM_00209] [SWS_PHM_00210] [SWS_PHM_00211] [SWS_PHM_00212] [SWS_PHM_00213] [SWS_PHM_00214] [SWS_PHM_00215] [SWS_PHM_00216] [SWS_PHM_00217] [SWS_PHM_00218] [SWS_PHM_00219] [SWS_PHM_00220] [SWS_PHM_00221] [SWS_PHM_00222] [SWS_PHM_00223] [SWS_PHM_00224] [SWS_PHM_00225] [SWS_PHM_00226] [SWS_PHM_00227] [SWS_PHM_00228] [SWS_PHM_00229] [SWS_PHM_00230] [SWS_PHM_00231] [SWS_PHM_00232] [SWS_PHM_00233] [SWS_PHM_00234] [SWS_PHM_00235] [SWS_PHM_00236] [SWS_PHM_00237] [SWS_PHM_00238] [SWS_PHM_00239]
[RS_PHM_00112]	Platform Health Management shall provide configurable delays of error reactions.	[SWS_PHM_00224] [SWS_PHM_00225] [SWS_PHM_00228] [SWS_PHM_00229] [SWS_PHM_00230] [SWS_PHM_00231] [SWS_PHM_00238] [SWS_PHM_00239]
[RS_PHM_09240]	Platform Health Management shall support multiple occurrences of the same Supervised Entity.	[SWS_PHM_01123] [SWS_PHM_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215]

Requirement	Description	Satisfied by
[RS_PHM_09241]	Health Monitoring shall support multiple instances of Checkpoints in a Supervised Entity occurrence.	[SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_00426]
[RS_PHM_09255]	`Platform Health Management` shall provide an interface to receive Health Channel supervision status	[SWS_PHM_00010] [SWS_PHM_00102]
[RS_PHM_09257]	`Platform Health Management` shall provide an interface to Supervised Entities to report their health status.	[SWS_PHM_00457] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01122] [SWS_PHM_01128] [SWS_PHM_01129] [SWS_PHM_01221] [SWS_PHM_01222] [SWS_PHM_01223] [SWS_PHM_01224] [SWS_PHM_01225] [SWS_PHM_01328] [SWS_PHM_01329] [SWS_PHM_01330]

7 Functional specification

7.1 General description

The Platform Health Management monitors applications with respect to timing constraints (Alive Supervision and Deadline Supervision) and logical program sequence (Logical Supervision) as well as platform health (Health Channel Supervision). In case of a detected failure, Platform Health Management notifies State Management. As coordinator of the platform, State Management can decide how to handle the error and trigger a suitable recovery action.

Platform Health Management has also an interface to the hardware watchdog and can trigger a watchdog reaction in case of a critical failure where a notification to State Management is not sufficient.

All the algorithms and the procedures for the Platform Health Management are described in the Autosar Foundation document [4] and are not specified here: only the Autosar Adaptive specificities, including the interfaces with the other functional clusters, are shown here below.

The interfaces of Health Management to other Functional Clusters are only informative and are not standardized.

7.2 Supervision of Supervised Entities

State Management coordinates the platform through Function Groups [9]. Within a Function Group, there may be multiple Processes running.

Platform Health Management monitors Supervised Entitys. Each Supervised Entity maps to whole or part of a Process. The monitoring is active as long as the corresponding Process is active.

The details of the supervisions are described in [4]. The results of the supervisions of a Supervised Entity Instance are reflected in the Local Supervision Status.

The status of local Supervisions within a Function Group is conglomerated in the corresponding Global Supervision Status.

[SWS_PHM_00100]{DRAFT} **Scope of Global Supervision** ⌈The Platform Health Management shall support one or a few GlobalSupervision for a Function Group.⌋*(RS_HM_09237, RS_HM_09249)*

As described in [4], the supervisions are based on checkpoints which are reported by the Supervised Entity Instance.

[SWS_PHM_01227]{DRAFT} **Consistency of Checkpoint Identifier** ⌈The value of checkpointId reported via ReportCheckpoint shall match the declared check-

pointId of the respective PhmSupervisedEntityInterface.checkpoint.⌋ *(RS_PHM_00101, RS_HM_09254)*

[SWS_PHM_01228]{DRAFT} **Reporting of undefined Checkpoint Identifier** ⌈If a checkpointId is reported to Platform Health Management via ReportCheckpoint which is not configured in the context of the reporting SupervisedEntity, PHM shall ignore the checkpoint for evaluation of supervisions.⌋*(RS_PHM_00101, RS_HM_09254)*

[SWS_PHM_01229]{DRAFT} **Restricted access on reporting of Checkpoints** ⌈The Platform Health Management shall ignore the execution of ReportCheckpoint for evaluation of Alive, Deadline and Logical Supervision unless the calling Adaptive Application is associated with the reported Checkpoint instance by modelling, i.e., there exists a SupervisionCheckpoint with SupervisionCheckpoint.process referencing the requesting reporting application process and SupervisionCheckpoint.phmCheckpoint referencing the reported Checkpoint.⌋*(RS_PHM_00101, RS_HM_09254, RS_IAM_00002, RS_IAM_00010)*

7.2.1 Supervision of processes started before Platform Health Management

Start of Supervision (Alive Supervision/Deadline Supervision/Logical Supervision) in case of processes that are started before Platform Health Management process (e.g, process corresponding to Execution Management) is not standardized. It is up to Adaptive Platform Vendor specific decision.

7.3 Health Channel Supervision

Using Health Channel Supervision the system integrator can hook external supervision results to the Platform Health Management. External supervision can be routines like RAM test, ROM test, kernel status, voltage monitoring etc. The external supervision performs the monitoring and debouncing. The determined result is classified according to the possible Health Status values and sent to Platform Health Management.

A Health Channel can be

- the Global supervision status of the software under supervision.
- the result of an environment monitoring algorithm. e.g. Voltage Monitoring, Temperature Monitoring.
- the result of a memory integrity test routine, e.g. RAM test, ROM test.
- the status of the operating system or Kernel. e.g. OS Status, Kernel Status.
- the status of another platform instance or Virtual Machine or ECU.

The various external monitoring routines shall report their result or status in the form of defined Health Statuses to the Platform Health Management. The Health Status of a Health Channel is the abstract format of the information that a Health Channel provides to the Platform Health Management. Two different Health Channels may have same Health Status names to represent its result, e.g. high, low, normal.

If a reaction on a determined Health Status is necessary, Platform Health Management reports the status to State Management.

7.3.1 Health Status after Initialization

The Health Status after initialization is controlled by the configuration container HealthStatusInitValue. This parameter may be configured once for each Health Channel in the configuration.

[SWS_PHM_00010]{OBSOLETE} **Not initialized Health Channel** ⌈If the container HealthStatusInitValue does not exist or the Health Channel does not already have an initial value, the Platform Health Management shall treat the corresponding Health Status as undefined and not use it until the corresponding Health Channel has been updated for the first time.⌋*(RS_PHM_09255, RS_HM_09249)*

7.3.2 Configuration of Health Channel

A Health Channel has the following configuration options:

1. Name: Globally unique name identifier, used by Applications.
2. ID: Globally unique identifier (number)
3. HealthStatusInitValue: Initial value of the corresponding Health Status.

A Health Status represents a possible value of the Health Channel and has the following options:

1. Name: used by Applications, unique within the Health Channel
2. ID: Identifier of the Health Status, unique within the Health Channel.

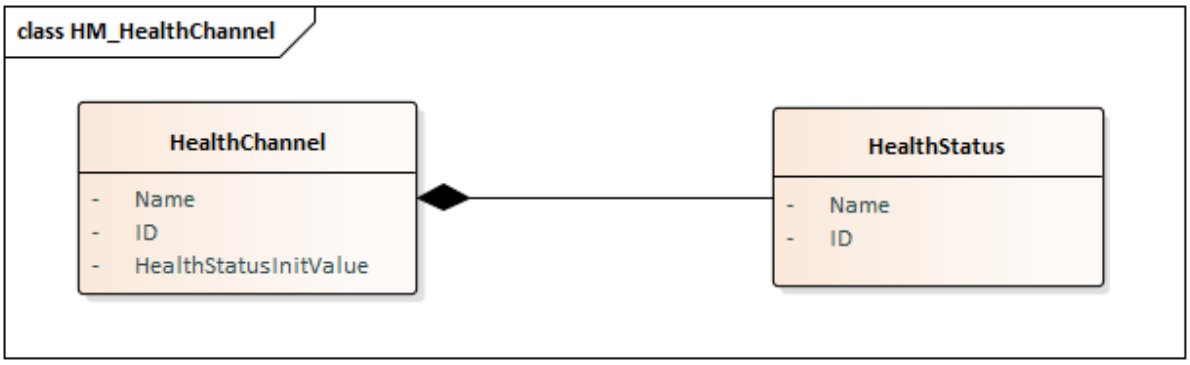

Figure 7.1: `Health Channel` configuration

7.3.3 Reporting of Health Channel

The current `Health Status` is reported to `Platform Health Management` via the method `ReportHealthStatus`.

[SWS_PHM_01328]{OBSOLETE} **Consistency of Health Status Identifier** ⌈The value of `healthStatusId` reported via `ReportHealthStatus` shall match the declared `statusId` of the respective `PhmHealthChannelInterface.status`.⌋*(RS_PHM_00102, RS_PHM_09257)*

[SWS_PHM_01329]{OBSOLETE} **Reporting of undefined Health Status Identifier** ⌈If a `healthStatusId` is reported to `Platform Health Management` and no corresponding `PhmHealthChannelStatus` is configured in the context of the reporting `PhmHealthChannelInterface`, PHM shall ignore the reporting of healthStatus.⌋ *(RS_PHM_00102, RS_PHM_09257)*

[SWS_PHM_01330]{OBSOLETE} **Restricted access on reporting of Health Status** ⌈The execution of `ReportHealthStatus` shall be prevented (i.e, shall not be considered for evaluation) unless the calling Adaptive Application is associated with the reported `Health Status` instance by modelling, i.e., there exists a `HealthChannelExternalStatus` with `HealthChannelExternalStatus.process` referencing the requesting reporting application process and `HealthChannelExternalStatus.notifiedStatus` referencing the reported `Health Status` instance.⌋*(RS_PHM_00102, RS_PHM_09257, RS_IAM_00002, RS_IAM_00010)*

7.4 Supervision Modes

Expected execution (timing or sequence) of the Software can change based on certain conditions. Hence, the value of the Supervision (Alive/Deadline/Logical) parameters might have to be changed based on conditions. For each such condition a mode called a `Supervision Mode` can be configured. Currently, this condition can be configured based on `Function Group` State.

Note: It is possible to exclude (disable) Supervision for a Supervised Entity Instance in a Supervision Mode. This can be achieved by configuring NoSupervision for the Supervised Entity Instance in the Supervision Mode.

7.4.1 Effect of changing Mode

In AUTOSAR Adaptive Platform, Supervision Mode changes on Function Group State change.

Function Group State change has following impact on processes:

- Certain processes are terminated.
- Certain processes are newly started.
- Certain processes are restarted.
- Remaining processes continue to execute.

Supervisions (Alive, Deadline and Logical) of the Supervised Entitys corresponding to the processes shall be handled as follows.

[SWS_PHM_00240]{DRAFT} **Supervisions on termination of process** ⌈Alive Supervision, Deadline Supervision and Logical Supervision shall be stopped on termination of the corresponding process. Results of Alive, Deadline and Logical Supervision shall be set to correct.⌋ *(RS_PHM_00104)*

The termination of the process could be due to various reasons. It could be due to change in Function Group State (the process is not configured to be executed in the new Function Group State), a self-terminating process is terminating on its own or abrupt termination of a process (e.g. due to out of bound memory access).

Note:

1. On termination of process, Local Supervision Status of the corresponding Supervised Entity Instance will be set to LOCAL_STATUS_DEACTIVATED.
2. For a process, monitoring is active when the process is executing (that is, when the Execution state of the process is "Initializing" or "Running" or "Terminating"). It is deactivated (stopped) when the process is terminated.

[SWS_PHM_00241]{DRAFT} **Supervisions on Start of Process** ⌈On start of the process for which a Supervision (Alive Supervision, Deadline Supervision and/or Logical Supervision) is configured in the new Function Group State, the Supervision (Alive Supervision, Deadline Supervision and/or Logical Supervision) shall be performed as per the configured Supervision parameter values in the Supervision Mode corresponding to new Function Group State.⌋ *(RS_PHM_00104)*

[SWS_PHM_00244]{DRAFT} **NoSupervision on Start of Process** ⌈On start of the process in the new Function Group State, if NoSupervision is configured for

a Supervised Entity Instance corresponding to the process in the Supervision Mode corresponding to the new Function Group State, then no Supervision (no Alive Supervision, Deadline Supervision or Logical Supervision) shall be performed for the Supervised Entity Instance in the Supervision Mode corresponding to new Function Group State.⌋(*RS_PHM_00104*)

Note: Even though it is supported to exclude (disable) Supervision in a particular Supervision Mode, dynamic change between Supervision inclusion (enable) and exclusion (disable) during execution of Process is not supported. Supervision exclusion can be applied starting from the Supervision Mode corresponding to the Function Group state in which the execution of the process is started. Supervision exclusion continues until the termination of the process. The same principle applies to a change in supervision parameters.

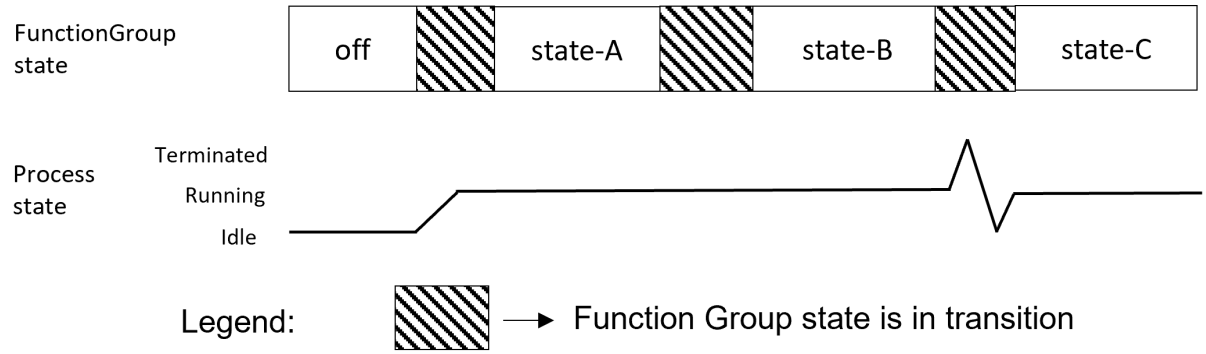

Figure 7.2: Supervision Exclusion and change of Function Group State

Figure 7.2 shows an example: If Supervision is excluded in Function Group state-A, same will continue in Function Group state-B. Supervision can be applied again in state-C wherein the process is restarted (but not in state-B).

[SWS_PHM_00242]{DRAFT} **Supervisions on Restart of Process** ⌈Supervisions on restart of a process due to Function Group State change shall be handled as termination of process (see [SWS_PHM_00240]) followed by start of process (see [SWS_PHM_00241]).⌋(*RS_PHM_00104*)

[SWS_PHM_00243]{DRAFT} **Continuation of Supervisions** ⌈Supervisions (Alive, Deadline and Logical) shall be continued with same values of Supervision parameters if the corresponding process continues to execute on Function Group State change.⌋ (*RS_PHM_00104*)

[SWS_PHM_00245]{DRAFT} **Continuation of NoSupervision (Supervision Exclusion)** ⌈If NoSupervision is configured for a Supervised Entity Instance in the Supervision Mode corresponding to the Function Group State, in which the execution of the corresponding process starts, then no Supervision (no Alive Supervision, Deadline Supervision or Logical Supervision) shall be continued on change in Function Group State to a new state if the process continues to execute on Function Group State change.⌋(*RS_PHM_00104*)

7.5 Determination of Supervision Status

Based on the results of Alive Supervision, Deadline Supervision and Logical Supervision the Local Supervision Status and Global Supervision Status are determined. Please refer [4] for details of these Supervisions.

7.5.1 Determination of Local Supervision Status

The Local Supervision Status state machine determines the status of the Supervised Entity Instance. This is done based on the following:

1. Previous value of the Local Supervision Status,
2. Current values of: result of Alive Supervisions, result of Deadline Supervisions, result of Logical Supervisions involving SupervisionCheckpoints corresponding to the Supervised Entity Instance.

The state machine is initialized at the initialization of the Platform Health Management.

[SWS_PHM_00201]{DRAFT} ⌈The Platform Health Management shall track the Local Supervision Status of each Supervised Entity Instance, see ara::phm::LocalSupervisionStatus.⌋*(RS_PHM_00111)*

Figure 7.3 shows the state machine for Local Supervision Status of a Supervised Entity Instance with all possible states.

[SWS_PHM_00202]{DRAFT} ⌈The Platform Health Management shall have the local statuses LOCAL_STATUS_OK, LOCAL_STATUS_DEACTIVATED, LOCAL_STATUS_EXPIRED and LOCAL_STATUS_FAILED, see ara::phm::LocalSupervisionStatus.⌋*(RS_PHM_00111)* See also figure 7.3.

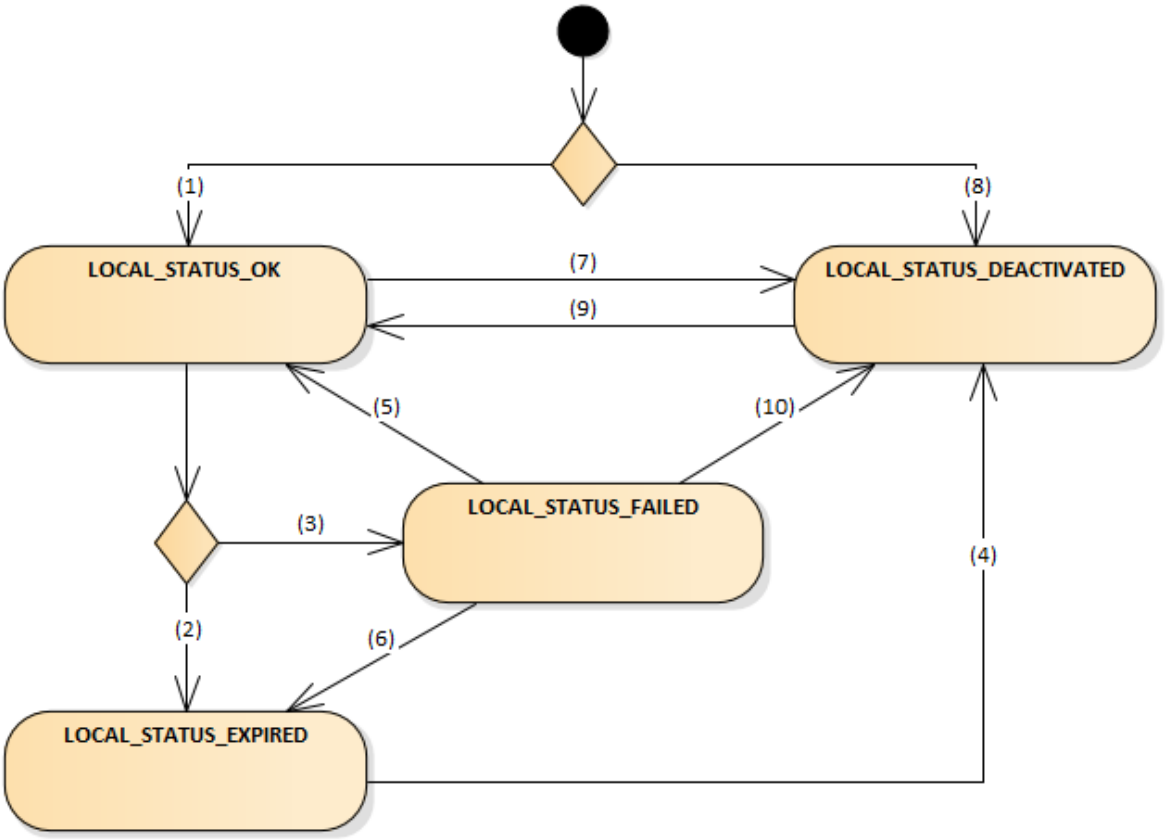

Figure 7.3: Local Supervision Status

For the transitions between the states of the `Local Supervision Status` the following rules apply:

[SWS_PHM_00203]{DRAFT} ⌈If the `Supervised Entity` Instance is configured to be monitored in the current `Supervision Mode` (that is, at least one `Alive Supervision` / `Deadline Supervision` / `Logical Supervision` referencing `SupervisionCheckpoint` corresponding to the `Supervised Entity` Instance is configured in the current `Supervision Mode`), once the process corresponding to the `Supervised Entity` Instance is started, the `Platform Health Management` shall set the `Local Supervision Status` for this `Supervised Entity` Instance to LOCAL_STATUS_OK and the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111)* See transition (1) in figure 7.3.

[SWS_PHM_00204]{DRAFT} ⌈If the `Supervised Entity` Instance is not configured to be monitored in the current `Supervision Mode` (that is `NoSupervision` is configured for the `Supervised Entity` Instance in the current `Supervision Mode`) or if the `Supervised Entity` Instance is configured to be monitored in the current `Supervision Mode` but the process corresponding to the `Supervised Entity` Instance is not yet started, then the `Platform Health Management` shall set the `Local Supervision Status` for this `Supervised Entity` Instance to LOCAL_STATUS_DEACTIVATED and the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111)* See transition (8) in figure 7.3.

[SWS_PHM_00205]{DRAFT} ⌈If all results of Deadline Supervision and Logical Supervision for the Supervised Entity Instance are correct, there is no failure in any Alive Supervision (counter for failed Alive Supervision reference cycles is zero) of the Supervised Entity Instance and the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_OK, then the Platform Health Management shall keep the Supervised Entity Instance in the Local Supervision Status LOCAL_STATUS_OK.⌋*(RS_PHM_00111)*

[SWS_PHM_00206]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_OK AND:

1. At least one Alive Supervision of the Supervised Entity Instance has permanent failure (counter for failed Alive Supervision reference cycles exceeds failure folerance failedReferenceCyclesTolerance) OR
2. If the result of at least one Deadline Supervision of the Supervised Entity Instance or the result of at least one Logical Supervision of the Supervised Entity Instance is incorrect,

THEN the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_EXPIRED. Supervisions (Alive Supervision, Deadline Supervision and Logical Supervision) shall be stopped on reaching state LOCAL_STATUS_EXPIRED. Supervisions shall be restarted only on reaching state LOCAL_STATUS_OK.⌋*(RS_PHM_00111)* See transition (2) in figure 7.3.

The below requirements shows the important difference of Alive Supervision versus Deadline Supervision and Logical Supervision: the Alive Supervision has an error tolerance for failed reference cycles.

[SWS_PHM_00207]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_OK AND:

1. At least one Alive Supervision of the Supervised Entity Instance has temporary failure (counter for failed Alive Supervision reference cycles is greater than zero but does not exceed failure tolerance failedReferenceCyclesTolerance) AND
2. None of the Alive Supervisions of the Supervised Entity Instance has permanent failure (counter for failed Alive Supervision reference cycles exceeds failure tolerance failedReferenceCyclesTolerance) AND
3. If all the results of Deadline Supervision and Logical Supervision of the Supervised Entity Instance are correct,

THEN the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_FAILED.⌋*(RS_PHM_00111)* See transition (3) in figure 7.3.

[SWS_PHM_00208]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. At least one Alive Supervision of the Supervised Entity Instance has temporary failure (counter for failed Alive Supervision reference cycles is greater than zero but does not exceed failure tolerance failedReferenceCyclesTolerance) AND
2. None of the Alive Supervisions of the Supervised Entity Instance has permanent failure (counter for failed Alive Supervision reference cycles exceeds failure tolerance failedReferenceCyclesTolerance) AND
3. If all the results of Deadline Supervision and Logical Supervision of the Supervised Entity Instance are correct,

THEN the Platform Health Management shall keep the Local Supervision Status in LOCAL_STATUS_FAILED.⌋*(RS_PHM_00111)*

[SWS_PHM_00209]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. If there is no failure (counter for failed Alive Supervision reference cycles is zero) in any Alive Supervisions of the Supervised Entity Instance AND
2. If all the results of Deadline Supervision and Logical Supervision of the Supervised Entity Instance are correct,

THEN the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_OK.⌋*(RS_PHM_00111)* See transition (5) in figure 7.3.

[SWS_PHM_00210]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. If at least one Alive Supervisions of the Supervised Entity Instance has permanent failure (counter for failed Alive Supervision reference cycles exceeds failure tolerance failedReferenceCyclesTolerance) OR
2. If at least one result of Deadline Supervision or Logical Supervision of the Supervised Entity Instance is incorrect,

THEN the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_EXPIRED. Supervisions (Alive Supervision, Deadline Supervision and Logical Supervision) shall be stopped on reaching state LOCAL_STATUS_EXPIRED. Supervisions shall be restarted only on reaching state LOCAL_STATUS_OK.⌋*(RS_PHM_00111)* See transition (6) in figure 7.3.

[SWS_PHM_00211]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_OK and if the process corresponding to the Supervised Entity Instance is terminated, then the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_DEACTIVATED and the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111, RS_PHM_00104)* See transition (7) in figure 7.3.

[SWS_PHM_00212]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_FAILED and if the process corresponding to the Supervised Entity Instance is terminated, then the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_DEACTIVATED and the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111, RS_PHM_00104)* See transition (10) in figure 7.3.

[SWS_PHM_00213]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_DEACTIVATED then, unless

- there is a switch to a Supervision Mode wherein the Supervised Entity Instance is configured to be monitored AND
- the execution of the corresponding process is started,

the Platform Health Management shall not perform any Supervisions for this Supervised Entity and keep the Local Supervision Status in the state LOCAL_STATUS_DEACTIVATED.⌋*(RS_PHM_00111, RS_PHM_00104)*

[SWS_PHM_00214]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_DEACTIVATED and there is a switch to a Supervision Mode (due to change in corresponding Function Group State) in which the Supervised Entity Instance is configured to be monitored, once the execution of the process corresponding to the Supervised Entity Instance starts in the new Supervision Mode, the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_OK.⌋*(RS_PHM_00111, RS_PHM_00104)* See transition (9) in figure 7.3.

[SWS_PHM_00215]{DRAFT} ⌈If the Supervised Entity Instance was in Local Supervision Status LOCAL_STATUS_EXPIRED and the corresponding process is terminated (this could be due to recovery action of change in Function Group State which terminates the faulty process), then the Platform Health Management shall change the Local Supervision Status to LOCAL_STATUS_DEACTIVATED and the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111)* See transition (4) in figure 7.3.

7.5.2 Determination of Global Supervision Status

Based on the Local Supervision Status of all Supervised Entity Instances of a software subsystem, the Global Supervision Status is computed. Part/whole of a Function Group is considered as a subsystem.

The Global Supervision Status has similar values as the Local Supervision Status. The main differences are the addition of the GLOBAL_STATUS_STOPPED value. Figure 7.4 shows the values and transitions between them.

A change in Global Supervision Status can be logged by Platform Health Management for test/debugging purposes.

Global Supervision Status is "worst-of" all included Local Supervision Statuses.

[SWS_PHM_00219]{DRAFT} ⌈The Platform Health Management shall calculate the Global Supervision Status of each configured GlobalSupervision.⌋ *(RS_PHM_00111)*

Whether the evaluation of Global Supervision Status and the Local Supervision Status that it aggregates is time triggered (periodic evaluation) or event triggered (on availability of a new result for Alive Supervision / Deadline Supervision / Logical Supervision) is up to Adaptive Platform Vendor's decision.

[SWS_PHM_00216]{DRAFT} ⌈The Platform Health Management shall have the Global Supervision Statuses GLOBAL_STATUS_OK, GLOBAL_STATUS_DEACTIVATED, GLOBAL_STATUS_FAILED, GLOBAL_STATUS_EXPIRED and GLOBAL_STATUS_STOPPED, see ara::phm::GlobalSupervisionStatus.⌋ *(RS_PHM_00111)* See also figure 7.4.

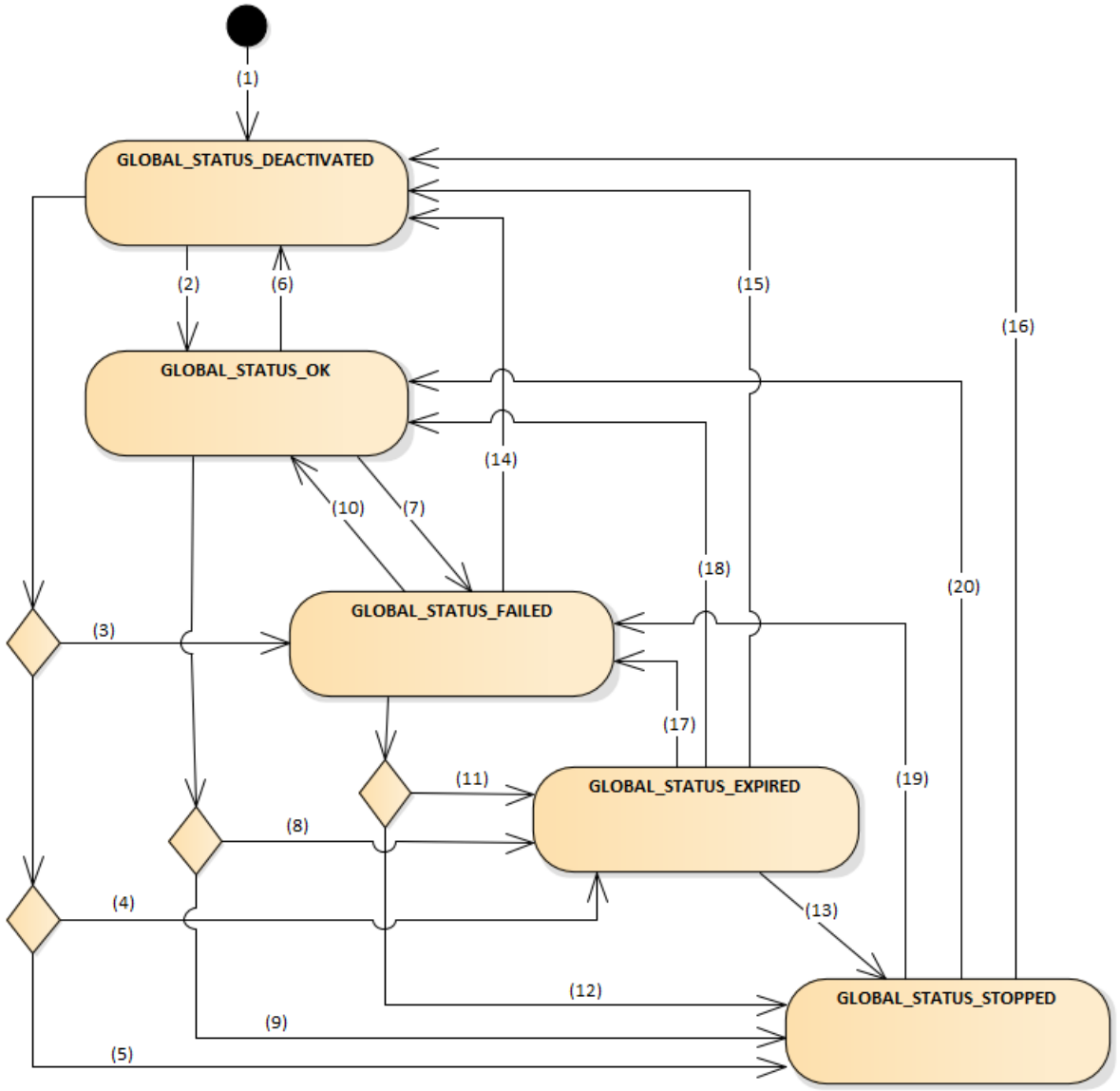

Figure 7.4: Global Supervision Status

[SWS_PHM_00217]{DRAFT} ⌈The `Platform Health Management` shall have one `Global Supervision Status` for a software subsystem.⌋*(RS_PHM_00111)* Note: In `AUTOSAR Adaptive Platform` whole or part of a `Function Group` is considered as software subsystem.

[SWS_PHM_00218]{DRAFT} ⌈The `Global Supervision Status` shall be initialized with GLOBAL_STATUS_DEACTIVATED.⌋*(RS_PHM_00111)* See transition (1) in figure 7.4.

The `Platform Health Management` provides a feature to postpone the error reaction (the error reaction being not setting a correct watchdog trigger condition) for a configurable amount of time, named `expiredSupervisionTolerance`.

The Expired Supervision Tolerance is implemented within the state machine of the Global Supervision Status. The defined state machine is in the state GLOBAL_STATUS_EXPIRED while the error reaction is postponed.

[SWS_PHM_00220]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_DEACTIVATED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_ OK and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_FAILED or LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_OK.⌋*(RS_PHM_00111)* See transition (2) in figure 7.4.

[SWS_PHM_00221]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_OK and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_FAILED or LOCAL_STATUS_EXPIRED, then the Platform Health Management shall keep the Global Supervision Status GLOBAL_STATUS_OK.⌋*(RS_PHM_00111)*

[SWS_PHM_00222]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_OK or GLOBAL_STATUS_FAILED or GLOBAL_STATUS_EXPIRED or GLOBAL_STATUS_STOPPED AND the Local Supervision Status of all Supervised Entitys is LOCAL_STATUS_DEACTIVATED, then the Platform Health Management shall set the Global Supervision Status to GLOBAL_STATUS_DEACTIVATED and stop measuring Expired Supervision Time.⌋*(RS_PHM_00111)* See transitions (6), (14), (15) and (16) in figure 7.4.

[SWS_PHM_00223]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_FAILED and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_FAILED.⌋*(RS_PHM_00111)* See transition (7) in figure 7.4.

The Platform Health Management supports a feature to delay the error reaction (switching to GLOBAL_STATUS_STOPPED) for a configurable amount of time. This could be used to allow clean-up activities before a watchdog reset, e.g. writing the error cause, writing NVRAM data.

[SWS_PHM_00224]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the expiredSupervisionTolerance is configured to a value larger than zero, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_EXPIRED and start measuring Expired Supervision Time.⌋ *(RS_PHM_00111, RS_PHM_00112)* See transition (8) in figure 7.4.

[SWS_PHM_00225]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the expiredSupervisionTolerance is configured to zero, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED.⌋ *(RS_PHM_00111, RS_PHM_00112)* See transition (9) in figure 7.4.

[SWS_PHM_00226]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_ FAILED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_FAILED and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Platform Health Management shall keep the Global Supervision Status GLOBAL_STATUS_FAILED.⌋*(RS_PHM_00111)*

[SWS_PHM_00227]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_ FAILED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_OK and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_FAILED or LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_OK.⌋*(RS_PHM_00111)* See transition (10) in figure 7.4.

[SWS_PHM_00228]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_FAILED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the expiredSupervisionTolerance is configured to a value larger than zero, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_EXPIRED and start measuring Expired Supervision Time.⌋ *(RS_PHM_00111, RS_PHM_00112)* See transition (11) in figure 7.4.

[SWS_PHM_00229]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_FAILED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the expiredSupervisionTolerance is configured to zero, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED.⌋*(RS_PHM_00111, RS_PHM_00112)* See transition (12) in figure 7.4.

[SWS_PHM_00230]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_EXPIRED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the measured Expired Supervision Time is less than the configured expiredSupervisionTolerance, then the Platform Health Management shall keep the Global Supervision Status GLOBAL_STATUS_EXPIRED.⌋*(RS_PHM_00111, RS_PHM_00112)*

[SWS_PHM_00231]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_EXPIRED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the measured

Expired Supervision Time is equal to or greater than the configured expiredSupervisionTolerance, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED.⌋(*RS_PHM_00111, RS_PHM_00112*) See transition (13) in figure 7.4.

[SWS_PHM_00232]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_STOPPED and the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED, then the Platform Health Management shall keep the Global Supervision Status GLOBAL_STATUS_STOPPED.⌋(*RS_PHM_00111*)

[SWS_PHM_00233]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_EXPIRED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_OK and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_DEACTIVATED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_OK and stop measuring Expired Supervision Time.⌋(*RS_PHM_00111*) See transition (18) in figure 7.4.

[SWS_PHM_00234]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_EXPIRED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_FAILED and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_FAILED and stop measuring Expired Supervision Time.⌋(*RS_PHM_00111*) See transition (17) in figure 7.4.

[SWS_PHM_00235]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_STOPPED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_OK and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_FAILED or LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_OK and stop measuring Expired Supervision Time.⌋(*RS_PHM_00111*) See transition (20) in figure 7.4.

[SWS_PHM_00236]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_STOPPED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_FAILED and no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_FAILED and stop measuring Expired Supervision Time.⌋(*RS_PHM_00111*) See transition (19) in figure 7.4.

[SWS_PHM_00237]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_DEACTIVATED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_FAILED and

no Supervised Entity Instance is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_FAILED.⌋(*RS_PHM_00111*) See transition (3) in figure 7.4.

[SWS_PHM_00238]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_DEACTIVATED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the expiredSupervisionTolerance is configured to a value larger than zero, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_EXPIRED and start measuring Expired Supervision Time.⌋(*RS_PHM_00111, RS_PHM_00112*) See transition (4) in figure 7.4.

[SWS_PHM_00239]{DRAFT} ⌈If the Global Supervision Status was GLOBAL_STATUS_DEACTIVATED, the Local Supervision Status of at least one Supervised Entity Instance is LOCAL_STATUS_EXPIRED and the expiredSupervisionTolerance is configured to zero, then the Platform Health Management shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED.⌋(*RS_PHM_00111, RS_PHM_00112*) See transition (5) in figure 7.4.

7.6 Recovery actions

The scope of Platform Health Management is to monitor the safety relevant Processs on the platform and report detect failures to State Management. If a failure in State Management is detected, Platform Health Management can trigger a reaction via hardware watchdog.

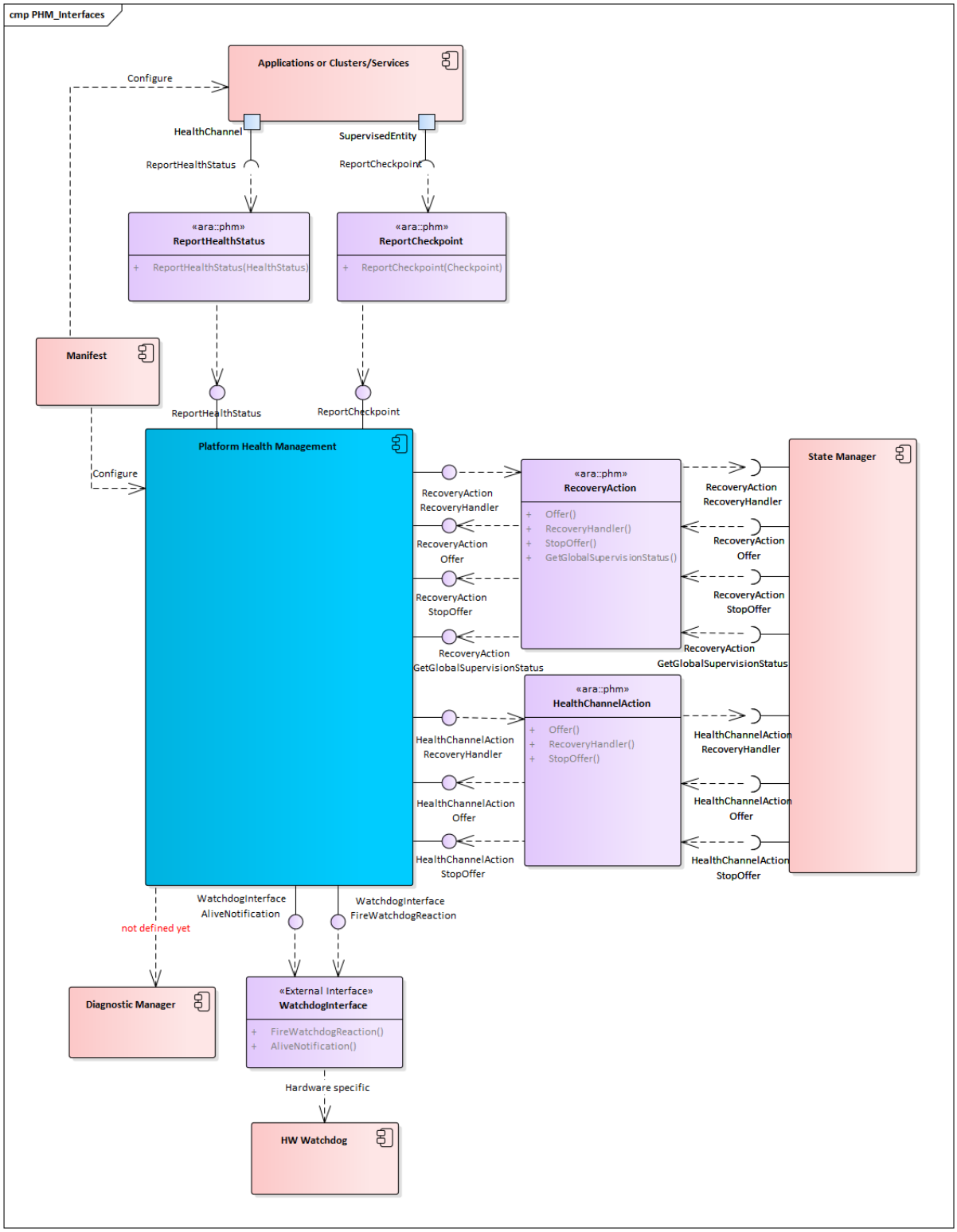

Figure 7.5: Platform Health Management and the environment

7.6.1 Notificaton to State Management

The Platform Health Management debounces the failures of Supervised Entitys, see the status LOCAL_STATUS_FAILED in chapter 7.5. After the debouncing, a recovery action is necessary. Thus, Platform Health Management notifies State Management. State Management as a coordinator of the platform can decide how a detected failure shall be handled and can trigger corresponding recovery actions. In most cases this might include switching the faulty Function Group to another state.

According to ISO 26262, it has to be ensured that a reaction is triggered after a safety-relevant failure occurred. Therefore, Platform Health Management has to make sure that State Management receives the notification on a detected failure. The Platform Health Management monitors the return of the RecoveryHandler with a configurable timeout. If State Management will not regularly return from the RecoveryHandler in time, the PHM will do its own countermeasures by wrongly triggering or stop triggering the serviced watchdog.

[SWS_PHM_00101]{DRAFT} **Notification to State Management due to Supervision failure** ⌈If the status of the mapped GlobalSupervision via RecoveryNotificationToPPortPrototypeMapping switches to state GLOBAL_STATUS_STOPPED, the Platform Health Management shall notify State Management via the method RecoveryHandler. The parameter executionError shall contain the corresponding Function Group and the current ProcessExecutionError. The parameter supervision shall contain the TypeOfSupervision which causes the transition to state GLOBAL_STATUS_STOPPED.⌋*(RS_HM_09159, RS_HM_09249)*

Note: A GlobalSupervision corresponds to whole or part of a Function Group, i.e. for each GlobalSupervision always the same Function Group is reported. The ProcessExecutionError is defined within the StartupConfig, wherefore the executionError.executionError depends on the current used StartupConfig.

[SWS_PHM_00102]{OBSOLETE} **Notification to State Management due to Health Status** ⌈If the Health Status of a Health Channel switches and a reaction of State Management is required, i.e. PhmHealthChannelStatus.triggersRecoveryNotification equals true for the corresponding PhmHealthChannelStatus.statusId, the Platform Health Management shall notify State Management via the method RecoveryHandler. The parameter healthStatusId shall be passed from the method ReportHealthStatus.⌋*(RS_HM_09159, RS_HM_09249, RS_PHM_09255)*

This means that the information about whether a reaction is required has to be configured for Platform Health Management.

[SWS_PHM_00104]{DRAFT} **Reaction on timeout for notification to State Management** ⌈If after sending a notification on a failure to State Management via the method RecoveryHandler no acknowledgment by State Management is received before RecoveryNotification.recoveryNotificationTimeout, Platform

Health Management shall stop calling WatchdogInterface::AliveNotification and call WatchdogInterface::FireWatchdogReaction.⌋(*RS_HM_09159, RS_HM_09249, RS_HM_09226*)

[SWS_PHM_01147]{DRAFT} **Enable handler** ⌈Platform Health Management shall enable potential invocations of RecoveryHandler when Offer is called.⌋(*RS_HM_09159*)

[SWS_PHM_01148]{DRAFT} **Disable handler** ⌈Platform Health Management shall disable invocations of RecoveryHandler when StopOffer is called.⌋(*RS_HM_09159*)

7.6.2 Handling of Hardware Watchdog

The Platform Health Management is the only Functional Cluster with an interface to the hardware watchdog. Therefore, the watchdog supervises Platform Health Management and PHM can initiate a reaction of the watchdog by stop triggering or by sending a false trigger. Since this reaction means usually a reset of the machine, it has an impact on all functions and should be used only as a last resort in order to ensure freedom from interference. Failures that require a watchdog reaction are supervision failures in State Management and Execution Management since in these cases a recovery action via State Management as described in section 7.6.1 is not possible.

Platform Health Management handles the hardware watchdog via the WatchdogInterface. PHM indicates aliveness to WatchdogInterface cyclically. WatchdogInterface will trigger the hardware watchdog correctly as long as PHM indicates aliveness. If PHM does not report aliveness in configured time, WatchdogInterface shall initiate watchdog reaction.

In case a critical failure is detected, PHM can trigger recovery action through WatchdogInterface.

[SWS_PHM_00106]{DRAFT} **Recovery Action for Failures in Execution or State Management** ⌈As long as no Global Supervision Status corresponding to State Management or Execution Management has reached state GLOBAL_STATUS_STOPPED and Notification to State Management has not failed, Platform Health Management shall call WatchdogInterface::AliveNotification periodically.⌋(*RS_HM_09249, RS_HM_09226*)

[SWS_PHM_00105]{DRAFT} **Recovery Action for Failures in Execution Management or State Management** ⌈If the Global Supervision Status corresponding to State Management or Execution Management switches to GLOBAL_STATUS_STOPPED, Platform Health Management shall stop calling WatchdogInterface::AliveNotification and call WatchdogInterface::FireWatchdogReaction.⌋(*RS_HM_09249, RS_HM_09226*)

7.6.3 Configuration Parameters

Configuration of recovery actions within `Platform Health Management` has one parameter:

1. `recoveryNotificationTimeout`: the maximum acceptable amount of time `Platform Health Management` waits for an acknowledgment by State Management after sending the notification.

7.7 Multiple processes and multiple instances

During the application deployment phase, a single Supervised Entity or a single Health Channel may be instanciated several times: this happens for example when the same C++ object class representing a Supervised Entity or a Health Channel is explicitly instanciated inside the code or when the same executable containing the Supervised Entity or the Health Channel is started/run multiple times. In such a case, each instance of the Supervised Entity is individually supervised, each of them generating an instance of the Local Supervision Status.

In order to identify the relevant instance of the Supervised Entity or Health Channel, the reference to meta-model name of the specific instance is used. This can be done by a call to the function

```
ara::core::InstanceSpecifier (StringView metaModelIdentifier);
```

that is defined in the Autosar Adaptive Platform Core specification [8].

The parameter metaModelIdentifier is actually a meta-model related string in the syntax <shortname context1>/<shortname context2>/.../<shortname contextN>/<shortname of PortPrototype>, where the shortname of the Port Prototype corresponds to the Port representing the Supervised Entity or Health Channel in the meta-model.

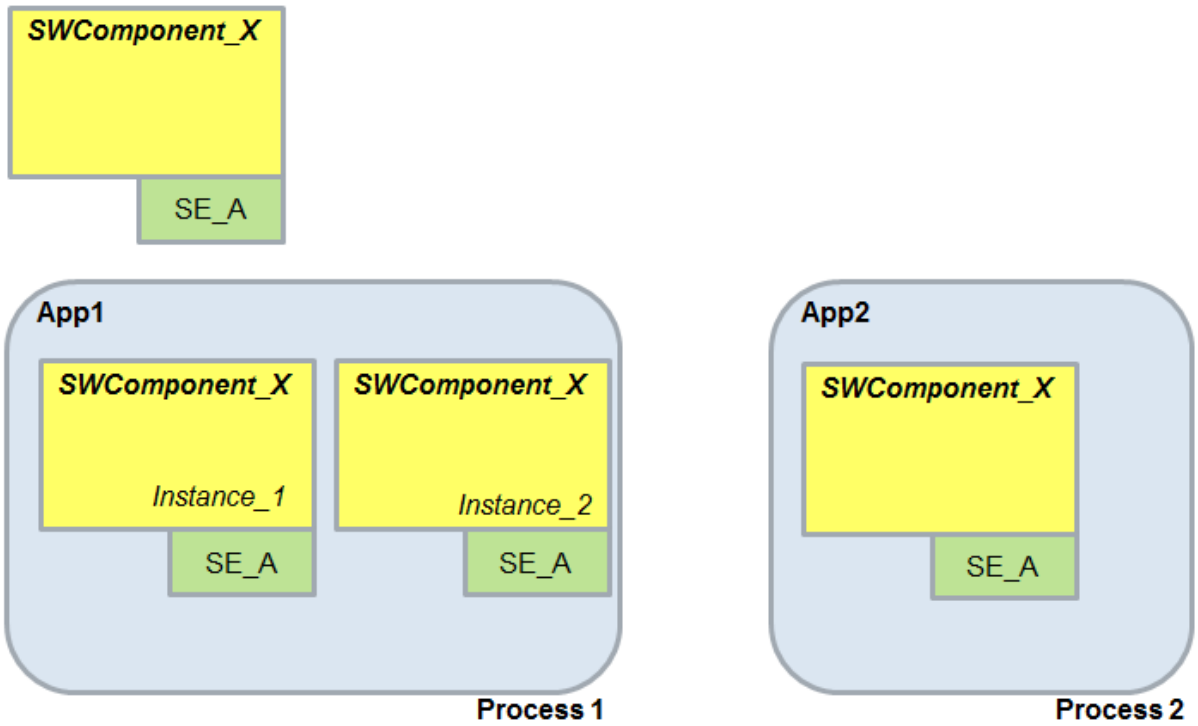

Figure 7.6: Example of multiple instance of the same Supervised Entity

Figure 7.6 shows an example of a single `Supervised Entity` (called `SE_A`) belonging to a unique SW Component (`SWCompontent_X` in the example). `SWComponent_X` is instanciated explicitly twice in the same process (`Process 1`) and another time in a different process/application (`process 2`). In such a case, three instances of the Port Prototype representing the `Supervised Entity` are created.

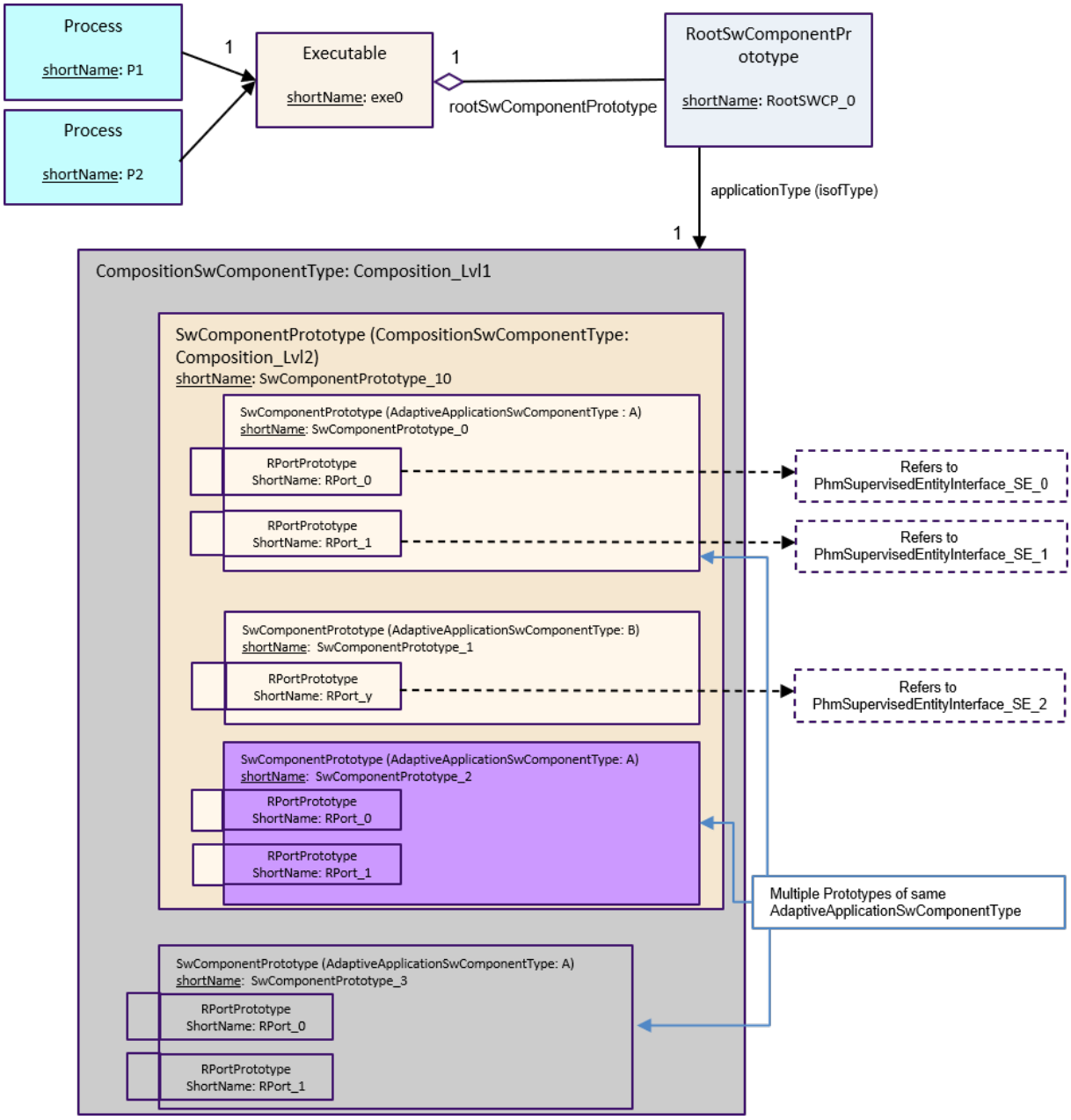

Figure 7.7: Example for Instance Specifier and context

Figure 7.7 shows a more complex example to demonstrate multiple levels of context. In the figure, all RPortPrototypes shown are referring to PHM `Supervised Entity` interface. `<shortname context1>` is the short name of the executable. The other contexts, i.e. `<shortname context2>` to `<shortname contextN>`, represent the different levels of composition. See [8] for details on the syntax of Instance Specifier.

Using the naming and composition presented in figure 7.7, the following strings are possible instance specifiers:

- exe0/RootSWCP_0/SwComponentPrototype_10/SwComponentPrototype_0/RPort_0
- exe0/RootSWCP_0/SwComponentPrototype_10/SwComponentPrototype_0/RPort_1
- exe0/RootSWCP_0/SwComponentPrototype_10/SwComponentPrototype_1/Rport_y
- exe0/RootSWCP_0/SwComponentPrototype_10/SwComponentPrototype_2/RPort_0
- exe0/RootSWCP_0/SwComponentPrototype_10/SwComponentPrototype_2/RPort_1
- exe0/RootSWCP_0/SwComponentPrototype_3/RPort_0
- exe0/RootSWCP_0/SwComponentPrototype_3/RPort_1

7.8 Functional cluster life-cycle

7.8.1 Startup

The startup behavior of `Platform Health Management` functional cluster will be specified in a future release.

7.8.2 Shutdown

Using `ara::core::Deinitialize`, all functional clusters with direct ARA interfaces can be shut down. When `Platform Health Management` functional cluster is shut down, all configured supervisions are terminated. It is the integrators responsibility to make correct use of the shutdown mechanism. Details for ensuring safe execution are given in [11].

8 API specification

8.1 API Header files

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

The generated header files provide the generated types for `Supervised Entitys` and `Health Channels`.

8.1.1 `Supervised Entity`

For each `Supervised Entity`, a separate namespace 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`, 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. These namespaces are taken from `namespace` attribute configured under `PhmSupervisedEntityInterface`. Also, see "Namespace" under "Service Interface" chapter in [12].⌋*(RS_PHM_00002)*

So an 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 and 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 Entitys ⌈The Platform 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.

8.1.2 Health Channel

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

[SWS_PHM_01113]{OBSOLETE} **Namespace of generated header files for a Health Channel** ⌈Based on the symbol attributes of the ordered SymbolProps aggregated by PhmHealthChannelInterface, 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>

```

```
14 } // namespace <PhmHealthChannelInterface.namespace[n].symbol>
15 } // namespace <...>
16 } // namespace <PhmHealthChannelInterface.namespace[1].symbol>
17 } // namespace <PhmHealthChannelInterface.namespace[0].symbol>
18
19 } // namespace health_channels
20
21 } // namespace phm
22 } // namespace ara
```

with all namespace names converted to lower-case letters. These namespaces are taken from `namespace` attribute configured under `PhmHealthChannelInterface`. Also, see "Namespace" under "Service Interface" chapter in [12].⌋*(RS_PHM_00002)*

So an 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 and uniquely identify the `Health Channel`.

[SWS_PHM_01114]{OBSOLETE} **Folder structure for `Health Channel` 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]{OBSOLETE} **Generated header files for `Health Channels`** ⌈The `Platform 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.

8.2 API Common Data Types

This chapter describes the standardized types provided by the `ara::phm` API. The `ara::phm` API is based on the `ara::core` types defined in [8].

8.2.1 Generated Types

This chapter describes the types used by Platform Health Management which are generated dependent on the input configuration.

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.

8.2.1.1 Enumeration for Checkpoint

For each Supervised Entity, an enumeration is generated containing the corresponding Checkpoints.

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

```
enum class Checkpoints : uint32_t {
  <enumerator-list>
};
```

where <enumerator-list> are the enumerators as defined by [SWS_PHM_00425].⌋*(RS_PHM_00003, RS_PHM_00101, RS_HM_09254, RS_PHM_09241)*

[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.checkpointId

<suffix> shall be "U".

⌋*(RS_PHM_00003, RS_PHM_00101, RS_HM_09254, RS_PHM_09241)*

For example, this can generate:

```
enum class Checkpoints : uint32_t
{
        Initializing = 0U,
        StartupTest = 1U,
        InitializingFinished = 2U
};
```

[SWS_PHM_00426] Namespace for `Checkpoints` ⌈The enumeration containing `Checkpoints` specified in [SWS_PHM_00424] shall be generated in the namespace of the corresponding `PhmSupervisedEntityInterface` described in [SWS_PHM_01005].⌋ *(RS_PHM_00003, RS_PHM_00101, RS_HM_09254, RS_PHM_09241)*

8.2.1.2 Enumeration for `Health Status`

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

For each `Health Channel`, an enumeration is generated containing the corresponding `Health Statuses`.

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

```
enum class HealthStatuses : uint32_t {
        <enumerator-list>
};
```

where `<enumerator-list>` are the enumerators as defined by [SWS_PHM_01119]⌋ *(RS_PHM_00003, RS_PHM_00102, RS_PHM_09257)*

[SWS_PHM_01119]{OBSOLETE} **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_01118] as:

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

where:

`<enumeratorLiteral>` is `PhmHealthChannelStatus.shortName`

`<initializer>` is the `PhmHealthChannelStatus.statusId`

`<suffix>` shall be "U".

⌋ *(RS_PHM_00003, RS_PHM_00102, RS_PHM_09257)*

For example, this can generate:

```
enum class HealthStatuses : uint32_t
{
        Low = 0U,
        High = 1U,
        Ok = 2U,
        VeryLow = 3U,
        VeryHigh = 4U
};
```

[SWS_PHM_01129]{OBSOLETE} **Enumeration for `Health Channel`** ⌈The enumeration containing `HealthStatuses` specified in [SWS_PHM_01118] shall be generated in the namespace of the corresponding `PhmHealthChannelInterface` described in [SWS_PHM_01113]⌋*(RS_PHM_00003, RS_PHM_00102, RS_PHM_09257)*

8.2.2 Non-generated types

This section defines the types that are non-generated.

8.2.2.1 LocalSupervisionStatus

[SWS_PHM_01136]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	LocalSupervisionStatus	
Scope:	namespace ara::phm	
Underlying type:	uint32_t	
Syntax:	`enum class LocalSupervisionStatus : uint32_t {...};`	
Values:	kOK= 0	Supervision is active and no failure is present.
	kFailed= 1	A failure was detected but still within tolerance/ debouncing.
	kExpired= 2	A failure was detected and qualified.
	kDeactivated= 4	Supervision is not active.
Header file:	#include "ara/phm/supervised_entity.h"	
Description:	Enumeration of local supervision status. Scoped Enumeration of uint32_t.	

⌋*(RS_HM_09237)*

8.2.2.2 GlobalSupervisionStatus

[SWS_PHM_01137]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	GlobalSupervisionStatus	
Scope:	namespace ara::phm	
Underlying type:	uint32_t	
Syntax:	`enum class GlobalSupervisionStatus : uint32_t {...};`	
Values:	kOK= 0	At least one Local Supervision corresponding to the Global Supervision is in status kOK and none in status kFailed or kExpired.

▽

△

	kFailed= 1	At least one Local supervision corresponding to the Global Supervision is in status kFailed but none in status kExpired.
	kExpired= 2	At least one local supervision corresponding to the Global Supervision is in status kExpired but the time elapsed since reaching kExpired has not exceeded the tolerance.
	kStopped= 3	At least one local supervision corresponding to the Global Supervision is in status kExpired and the time elapsed since reaching kExpired has exceeded the tolerance.
	kDeactivated= 4	All Local Supervisions corresponding to the Global Supervision are in status kDeactivated.
Header file:	#include "ara/phm/supervised_entity.h"	
Description:	Enumeration of global supervision status. Scoped Enumeration of uint32_t.	

⌋(*RS_HM_09237*)

8.2.2.3 SupervisedEntity

[SWS_PHM_01132] ⌈

Kind:	class	
Symbol:	SupervisedEntity	
Scope:	namespace ara::phm	
Syntax:	`template <typename EnumT>` `class SupervisedEntity {...};`	
Template param:	typename EnumT	An enum type that contains a list of checkpoint identifier
Header file:	#include "ara/phm/supervised_entity.h"	
Description:	SupervisedEntity Class.	

⌋(*RS_PHM_00003*, *RS_PHM_00101*, *RS_HM_09254*, *RS_PHM_00001*, *RS_PHM_-00002*)

8.2.2.4 HealthChannel

[SWS_PHM_01122]{OBSOLETE} ⌈

Kind:	class	
Symbol:	HealthChannel	
Scope:	namespace ara::phm	
Syntax:	`template <typename EnumT>` `class HealthChannel {...};`	
Template param:	typename EnumT	An enum type that contains health status Identifier

▽

△

Header file:	#include "ara/phm/health_channel.h"
Description:	HealthChannel Class.

⌋(*RS_PHM_00003*, *RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_-00002*)

8.2.2.5 RecoveryAction

[SWS_PHM_01140]{DRAFT} ⌈

Kind:	class
Symbol:	RecoveryAction
Scope:	namespace ara::phm
Syntax:	`class RecoveryAction {...};`
Header file:	#include "ara/phm/recovery_action.h"
Description:	RecoveryAction abstract class.

⌋(*RS_PHM_00003*)

8.2.2.6 HealthChannelAction

[SWS_PHM_01139]{OBSOLETE} ⌈

Kind:	class	
Symbol:	HealthChannelAction	
Scope:	namespace ara::phm	
Syntax:	`template <typename EnumT>` `class HealthChannelAction {...};`	
Template param:	typename EnumT	An enum type that contains checkpoint identifier
Header file:	#include "ara/phm/health_channel_action.h"	
Description:	HealthChannelAction abstract class.	

⌋(*RS_PHM_00003*)

8.2.2.7 TypeOfSupervision

[SWS_PHM_01138]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	TypeOfSupervision	
Scope:	namespace ara::phm	
Underlying type:	uint32_t	
Syntax:	`enum class TypeOfSupervision : uint32_t {...};`	
Values:	AliveSupervision= 0	Supervision is of type AliveSupervision.
	DeadlineSupervision= 1	Supervision is of type DeadlineSupervision.
	LogicalSupervision= 2	Supervision is of type LogicalSupervision.
Header file:	#include "ara/phm/recovery_action.h"	
Description:	Enumeration of type of supervision. Scoped Enumeration of uint32_t.	

⌋(*RS_PHM_00003*)

8.2.2.8 Daisy Chaining Related Types (Non-generated)

`Daisy chaining` is not supported in this AUTOSAR release.

8.2.2.9 Error and Exception Types

The ara::phm API does not explicitly make use of C++ exceptions. The AUTOSAR implementer is free to provide an exception-free implementation or an implementation that uses Unchecked Exceptions. The implementer is however not allowed to define Checked Exceptions.

ara::phm API builds upon a clean separation of exception types into Unchecked Exceptions and Checked Exceptions.

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., Checked Exceptions) are not used by Health Management API.

8.2.2.10 E2E Related Data Types

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

8.3 API Reference

8.3.1 SupervisedEntity API

`SupervisedEntity` API can be used to report `Checkpoint`s or to query the status of a `SupervisedEntity`.

8.3.1.1 SupervisedEntity::SupervisedEntity

[SWS_PHM_01123] ⌈

Kind:	function	
Symbol:	SupervisedEntity(const ara::core::InstanceSpecifier &instance)	
Scope:	class ara::phm::SupervisedEntity	
Syntax:	`explicit SupervisedEntity (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier of the supervised entity.
Header file:	#include "ara/phm/supervised_entity.h"	
Description:	Creation of a SupervisedEntity.	

⌋*(RS_PHM_00101, RS_HM_09254, RS_PHM_09240, RS_PHM_00001, RS_PHM_00002)*

Note that additionally process Identification information is necessary. This has to be obtained by the constructor.

[SWS_PHM_01212] ⌈

Kind:	function
Symbol:	SupervisedEntity(const SupervisedEntity &se)
Scope:	class ara::phm::SupervisedEntity
Syntax:	`SupervisedEntity (const SupervisedEntity &se)=delete;`
Header file:	#include "ara/phm/supervised_entity.h"
Description:	The copy constructor for SupervisedEntity shall not be used.

⌋*(RS_PHM_00101, RS_HM_09254, RS_PHM_09240, RS_PHM_00001, RS_PHM_00002)*

[SWS_PHM_01214] ⌈

Kind:	function	
Symbol:	SupervisedEntity(SupervisedEntity &&se)	
Scope:	class ara::phm::SupervisedEntity	
Syntax:	`SupervisedEntity (SupervisedEntity &&se) noexcept;`	
Parameters (in):	se	The SupervisedEntity object to be moved.

▽

△

Exception Safety:	noexcept
Header file:	#include "ara/phm/supervised_entity.h"
Description:	Move constructor for SupervisedEntity.

⌋(*RS_PHM_00101*, *RS_HM_09254*, *RS_PHM_09240*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.1.2 SupervisedEntity::ReportCheckpoint

[SWS_PHM_01127] ⌈

Kind:	function	
Symbol:	ReportCheckpoint(EnumT checkpointId)	
Scope:	class ara::phm::SupervisedEntity	
Syntax:	`void ReportCheckpoint (EnumT checkpointId) noexcept;`	
Parameters (in):	checkpointId	checkpoint identifier.
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/phm/supervised_entity.h"	
Description:	Reports an occurrence of a Checkpoint.	

⌋(*RS_PHM_00101*, *RS_HM_09254*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.1.3 SupervisedEntity::~SupervisedEntity

[SWS_PHM_01211] ⌈

Kind:	function
Symbol:	~SupervisedEntity()
Scope:	class ara::phm::SupervisedEntity
Syntax:	`~SupervisedEntity () noexcept;`
Exception Safety:	noexcept
Header file:	#include "ara/phm/supervised_entity.h"
Description:	Destructor of a SupervisedEntity.

⌋(*RS_PHM_00101*, *RS_HM_09254*, *RS_PHM_09240*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.1.4 SupervisedEntity::Operator=

[SWS_PHM_01213] ⌈

Kind:	function
Symbol:	operator=(const SupervisedEntity &se)
Scope:	class ara::phm::SupervisedEntity
Syntax:	`SupervisedEntity& operator= (const SupervisedEntity &se)=delete;`
Header file:	#include "ara/phm/supervised_entity.h"
Description:	The copy assignment operator for SupervisedEntity shall not be used.

⌋*(RS_PHM_00101, RS_HM_09254, RS_PHM_09240, RS_PHM_00001, RS_PHM_00002)*

[SWS_PHM_01215] ⌈

Kind:	function	
Symbol:	operator=(SupervisedEntity &&se)	
Scope:	class ara::phm::SupervisedEntity	
Syntax:	`SupervisedEntity& operator= (SupervisedEntity &&se) noexcept;`	
Parameters (in):	se	The SupervisedEntity object to be moved.
Return value:	SupervisedEntity &	The moved SupervisedEntity object.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/supervised_entity.h"	
Description:	Move assignment operator for SupervisedEntity.	

⌋*(RS_PHM_00101, RS_HM_09254, RS_PHM_09240, RS_PHM_00001, RS_PHM_00002)*

8.3.2 HealthChannel API

8.3.2.1 HealthChannel::HealthChannel

[SWS_PHM_00457]{OBSOLETE} ⌈

Kind:	function	
Symbol:	HealthChannel(const ara::core::InstanceSpecifier &instance)	
Scope:	class ara::phm::HealthChannel	
Syntax:	`explicit HealthChannel (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier of the health channel
Header file:	#include "ara/phm/health_channel.h"	
Description:	Creation of a HealthChannel.	

⌋*(RS_PHM_00102, RS_PHM_09257, RS_PHM_00001, RS_PHM_00002)*

Note that additionally process Identification information is necessary. This has to be obtained by the constructor.

[SWS_PHM_01222]{OBSOLETE} ⌈

Kind:	function
Symbol:	HealthChannel(const HealthChannel &channel)
Scope:	class ara::phm::HealthChannel
Syntax:	`HealthChannel (const HealthChannel &channel)=delete;`
Header file:	#include "ara/phm/health_channel.h"
Description:	The copy constructor for HealthChannel shall not be used.

⌋(*RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_00002*)

[SWS_PHM_01224]{OBSOLETE} ⌈

Kind:	function	
Symbol:	HealthChannel(HealthChannel &&channel)	
Scope:	class ara::phm::HealthChannel	
Syntax:	`HealthChannel (HealthChannel &&channel) noexcept;`	
Parameters (in):	channel	The HealthChannel object to be moved.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/health_channel.h"	
Description:	Move constructor for HealthChannel.	

⌋(*RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.2.2 HealthChannel::ReportHealthStatus

[SWS_PHM_01128]{OBSOLETE} ⌈

Kind:	function	
Symbol:	ReportHealthStatus(EnumT healthStatusId)	
Scope:	class ara::phm::HealthChannel	
Syntax:	`void ReportHealthStatus (EnumT healthStatusId) noexcept;`	
Parameters (in):	healthStatusId	The identifier representing the Health Status. The mapping is implementation specific.
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/phm/health_channel.h"	
Description:	Reports a Health Status.	

⌋(*RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.2.3 HealthChannel::~HealthChannel

[SWS_PHM_01221]{OBSOLETE} ⌈

Kind:	function
Symbol:	~HealthChannel()
Scope:	class ara::phm::HealthChannel
Syntax:	`~HealthChannel () noexcept;`
Exception Safety:	noexcept
Header file:	#include "ara/phm/health_channel.h"
Description:	Destructor of a HealthChannel.

⌋(*RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.2.4 HealthChannel::Operator=

[SWS_PHM_01223]{OBSOLETE} ⌈

Kind:	function
Symbol:	operator=(const HealthChannel &channel)
Scope:	class ara::phm::HealthChannel
Syntax:	`HealthChannel& operator= (const HealthChannel &channel)=delete;`
Header file:	#include "ara/phm/health_channel.h"
Description:	The copy assignment operator for HealthChannel shall not be used.

⌋(*RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_00002*)

[SWS_PHM_01225]{OBSOLETE} ⌈

Kind:	function	
Symbol:	operator=(HealthChannel &&channel)	
Scope:	class ara::phm::HealthChannel	
Syntax:	`HealthChannel& operator= (HealthChannel &&channel) noexcept;`	
Parameters (in):	channel	The HealthChannel object to be moved.
Return value:	HealthChannel &	The moved HealthChannel object.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/health_channel.h"	
Description:	Move assignment operator for HealthChannel.	

⌋(*RS_PHM_00102*, *RS_PHM_09257*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.3 RecoveryAction API

8.3.3.1 RecoveryAction::RecoveryAction

[SWS_PHM_01141]{DRAFT} ⌈

Kind:	function	
Symbol:	RecoveryAction(const ara::core::InstanceSpecifier &instance)	
Scope:	class ara::phm::RecoveryAction	
Syntax:	`explicit RecoveryAction (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier to the PPortPrototype of a Phm RecoveryActionInterface
Header file:	#include "ara/phm/recovery_action.h"	
Description:	Creation of an RecoveryAction.	

⌋(*RS_PHM_00003*)

Note that additionally process Identification information is necessary. This has to be obtained by the constructor.

[SWS_PHM_01149]{DRAFT} ⌈

Kind:	function	
Symbol:	RecoveryAction(RecoveryAction &&ra)	
Scope:	class ara::phm::RecoveryAction	
Syntax:	`RecoveryAction (RecoveryAction &&ra) noexcept;`	
Parameters (in):	ra	The RecoveryAction object to be moved.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/recovery_action.h"	
Description:	Move constructor for RecoveryAction.	

⌋(*RS_PHM_00003*)

[SWS_PHM_01150]{DRAFT} ⌈

Kind:	function
Symbol:	RecoveryAction(const RecoveryAction &)
Scope:	class ara::phm::RecoveryAction
Syntax:	`RecoveryAction (const RecoveryAction &)=delete;`
Header file:	#include "ara/phm/recovery_action.h"
Description:	The copy constructor for RecoveryAction shall not be used.

⌋(*RS_PHM_00003*)

8.3.3.2 RecoveryAction::Operator=

[SWS_PHM_01151]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(RecoveryAction &&ra)	
Scope:	class ara::phm::RecoveryAction	
Syntax:	`RecoveryAction& operator= (RecoveryAction &&ra) &noexcept;`	
Parameters (in):	ra	The RecoveryAction object to be moved.
Return value:	RecoveryAction &	The moved RecoveryAction object.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/recovery_action.h"	
Description:	Move assignment operator for RecoveryAction.	

⌋*(RS_PHM_00003)*

[SWS_PHM_01152]{DRAFT} ⌈

Kind:	function
Symbol:	operator=(const RecoveryAction &)
Scope:	class ara::phm::RecoveryAction
Syntax:	`RecoveryAction& operator= (const RecoveryAction &)=delete;`
Header file:	#include "ara/phm/recovery_action.h"
Description:	The copy assignment operator for RecoveryAction shall not be used.

⌋*(RS_PHM_00003)*

8.3.3.3 RecoveryAction::~RecoveryAction

[SWS_PHM_01145]{DRAFT} ⌈

Kind:	function
Symbol:	~RecoveryAction()
Scope:	class ara::phm::RecoveryAction
Syntax:	`virtual ~RecoveryAction () noexcept;`
Exception Safety:	noexcept
Header file:	#include "ara/phm/recovery_action.h"
Description:	Destructor for RecoveryAction.

⌋*(RS_PHM_00003)*

8.3.3.4 RecoveryAction::RecoveryHandler

[SWS_PHM_01142]{DRAFT} ⌈

Kind:	function	
Symbol:	RecoveryHandler(const ara::exec::ExecutionErrorEvent &executionError, TypeOfSupervision supervision)	
Scope:	class ara::phm::RecoveryAction	
Syntax:	`virtual void RecoveryHandler (const ara::exec::ExecutionErrorEvent &executionError, TypeOfSupervision supervision)=0;`	
Parameters (in):	executionError	Information on detected error, shall give further information for error recovery.
	supervision	The type of local supervision which failed.
Return value:	None	
Header file:	#include "ara/phm/recovery_action.h"	
Description:	RecoveryHandler to be invoked by PHM. The handler invocation needs to be enabled before by a call of RecoveryAction::Offer.	

⌋(*RS_PHM_00003*)

8.3.3.5 RecoveryAction::Offer

[SWS_PHM_01143]{DRAFT} ⌈

Kind:	function	
Symbol:	Offer()	
Scope:	class ara::phm::RecoveryAction	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	A Result, being either empty or containing any of the errors defined below.
Errors:	ara::phm::phmErrc::kOfferFailed	Returned if service could not be offered due to failure of communication with PHM daemon
Header file:	#include "ara/phm/recovery_action.h"	
Description:	Enables potential invocations of RecoveryHandler.	

⌋(*RS_PHM_00003*)

8.3.3.6 RecoveryAction::StopOffer

[SWS_PHM_01144]{DRAFT} ⌈

Kind:	function
Symbol:	StopOffer()

▽

△

Scope:	class ara::phm::RecoveryAction
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/phm/recovery_action.h"
Description:	Disables invocations of RecoveryHandler.

⌋(*RS_PHM_00003*)

8.3.3.7 RecoveryAction::GetGlobalSupervisionStatus

[SWS_PHM_01146]{DRAFT} ⌈

Kind:	function	
Symbol:	GetGlobalSupervisionStatus()	
Scope:	class ara::phm::RecoveryAction	
Syntax:	`ara::core::Result<GlobalSupervisionStatus> GetGlobalSupervisionStatus () const;`	
Return value:	ara::core::Result< GlobalSupervision Status >	A Result containing the current Global Supervision Status. In case of an error, it contains any of the errors defined below.
Errors:	ara::phm::phmErrc::kServiceNot Available	Returned if the service is not availabe, e.g. due to broken communication with PHM daemon.
Header file:	#include "ara/phm/recovery_action.h"	
Description:	Returns the status of global supervision that the supervised entity belongs to.	

⌋(*RS_PHM_00101*, *RS_HM_09237*, *RS_PHM_00001*, *RS_PHM_00002*)

8.3.4 HealthChannelAction API

8.3.4.1 HealthChannelAction::HealthChannelAction

[SWS_PHM_01231]{OBSOLETE} ⌈

Kind:	function	
Symbol:	HealthChannelAction(const ara::core::InstanceSpecifier &instance)	
Scope:	class ara::phm::HealthChannelAction	
Syntax:	`explicit HealthChannelAction (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier to the PPortPrototype of a Phm HealthChannelActionInterface
Header file:	#include "ara/phm/health_channel_action.h"	
Description:	Creation of an HealthChannelAction.	

⌋(*RS_PHM_00003*)

Note that additionally process Identification information is necessary. This has to be obtained by the constructor.

[SWS_PHM_01233]{OBSOLETE} ⌈

Kind:	function	
Symbol:	HealthChannelAction(HealthChannelAction &&hca)	
Scope:	class ara::phm::HealthChannelAction	
Syntax:	`HealthChannelAction (HealthChannelAction &&hca) noexcept;`	
Parameters (in):	hca	The HealthChannelAction object to be moved.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/health_channel_action.h"	
Description:	Move constructor for HealthChannelAction.	

⌋(*RS_PHM_00003*)

[SWS_PHM_01234]{OBSOLETE} ⌈

Kind:	function
Symbol:	HealthChannelAction(const HealthChannelAction &)
Scope:	class ara::phm::HealthChannelAction
Syntax:	`HealthChannelAction (const HealthChannelAction &)=delete;`
Header file:	#include "ara/phm/health_channel_action.h"
Description:	The copy constructor for HealthChannelAction shall not be used.

⌋(*RS_PHM_00003*)

8.3.4.2 HealthChannelAction::Operator=

[SWS_PHM_01235]{OBSOLETE} ⌈

Kind:	function	
Symbol:	operator=(HealthChannelAction &&hca)	
Scope:	class ara::phm::HealthChannelAction	
Syntax:	`HealthChannelAction& operator= (HealthChannelAction &&hca) &noexcept;`	
Parameters (in):	hca	The HealthChannelAction object to be moved.
Return value:	HealthChannelAction &	The moved HealthChannelAction object.
Exception Safety:	noexcept	
Header file:	#include "ara/phm/health_channel_action.h"	
Description:	Move assignment operator for HealthChannelAction.	

⌋(*RS_PHM_00003*)

[SWS_PHM_01236]{OBSOLETE} ⌈

Kind:	function
Symbol:	operator=(const HealthChannelAction &)
Scope:	class ara::phm::HealthChannelAction
Syntax:	`HealthChannelAction& operator= (const HealthChannelAction &)=delete;`
Header file:	#include "ara/phm/health_channel_action.h"
Description:	The copy assignment operator for HealthChannelAction shall not be used.

⌋(*RS_PHM_00003*)

8.3.4.3 HealthChannelAction::~HealthChannelAction

[SWS_PHM_01232]{OBSOLETE} ⌈

Kind:	function
Symbol:	~HealthChannelAction()
Scope:	class ara::phm::HealthChannelAction
Syntax:	`virtual ~HealthChannelAction () noexcept;`
Exception Safety:	noexcept
Header file:	#include "ara/phm/health_channel_action.h"
Description:	Destructor for HealthChannelAction.

⌋(*RS_PHM_00003*)

8.3.4.4 HealthChannelAction::RecoveryHandler

[SWS_PHM_01237]{OBSOLETE} ⌈

Kind:	function	
Symbol:	RecoveryHandler(EnumT healthStatusId)	
Scope:	class ara::phm::HealthChannelAction	
Syntax:	`virtual void RecoveryHandler (EnumT healthStatusId)=0;`	
Parameters (in):	healthStatusId	The identifier representing the Health Status. The mapping is implementation specific.
Return value:	None	
Header file:	#include "ara/phm/health_channel_action.h"	
Description:	RecoveryHandler to be invoked by PHM. The handler invocation needs to be enabled before by a call of HealthChannelAction::Offer.	

⌋(*RS_PHM_00003*)

8.3.4.5 HealthChannelAction::Offer

[SWS_PHM_01238]{OBSOLETE} ⌈

Kind:	function	
Symbol:	Offer()	
Scope:	class ara::phm::HealthChannelAction	
Syntax:	`ara::core::Result<void> Offer ();`	
Return value:	ara::core::Result< void >	A Result, being either empty or containing any of the errors defined below.
Errors:	ara::phm::phmErrc::kOfferFailed	Returned if service could not be offered due to failure of communication with PHM daemon
Header file:	#include "ara/phm/health_channel_action.h"	
Description:	Enables potential invocations of RecoveryHandler.	

⌋(*RS_PHM_00003*)

8.3.4.6 HealthChannelAction::StopOffer

[SWS_PHM_01239]{OBSOLETE} ⌈

Kind:	function
Symbol:	StopOffer()
Scope:	class ara::phm::HealthChannelAction
Syntax:	`void StopOffer ();`
Return value:	None
Header file:	#include "ara/phm/health_channel_action.h"
Description:	Disables invocations of RecoveryHandler.

⌋(*RS_PHM_00003*)

8.3.5 Forward supervision state (daisy-chain)

This feature is not supported by this AUTOSAR release.

8.4 API Errors

The `Platform Health Management` cluster implements an error handling based on `ara::core::Result`. The errors supported by the `Platform Health Management` cluster are listed in section 8.4.1.

8.4.1 PhmErrc

[SWS_PHM_01240]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	PhmErrc	
Scope:	namespace ara::phm	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class PhmErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kServiceNotAvailable= 1	Service not available. This could be due to communication error (e.g, communication with Phm daemon is broken)
	kOfferFailed= 2	Service could not be offered due to failure of communication with Phm daemon.
Header file:	#include "ara/phm/phm_error_domain.h"	
Description:	Defines an enumeration class for the Platform Health Management error codes.	

⌋*(RS_AP_00119)*

[SWS_PHM_01241]{DRAFT} ⌈The numerical ID of the PhmErrorDomain shall be 0x8000'0000'0000'0602.⌋*(RS_AP_00119)*

9 Service Interfaces

Platform Health Management does not specify any AUTOSAR Adaptive Platform Service Interface.

A Mentioned Manifest Elements

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. For further details, please refer chapters corresponding to below mentioned tables in [12].

Chapter is generated.

Class	**AliveSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines an AliveSupervision for one checkpoint. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
aliveReference Cycle	TimeValue	0..1	attr	Time period at which the Alive Supervision mechanism compares the amount of received Alive Indications for the SupervisionCheckpoint against the expectedAlive Indications. **Tags:**atp.Status=draft
checkpoint	SupervisionCheckpoint	0..1	ref	Reference to a checkpoint in the context of Alive Supervision. **Tags:**atp.Status=draft
expectedAlive Indications	PositiveInteger	0..1	attr	Defines the amount of expected Alive Indications of the SupervisionCheckpoint within the aliveReferenceCycle. **Tags:**atp.Status=draft
failedReference Cycles Tolerance	PositiveInteger	0..1	attr	This attribute defines the acceptable amount of alive ReferenceCycles with incorrect/failed AliveSupervision. **Tags:**atp.Status=draft
maxMargin	PositiveInteger	0..1	attr	Defines the amount of Alive Indications of the Supervision Checkpoint that are acceptable to be additional to the expectedAliveIndications within the aliveReferenceCycle. **Tags:**atp.Status=draft
minMargin	PositiveInteger	0..1	attr	Defines the amount of Alive Indications of the Supervision Checkpoint that are acceptable to be missing to the expectedAliveIndications within the aliveReferenceCycle. **Tags:**atp.Status=draft

Table A.1: AliveSupervision

Class	**GlobalSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines a collection of AliveSupervisions, DeadlineSupervisions, and LogicalSupervisions in order to provide an aggregated supervision state. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**GlobalSupervision**			
alive Supervision	AliveSupervision	*	aggr	Collection of AliveSupervisions in the context of this GlobalSupervision. **Tags:**atp.Status=draft
deadline Supervision	DeadlineSupervision	*	aggr	Collection of DeadlineSupervisions in the context of this GlobalSupervision. **Tags:**atp.Status=draft
logical Supervision	LogicalSupervision	*	aggr	Collection of LogicalSupervisions in the context of this GlobalSupervision. **Tags:**atp.Status=draft
noSupervision	NoSupervision	*	aggr	Collection of NoSupervisions in the context of this Global Supervision. **Tags:**atp.Status=draft
supervision Mode	SupervisionMode	*	aggr	Collection of SupervisionModes in the context of this GlobalSupervision. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=supervisionMode.shortName atp.Status=draft
transition	CheckpointTransition	*	aggr	Collection of CheckpointTransitions in the context of this GlobalSupervision. **Tags:**atp.Status=draft

Table A.2: GlobalSupervision

Class	***HealthChannel*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines the source of a health channel. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	HealthChannelExternalStatus, HealthChannelSupervision			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
recovery Notification	RecoveryNotification	0..1	aggr	Defines the RecoveryNotification. **Tags:**atp.Status=draft

Table A.3: HealthChannel

Class	**HealthChannelExternalStatus**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines a health channel representing the status of an external health channel. **Tags:**atp.Status=draft			
Base	*ARObject*, *HealthChannel*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
healthChannel	PhmHealthChannel Interface	0..1	iref	Refers to the HealthChannel. **Tags:**atp.Status=draft **InstanceRef implemented by:**PhmHealthChannelIn ExecutableInstanceRef

▽

△

Class	**HealthChannelExternalStatus**			
notifiedStatus	HealthChannelExternal ReportedStatus	*	aggr	This is a list of statuses which shall trigger the Recovery Notification of this HealthChannelExternalStatus. **Tags:**atp.Status=draft
process	Process	0..1	ref	Defines the Process this Health Channel shall be monitored. **Tags:**atp.Status=draft

Table A.4: HealthChannelExternalStatus

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***	***Mult.***	***Kind***	***Note***
symbol	CIdentifier	0..1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table A.5: ImplementationProps

Class	**NoSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines explicitly that NO supervision shall be applied for a specific Supervised Entity instance. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
process	Process	0..1	ref	Reference to the Process this NoSupervision applies to. **Tags:**atp.Status=draft
targetPhm Supervised Entity	RPortPrototype	0..1	iref	Instance reference to the RPortPrototype which represents the Supervised Entity instance. **Tags:**atp.Status=draft **InstanceRef implemented by:**RPortPrototypeIn ExecutableInstanceRef

Table A.6: NoSupervision

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***	***Mult.***	***Kind***	***Note***

▽

△

Class	PhmCheckpoint			
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 A.7: 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=PlatformHealthManagementInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *Port Interface*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
status	PhmHealthChannel Status	*	aggr	Defines the possible set of status information available to the health channel. **Tags:**atp.Status=draft

Table A.8: 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**	**Mult.**	**Kind**	**Note**
statusId	PositiveInteger	0..1	attr	Defines the numeric value which is used to indicate the indication of this status the Phm. **Tags:**atp.Status=draft
triggers Recovery Notification	Boolean	0..1	attr	Defines whether this PhmHealthChannelStatus shall cause the Phm to trigger the Health Channel recovery notification. True: Indicates unhealthy state. Phm to trigger the Health Channel recovery notification when the Health channel status changes to this state. False: Indicates healthy state. Phm not to trigger the Health Channel recovery notification when the Health channel status changes to this state. **Tags:**atp.Status=draft

Table A.9: PhmHealthChannelStatus

Class	PhmSupervisedEntityInterface
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface

▽

△

Class	**PhmSupervisedEntityInterface**			
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=PlatformHealthManagementInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *Port Interface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
checkpoint	PhmCheckpoint	*	aggr	Defines the set of checkpoints which can be reported on this supervised entity. **Tags:**atp.Status=draft

Table A.10: PhmSupervisedEntityInterface

Class	***PortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractRawDataStreamInterface*, *AbstractSynchronizedTimeBaseInterface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, LogAndTraceInterface, ModeSwitchInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, SecurityEventReportInterface, Service Interface, TriggerInterface			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName atp.Status=draft

Table A.11: PortInterface

Class	**ProcessExecutionError**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class has the ability to describe the value of a execution error along with a documentation of its semantics. **Tags:** atp.Status=draft atp.recommendedPackage=ProcessExecutionErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
executionError	PositiveInteger	0..1	attr	This attribute defines the numeric value which Execution Management and Platform Health Management reports to State Management if the Process terminates unexpectedly or violates its supervision. It shall give further error information for error recovery. **Tags:**atp.Status=draft

Table A.12: ProcessExecutionError

Class	**RecoveryNotification**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This meta-class represents a PHM action that can trigger a recovery operation inside a piece of State Management software. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
recovery Notification Timeout	TimeValue	0..1	attr	The maximum acceptable amount of time (in seconds), Platform Health Management waits for an acknowledgement by State Management after sending the notification. **Tags:**atp.Status=draft

Table A.13: RecoveryNotification

Class	**RecoveryNotificationToPPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This meta-class represents the ability to associate a RecoveryNotification to a PPortPrototype while also being able to identify the respective Process in which the actual recovery executes. **Tags:** atp.Status=draft atp.recommendedPackage=RecoveryNotificationMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
process	Process	0..1	ref	Reference to the process which represents the State Management instance that the recovery notification shall be applied to. **Tags:**atp.Status=draft
recoveryAction	PPortPrototype	0..1	iref	This reference identifies the PortPrototype to be addressed as part of a PHM recovery. **Tags:**atp.Status=draft **InstanceRef implemented by:**PPortPrototypeIn ExecutableInstanceRef
recovery Notification	RecoveryNotification	0..1	ref	This reference identifies the applicable Recovery Notification to be mapped. **Tags:**atp.Status=draft

Table A.14: RecoveryNotificationToPPortPrototypeMapping

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*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent

▽

△

Class	***Referrable*** (abstract)			
Attribute	***Type***	***Mult.***	***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. **Stereotypes:** atpIdentityContributor **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 A.15: Referrable

Class	**StartupConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class represents a reusable startup configuration for processes.. **Tags:** atp.Status=draft atp.recommendedPackage=StartupConfigs			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
environment Variable	TagWithOptionalValue	*	aggr	This aggregation represents the collection of environment variables that shall be added to the respective Process's environment prior to launch. **Tags:**atp.Status=draft
executionError	ProcessExecutionError	0..1	ref	this reference is used to identify the applicable execution error **Tags:**atp.Status=draft
process Argument (ordered)	ProcessArgument	*	aggr	This aggregation represents the collection of command-line arguments applicable to the enclosing StartupConfig. **Tags:**atp.Status=draft
scheduling Policy	String	0..1	attr	This attribute represents the ability to define the scheduling policy for the initial thread of the application. **Tags:**atp.Status=draft
scheduling Priority	Integer	0..1	attr	This is the scheduling priority requested by the application itself. **Tags:**atp.Status=draft
termination Behavior	TerminationBehavior Enum	0..1	attr	This attribute defines the termination behavior of the Process. **Tags:**atp.Status=draft
timeout	EnterExitTimeout	0..1	aggr	This aggregation can be used to specify the timeouts for launching and terminating the process depending on the StartupConfig. **Tags:**atp.Status=draft

Table A.16: StartupConfig

Class	**SupervisionCheckpoint**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element contains an instance reference to a RPortPrototype representing a checkpoint for Platform Health Management. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
checkpointId	PositiveInteger	0..1	attr	Defines the numeric value which is used to identify the reporting of this SupervisionCheckpoint to the Phm. **Tags:**atp.Status=draft
phmCheckpoint	PhmCheckpoint	0..1	iref	Instance reference to the PhmCheckpoint defined in the context of a PortInterface. **Tags:**atp.Status=draft **InstanceRef implemented by:**PhmCheckpointIn ExecutableInstanceRef
process	Process	0..1	ref	Reference to the Process this checkoint shall be monitored. **Tags:**atp.Status=draft

Table A.17: SupervisionCheckpoint

Class	**SupervisionMode**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines a SupervisionMode. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
active Supervision	PhmSupervision	*	ref	The reference defines which PhmSupervisions shall be active in this specific SupervisionMode. **Tags:**atp.Status=draft
expired Supervision Tolerance	TimeValue	0..1	attr	Defines in this SupervisionMode the acceptable amount of time with EXPIRED supervision status of the enclosing GlobalSupervision before it is considered STOPPED. **Tags:**atp.Status=draft
modeCondition	SupervisionMode Condition	0..1	ref	Reference to SupervisionModeCondition (Condition under which the configuration made under this SupervisionMode are to be applied). **Tags:**atp.Status=draft

Table A.18: SupervisionMode

Class	***SwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	AdaptiveApplicationSwComponentType, *AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***SwComponentType*** (abstract)			
port	PortPrototype	*	aggr	The PortPrototypes through which this SwComponent Type can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=port.shortName, port.variationPoint.short Label vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, sw ComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10

Table A.19: SwComponentType

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to contribute a part of a namespace.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.20: SymbolProps

B Interfaces to other Functional Clusters (informative)

B.1 Overview

AUTOSAR decided not to standardize interfaces which are exclusively used between Functional Clusters (on platform-level only), to allow efficient implementations, which might depend e.g. on the used Operating System.

This chapter provides informative guidelines how the interaction between Functional Clusters looks like, by clustering the relevant requirements of this document to describe Inter-Functional Cluster (IFC) interfaces. In addition, the standardized public interfaces which are accessible by user space applications (see chapters 8 and 9) can also be used for interaction between Functional Clusters.

The goal is to provide a clear understanding of Functional Cluster boundaries and interaction, without specifying syntactical details. This ensures compatibility between documents specifying different Functional Clusters and supports parallel implementation of different Functional Clusters. Details of the interfaces are up to the platform provider. Additional interfaces, parameters and return values can be added.

C Platform Extension API (normative)

The focus of the APIs in this section are for OEM-specific platform extensions. The abstraction of the interfaces is lower which could lead to a higher machine dependency.

C.1 WatchdogInterface

This chapter lists the required APIs for PHM to interact with the hardware watchdog.

C.1.1 WatchdogInterface::AliveNotification

Kind:	function
Symbol:	AliveNotification()
Syntax:	`void AliveNotification ();`
Return value:	None
Description:	Called cyclically by PHM in configurable cycle time. Note: This time might differ from the cycle time of triggering the "real" hardware watchdog. If PHM does not report aliveness in configured time, WatchdogInterface shall initiate watchdog reaction.

Table C.1: WatchdogInterface::AliveNotification

C.1.2 WatchdogInterface::FireWatchdogReaction

Kind:	function
Symbol:	FireWatchdogReaction()
Syntax:	`void FireWatchdogReaction ();`
Return value:	None
Description:	Interface to fire an error reaction of the hardware watchdog.

Table C.2: WatchdogInterface::FireWatchdogReaction

D Removed requirements

- [SWS_PHM_01116]{DRAFT}: Definition of an identifier for a `Supervised Entitys` ([*RS_PHM_00003*], [*RS_PHM_09240*], [*RS_PHM_09241*])
- [SWS_PHM_01133]{DRAFT}: Definition of an identifier for a Supervised Entity Prototype ([*RS_PHM_00003*], [*RS_PHM_09240*],[*RS_PHM_09241*])
- [SWS_PHM_01120]{DRAFT}: Definition of an identifier for a `Health Channel` ([*RS_PHM_00003*], [*RS_PHM_09240*],[*RS_PHM_09241*])
- [SWS_PHM_01121]{DRAFT}: Definition of an identifier for a Health Channel Prototype ([*RS_PHM_00003*], [*RS_PHM_09240*],[*RS_PHM_09241*])
- [SWS_PHM_00321]{DRAFT}: Underlying data types ([*RS_PHM_00101*], [*RS_PHM_00102*], [*RS_HM_09254*], [*RS_PHM_09257*])
- [SWS_PHM_01131]{DRAFT}: Identifier Class Template ([*RS_PHM_00101*], [*RS_PHM_00102*], [*RS_HM_09254*], [*RS_PHM_09257*])
- [SWS_PHM_01010]{DRAFT}: PHM Class ([*RS_PHM_00101*], [*RS_PHM_00102*], [*RS_HM_09254*], [*RS_PHM_09257*])
- [SWS_PHM_00458]{DRAFT}: Creation of PHM service interface ([*RS_PHM_00101*], [*RS_PHM_00102*], [*RS_HM_09254*], [*RS_PHM_09257*])
- [SWS_PHM_01124]{DRAFT}: Copy constructor for the use by SupervisedEntity and by HealthChannel ([*RS_PHM_00101*], [*RS_PHM_00102*], [*RS_HM_09254*], [*RS_PHM_09257*])
- [SWS_PHM_01125]{DRAFT}: The Platform Health Management shall provide a protected method ReportCheckpoint, provided by PHM ([*RS_PHM_00101*], [*RS_HM_09254*])
- [SWS_PHM_01126]{DRAFT}: The Platform Health Management shall provide a protected method ReportHealthStatus, provided by PHM ([*RS_PHM_00102*], [*RS_HM_09254*])
- [SWS_PHM_01101]{DRAFT}: Folder structure for header files ([*RS_PHM_00001*])
- [SWS_PHM_01018]{DRAFT}: Header file namespace ([*RS_PHM_00002*])
- [SWS_PHM_01013]{DRAFT}: Header file existence ([*RS_PHM_00001*])
- [SWS_PHM_01134]{DRAFT} ([*RS_PHM_00101*], [*RS_HM_09237*], [*RS_PHM_00001*], [*RS_PHM_00002*])
- [SWS_PHM_01160]{DRAFT}: Restricted access on GetLocalSupervisionsStatus ([*RS_HM_09237*])
- [SWS_PHM_01135]{DRAFT} ([*RS_PHM_00101*], [*RS_HM_09237*], [*RS_PHM_00001*], [*RS_PHM_00002*])

- [SWS_PHM_01161]{DRAFT}: Restricted access on GetGlobalSupervisionStatus (*[RS_HM_09237]*)
- [SWS_PHM_00103]{DRAFT}: Timeout Monitoring for notification to State Management (*[RS_HM_09159]*, *[RS_HM_09249]*)

E Not applicable requirements

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