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.

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] Specification of Intrusion Detection System Manager for Adaptive Platform
AUTOSAR_SWS_AdaptiveIntrusionDetectionSystemManager

[12] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[13] Explanation of Adaptive Platform Software Architecture
AUTOSAR_EXP_SWArchitecture

[14] Guidelines for using Adaptive Platform interfaces
AUTOSAR_EXP_AdaptivePlatformInterfacesGuidelines

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) is set to obsolete.
  Note: It is not intended to remove this feature from AUTOSAR Adaptive Platform overall. Rather, it is an architectural question to which Functional Cluster this feature belongs to, that is expected to be resolved for 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".
- Supervision is not supported for non-reporting processes (for information regarding what is a non-reporting process, please refer [10]). Rationale: Supervision depends on process states. Non-reporting process is not expected to report its Execution State to Execution Management. Hence, Platform Health Management cannot be informed about the necessary process states by Execution Management.
- Handling of multiple hardware watchdog instances is up to implementation and not standardized in the specification.

- State machine of `Elementary Supervision Status` is not specified for inter process supervisions (inter process `Deadline Supervision` and `Logical Supervision`) in this release.

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].

Following process state information is needed from Execution Management with respect to processes for which supervision is configured:

- process reporting Execution State `kRunning`,
- process terminated,
- process is about to be informed by Execution Management to terminate.

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.

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.

Platform Health Management provides three kinds of supervisions to monitor a Supervised Entity: Alive Supervision, Deadline Supervision and Logical Supervision. The supervision algorithms are described in [4]. Only details specific for Adaptive Platform are described in this document.

The results of the supervisions of a Supervised Entity Instance are reflected in the Elementary Supervision Status. The status of elementary 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_01341]{DRAFT} **Reporting of Supervision Checkpoint mapped to No Supervision provision** ⌈If a SupervisionCheckpoint reported to Platform Health Management via ReportCheckpoint is

- configured to (referenced in) NoCheckpointSupervision or
- the corresponding Supervised Entity instance is configured to NoSupervision

in the Supervision Mode corresponding to the Function Group State in which the process is executing, then Platform Health Management shall ignore the reporting of the SupervisionCheckpoint for evaluation of supervisions (Alive, Deadline and Logical).⌋*(RS_PHM_00101, RS_HM_09254)*

Note: The behavior in case of reported, undefined checkpoints is currently not specified. This will be specified in the next release.

[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 if the reporting process does not correspond to the reported SupervisionCheckpoint, i.e. reporting process is not the same as reported SupervisionCheckpoint.process.⌋*(RS_PHM_00101, RS_HM_09254, RS_IAM_00002, RS_IAM_00010)*

Example: Consider SupervisionCheckpoint SV_CP_A is referencing Process Proc_A through attribute SupervisionCheckpoint.process in the manifest and it is referenced in AliveSupervision through attribute AliveSupervision.checkpoint. In runtime, if a process other than Proc_A (e.g: Proc_B) reports SV_CP_A, then this reporting is not to be considered for evaluation of Alive Supervision.

If a checkpoint is reported by the "'wrong"' process, this is considered as access violation and a potential security threat.

[SWS_PHM_01339]{DRAFT} **Reporting access violation w.r.t. checkpoints to IdsM** ⌈Security event PHM_SEV_ACCESSVIOLATION_CHECKPOINT with the context data given in table 7.1 shall be reported to IdsM (see [11]) if it occurs that the reported SupervisionCheckpoint does not correspond to the process reporting it, i.e. reporting process is not the same as reported SupervisionCheckpoint.process.⌋ *(RS_IAM_00002, RS_IAM_00010, RS_Ids_00810)*

SEV component	*Description*
Name	PHM_SEV_ACCESSVIOLATION_CHECKPOINT
Description	Access violation with respect to reporting of checkpoint
SEV ID	65
Context Data	• Identity of the process which is violating the access permissions • Function Group State in which process is executing when there is this violation • Which SupervisionCheckpoint is getting reported

Table 7.1: Checkpoint Access Violation SEV

7.2.1 Start and Stop of Supervisions

[SWS_PHM_01331]{DRAFT} **Start of Alive Supervision** ⌈The Platform Health Management shall start the first aliveReferenceCycle of a configured AliveSupervision of a Supervised Entity Instance as soon as the corresponding process reports Execution State kRunning.⌋*(RS_HM_09125, RS_HM_09249)*

Rationale: Cyclic execution is expected only after process reached state kRunning. Execution Management monitors that the process reaches state kRunning within a configured timeout.

The information of process reporting Execution state kRunning is to be provided by Execution Management. through a vendor specific Inter Functional Cluster Interface.

[SWS_PHM_01332]{DRAFT} **Checkpoints corresponding to Alive Supervision before kRunning** ⌈With respect to Alive Supervision, Platform Health Management shall ignore Checkpoints reported by a Supervised Entity Instance before the corresponding process reaches state kRunning.⌋*(RS_HM_09125, RS_HM_09249)*

Implementation hint: The same time base should be used between Execution Management and Platform Health Management to synchronize the kRunning state with the start of the Alive Supervision. See [SWS_PHM_01334] for details.

Note: The start of intra-process Deadline Supervision and Logical Supervision (i.e. Logical and Deadline Supervision with all referenced SupervisionCheckpoints corresponding to a single process) does not depend on the process reporting Execution State kRunning. That is, the Deadline Supervision and Logical Supervision can start even before the process reaching state kRunning. Please refer [4] for details of Deadline Supervision and Logical Supervision.

[SWS_PHM_01333]{DRAFT} **Termination of Supervised Processes** ⌈As soon as Platform Health Management receives the information from Execution Management that a supervised process is about to be notified to terminate (by issuing SIGTERM) or the process is terminated (considering the case of process terminat-

ing abruptly, i.e. without SIGTERM issued by Execution Management), Platform Health Management shall stop all intra-process supervisions corresponding to the process (that is stop all Alive, Deadline and Logical Supervision involving SupervisionCheckpoints of the corresponding process only).⌋*(RS_HM_09125, RS_HM_09249)*

Rationale: Process is expected to start terminating on receiving SIGTERM from Execution Management. Execution Management monitors the termination timeout once it issues SIGTERM to the process. Considering this, additional monitoring of the process by Platform Health Management via Supervisions is considered to be not necessary.

[SWS_PHM_01334]{DRAFT} **Time Source for Supervisions** ⌈All timing aspects related to Platform Health Management shall be measured in the context of the reporting process using the same time source.⌋*(RS_HM_09254, RS_HM_09249)*

To avoid effect of delays and jitter in the inter-process communication to Platform Health Management, timing aspects related to Platform Health Management (i.e. synchronization of kRunning state between Execution Management and Platform Health Management, the timestamp w.r.t reporting of checkpoints (consider Deadline Supervision)) shall be taken in the context of the reporting process using the same time source.

Implementation Hint: ara::core::SteadyClock could be used to obtain time stamp (in other words, for time keeping).

7.2.1.1 Stopping of Alive Supervision for Self Terminating Process

In case of a Self-Terminating Process, the process can intentionally terminate even without SIGTERM being issued by Execution Management. Hence, it is necessary to mark the point in time at which the process starts to (self-) terminate so that the Alive Supervision could be stopped. This is intended to be achieved by process reporting a checkpoint named as terminatingCheckpoint. Additionally, a timeout (configurable) has to be monitored by Platform Health Management to check that the process terminates within this duration since reporting of terminatingCheckpoint. This timeout check is to monitor that the process is not stuck in its execution and therefore is not terminating.

Note: Unless SIGTERM is issued to the process by Execution Management, Execution Management will not monitor for process termination timeout.

Platform Health Management is to be informed by Execution Management regarding the termination of the process.

[SWS_PHM_01335]{DRAFT} **Stopping of Alive Supervision for Self-Terminating Process** ⌈In case of Self-Terminating Process, Alive Supervision shall be stopped on reporting of terminatingCheckpoint by the process or as soon as Platform Health Management receives the information from Execution Manage-

ment that the process will be notified to terminate (by issuing SIGTERM), whichever is earlier.⌋*(RS_HM_09125, RS_HM_09249)*

[SWS_PHM_01336]{DRAFT} **Timeout monitoring for termination of Self-Terminating Process** ⌈On reporting of terminatingCheckpoint by a Self-Terminating Process, Platform Health Management shall start monitoring the timeout. That is, Platform Health Management shall monitor that the process terminates within terminatingCheckpointTimeoutUntilTermination since reporting of terminatingCheckpoint. In case the process takes longer than terminatingCheckpointTimeoutUntilTermination for termination, this shall be notified as failure to State Management.⌋*(RS_HM_09125, RS_HM_09249)*

[SWS_PHM_01337]{DRAFT} **Unintended termination of Self-Terminating Process** ⌈If an Alive Supervision is configured for a Self Terminating Process and if the process terminates without reporting terminatingCheckpoint and no SIGTERM was issued to the process by Execution Management, then Platform Health Management shall notify a failure of Alive Supervision to State Management via ara::phm::RecoveryAction::RecoveryHandler.⌋*(RS_HM_09125, RS_HM_09249)*

[SWS_PHM_01338]{DRAFT} **Avoid redundant Monitoring of Termination for Self-Terminating Process** ⌈If an Alive Supervision is configured for a Self Terminating Process and if after reporting of terminatingCheckpoint and before terminatingCheckpointTimeoutUntilTermination is elapsed Platform Health Management receives the information from Execution Management that the process will be notified to terminate via SIGTERM, then Platform Health Management shall stop monitoring the timeout.⌋*(RS_HM_09125, RS_HM_09249)*

This is because, once SIGTERM is issued by Execution Management to the process, Execution Management will monitor the process termination timeout.

7.2.2 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.

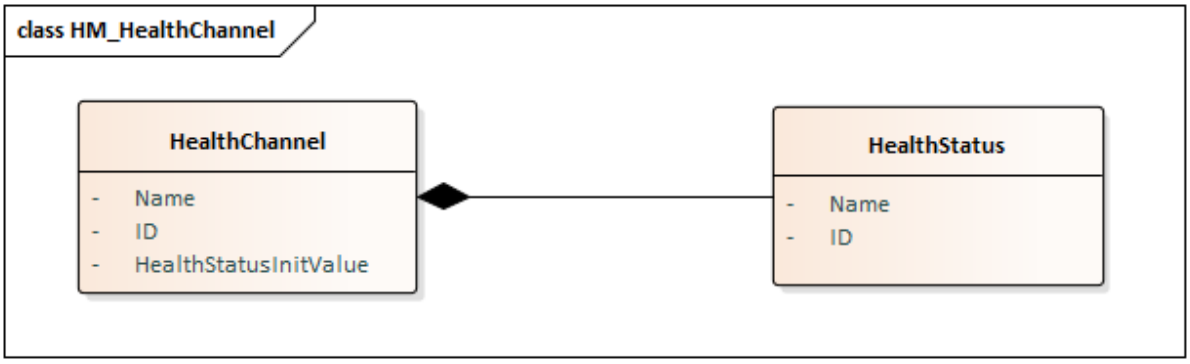

Figure 7.1: Health Channel configuration

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 notifying State Management) if the reporting process is not the same as the reported HealthChannelExternalStatus.process.⌋*(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, Elementary Supervision Status of the corresponding Supervised Entity Instance will be set to kDEACTIVATED.
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.

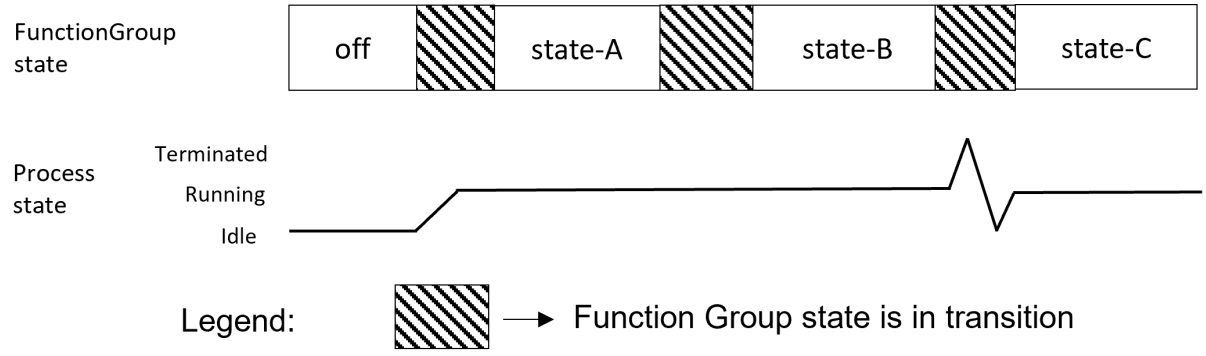

Figure 7.2: Supervision Exclusion and change of Function Group State

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 Elementary Supervision Status and Global Supervision Status are determined. Please refer [4] for details of these Supervisions.

7.5.1 Determination of Elementary Supervision Status

The Elementary Supervision Status state machine determines the status of an individual Alive Supervision, Deadline Supervision and Logical Supervision. This is done based on the following:

1. Previous value of the Elementary Supervision Status,
2. Current values of the result (correct/incorrect) of the corresponding Alive Supervision, Deadline Supervision and Logical Supervision

The state machine is initialized at the initialization of the Platform Health Management. Note: In this release, only state machine for Elementary Supervision Status for intra process supervision is specified.

[SWS_PHM_01342]{DRAFT} **Tracking of Elementary Supervision Status** ⌈The Platform Health Management shall track the Elementary Supervision Status of each Alive Supervision, Deadline Supervision and Logical Supervision.⌋*(RS_PHM_00111)*

Figure 7.3 shows the state machine for Elementary Supervision Status of a supervision with all possible states.

[SWS_PHM_01343]{DRAFT} **States of state machine for Elementary Supervision Status** ⌈The Platform Health Management shall have the Elementary Supervision Statuses kOK, kDEACTIVATED, kEXPIRED and kFAILED.⌋*(RS_PHM_00111)* See also figure 7.3 and ara::phm::ElementarySupervisionStatus.

Please note that the status kFAILED is only relevant for Alive Supervision.

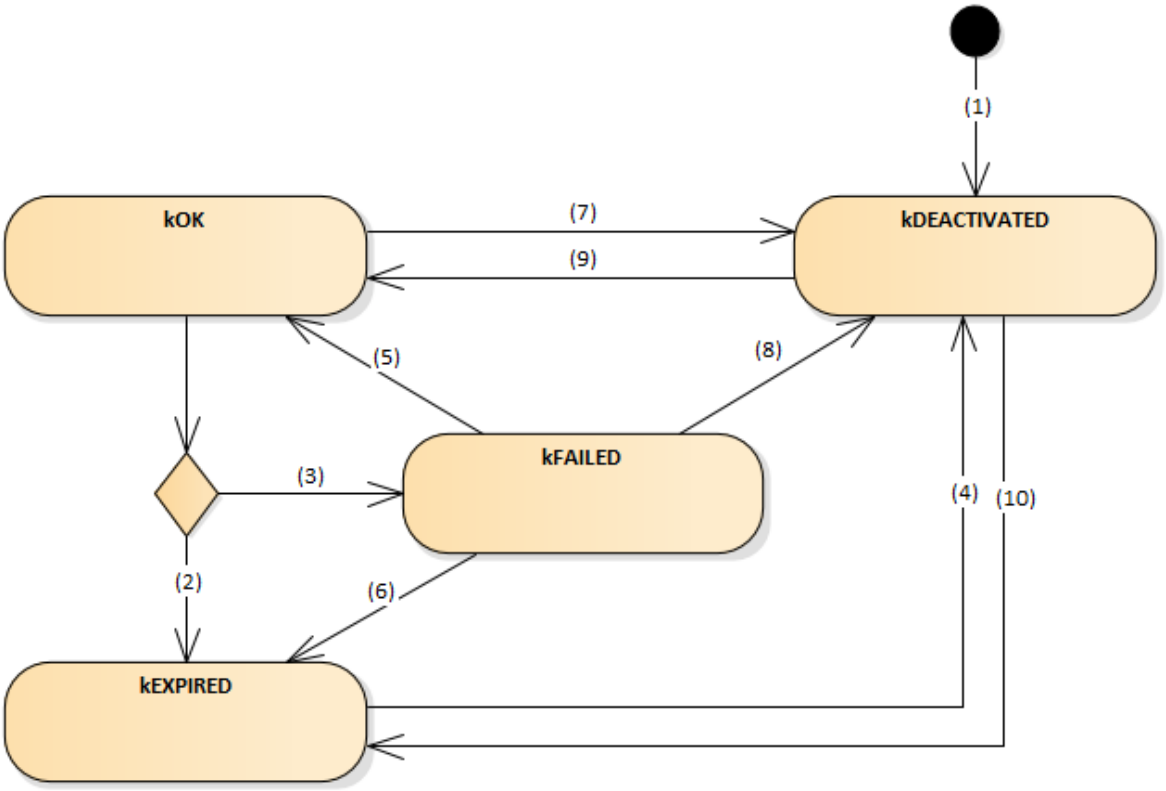

Figure 7.3: Elementary Supervision Status

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

[SWS_PHM_01344]{DRAFT} **Initialization of state machine for Elementary Supervision Status** ⌈On start of `Platform Health Management` all state machines for `Elementary Supervision Status` shall be initialized to `kDEACTIVATED` and for `Alive Supervision` 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_01345]{DRAFT} **Keep Elementary Supervision Status `kOK`** ⌈If the `Elementary Supervision Status` is `kOK` and the results of the corresponding supervision are correct, i.e. all checkpoints are reported according to configuration and in case of `Alive Supervision` the counter for failed Alive Supervision reference cycles is zero, then the `Platform Health Management` shall keep the supervision in the `Elementary Supervision Status` `kOK`.⌋*(RS_PHM_00111)*

[SWS_PHM_01346]{DRAFT} **Switch Elementary Supervision Status from `kOK` to `kEXPIRED`** ⌈If the `Elementary Supervision Status` is `kOK` AND in case the `Elementary Supervision Status` corresponds to

1. `Alive Supervision` a permanent failure is detected, i.e. the counter for failed Alive Supervision reference cycles exceeds failure tolerance `failedReferenceCyclesTolerance`) OR

2. Deadline Supervision or Logical Supervision the result of the supervision is incorrect

THEN the Platform Health Management shall change the Elementary Supervision Status to kEXPIRED and stop the corresponding supervision.⌋*(RS_PHM_00111)* See transition (2) in figure 7.3.

The below requirements show 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_01347]{DRAFT} **Switch Elementary Supervision Status from kOK to kFAILED** ⌈If Elementary Supervision Status is kOK AND the corresponding supervision is Alive Supervision AND a temporary failure is detected, i.e. the counter for failed Alive Supervision reference cycles is greater than zero but does not exceed failure tolerance failedReferenceCyclesTolerance, THEN the Platform Health Management shall change the Elementary Supervision Status to kFAILED.⌋*(RS_PHM_00111)* See transition (3) in figure 7.3.

[SWS_PHM_01348]{DRAFT} **Keep Elementary Supervision Status kFAILED** ⌈If the Elementary Supervision Status is kFAILED AND the counter for failed Alive Supervision reference cycles is greater than zero but does not exceed failure tolerance failedReferenceCyclesTolerance THEN the Platform Health Management shall keep the Elementary Supervision Status kFAILED.⌋*(RS_PHM_00111)*

[SWS_PHM_01349]{DRAFT} **Switch Elementary Supervision Status from kFAILED to kOK** ⌈If the Elementary Supervision Status is kFAILED AND there is no failure present in the Alive Supervision, i.e. the counter for failed Alive Supervision reference cycles is zero, THEN the Platform Health Management shall change the Elementary Supervision Status to kOK.⌋*(RS_PHM_00111)* See transition (5) in figure 7.3.

[SWS_PHM_01350]{DRAFT} **Switch Elementary Supervision Status from kFAILED to kEXPIRED** ⌈If the Elementary Supervision Status is kFAILED AND if the Alive Supervision has a permanent failure, i.e. the counter for failed Alive Supervision reference cycles exceeds failure tolerance failedReferenceCyclesTolerance, THEN the Platform Health Management shall change the Elementary Supervision Status to kEXPIRED and stop the corresponding supervision.⌋*(RS_PHM_00111)* See transition (6) in figure 7.3.

[SWS_PHM_01351]{DRAFT} **Switch Elementary Supervision Status from kOK to kDEACTIVATED** ⌈If the Elementary Supervision Status is kOK AND Platform Health Management receives the information from Execution Management that the corresponding process is about to be notified to terminate (by issuing SIGTERM) or the process is terminated (considering the case of process terminating abruptly, i.e. without SIGTERM issued by Execution Management), THEN the Platform Health Management shall change the Elementary Supervision Status to kDEACTIVATED

and for Alive Supervision 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_01352]{DRAFT} **Switch Elementary Supervision Status from kFAILED to kDEACTIVATED** ⌈If the Elementary Supervision Status is kFAILED AND Platform Health Management receives the information from Execution Management that the corresponding process is about to be notified to terminate (by issuing SIGTERM) or the process is terminated (considering the case of process terminating abruptly, i.e. without SIGTERM issued by Execution Management), THEN the Platform Health Management shall change the Elementary Supervision Status to kDEACTIVATED and the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111, RS_PHM_00104)* See transition (8) in figure 7.3.

[SWS_PHM_01353]{DRAFT} **Keep Elementary Supervision Status kDEACTIVATED** ⌈If the Elementary Supervision Status is kDEACTIVATED then, unless there is a switch to a Supervision Mode (due to change in corresponding Function Group State) in which the corresponding supervision is configured to be monitored AND

- for Alive Supervision: the corresponding Process reports Execution State kRunning
- for Deadline Supervision and Logical Supervision: any checkpoint corresponding to the supervision is reported

the Platform Health Management shall not perform the supervision and keep the Elementary Supervision Status kDEACTIVATED.⌋*(RS_PHM_00111, RS_PHM_00104)*

[SWS_PHM_01354]{DRAFT} **Switch Elementary Supervision Status from kDEACTIVATED to kOK** ⌈If the Elementary Supervision Status is kDEACTIVATED 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 AND

- for Alive Supervision: the corresponding Process reports Execution State kRunning
- for Deadline Supervision: when first time the checkpoint of the Supervision is reported
- for Logical Supervision: when first time the checkpoint of the Supervision is reported and the supervision result for reporting of this checkpoint is correct

THEN Platform Health Management shall change the Elementary Supervision Status to kOK.⌋*(RS_PHM_00111, RS_PHM_00104)* See transition (9) in figure 7.3.

[SWS_PHM_01355]{DRAFT} **Switch Elementary Supervision Status from kEXPIRED to kDEACTIVATED** ⌈If the Elementary Supervision Status is kEXPIRED AND the Elementary Supervision Status does not correspond to Operating System, Execution Management or State Management AND Platform Health Management receives the information from Execution Management that the corresponding process is about to be notified to terminate (by issuing SIGTERM) or the process is terminated (considering the case of process terminating abruptly, i.e. without SIGTERM issued by Execution Management), THEN the Platform Health Management shall change the Elementary Supervision Status to kDEACTIVATED and for Alive Supervision the counter for failed Alive Supervision reference cycles shall be set to zero (0).⌋*(RS_PHM_00111, RS_PHM_00104)* See transition (4) in figure 7.3.

Note: Transition (4) is not applicable in case of Elementary Supervision Status corresponding to supervision of Operating System, Execution Management or State Management reaches kEXPIRED. In this case, recovery (state change from kEXPIRED to kDEACTIVATED) is intended to be through watchdog action (see [SWS_PHM_00105]).

Note: How to determine whether a supervision corresponds to Execution Management/Operating System is not standardized. A relation to State Management can be determined via the attribute functionClusterAffiliation in the configuration of Process:
Configuration of Supervisions (AliveSupervision/ DeadlineSupervision/ LogicalSupervision) have reference to SupervisionCheckpoint which in turn refers Process in SupervisionCheckpoint.process.
This Process contains the attribute Process.functionClusterAffiliation and one of the values standardized for this attribute by AUTOSAR is "'STATE_MANAGEMENT"'. In this way it is possible to Identify which Supervisions correspond to State Management.

[SWS_PHM_01356]{DRAFT} **Keep Elementary Supervision Status kEXPIRED** ⌈If the Elementary Supervision Status is kEXPIRED then, unless Platform Health Management receives the information from Execution Management that the corresponding process is about to be notified to terminate (by issuing SIGTERM) or the process is terminated (considering the case of process terminating abruptly, i.e. without SIGTERM issued by Execution Management), the Platform Health Management shall not perform the supervision and keep the Elementary Supervision Status kEXPIRED.⌋*(RS_PHM_00111, RS_PHM_00104)*

[SWS_PHM_01357]{DRAFT} **Switch Elementary Supervision Status from kDEACTIVATED to kEXPIRED** ⌈If the Elementary Supervision Status is kDEACTIVATED and it corresponds to Logical Supervision, when first time the checkpoint of the supervision is reported and the supervision result for reporting of this checkpoint is incorrect, then Platform Health Management shall change the Elementary Supervision Status to kEXPIRED and stop the corresponding supervision.⌋*(RS_PHM_00111)* See transition (10) in figure 7.3.

Note: Transition (10) is applicable for Elementary Supervision Status of Logical Supervision only.

7.5.2 Determination of Global Supervision Status

The Global Supervision Status is determined based on the Elementary Supervision Status of a set of Alive, Deadline and/or Logical Supervisions within a Function Group which are configured as part of a single GlobalSupervision. Global Supervision Status is "worst-of" all included Elementary Supervision Statuses.

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

The Platform Health Management reports a detected failure to State Management as soon as state kEXPIRED is reached. State kSTOPPED is used only for critical failures which need a direct reaction via hardware watchdog. From AUTOSAR point of view, this is relevant for failures in supervisions corresponding to Operating System, State Management or Execution Management. Platform Health Management triggers the watchdog reaction by not setting a correct watchdog trigger condition as soon as state kSTOPPED is reached, see [SWS_PHM_00105]. This transition and therefore the reaction can be postponed for a configurable amount of time, named expiredSupervisionTolerance. This could be used to allow clean-up activities before a watchdog reset, e.g. writing the error cause, writing NVRAM data.

The expiredSupervisionTolerance is implemented within the state machine of the Global Supervision Status. The defined state machine is in the state kEXPIRED while the error reaction is postponed. Since the transition to state kSTOPPED is only applicable for supervisions triggering a watchdog reaction, the parameter expiredSupervisionTolerance is only relevant in this case. **That means, it is mandatory to configure expiredSupervisionTolerance only in case of Global Supervision corresponding to Operating System, State Management or Execution Management.** A constraint in this regard is not added in [12] as Execution Management is not a modelled process and Operating System is not represented in the model.

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

[SWS_PHM_00219]{DRAFT} **Calculation of Global Supervision Status** ⌈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 Elementary 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} **States of the state machine for Global Supervision Status** ⌈The Platform Health Management shall have the Global Supervision Statuses kOK, kDEACTIVATED, kFAILED, kEXPIRED and kSTOPPED, see ara::phm::GlobalSupervisionStatus.⌋*(RS_PHM_00111)* See also figure 7.4.

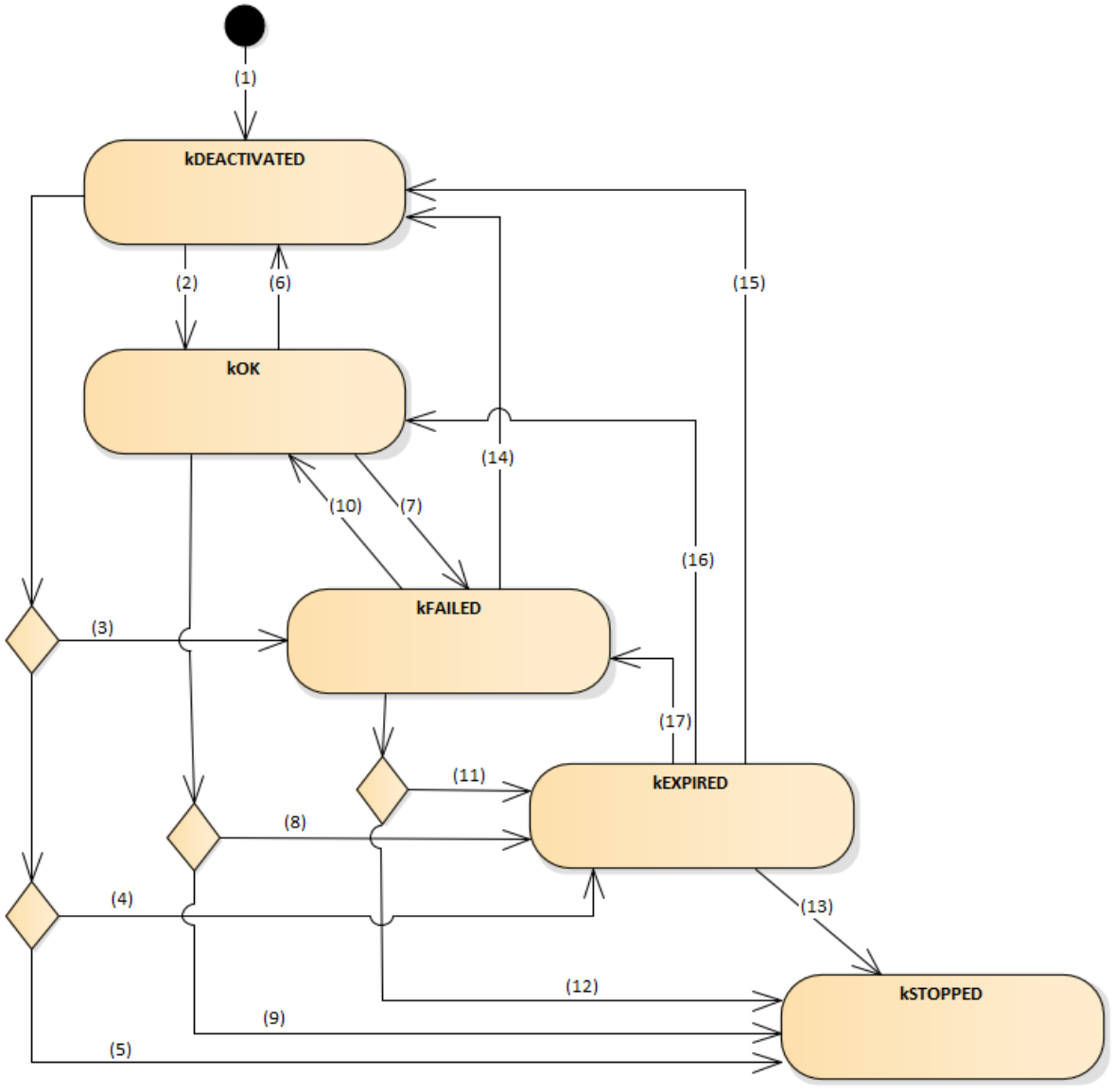

Figure 7.4: Global Supervision Status

[SWS_PHM_00217]{DRAFT} **One Global Supervision Status per Global Supervision** ⌈The Platform Health Management shall have one Global Supervision Status per GlobalSupervision configured.⌋*(RS_PHM_00111)*

Each GlobalSupervision is a set of Alive Supervision, Deadline Supervision and/or Logical Supervision corresponding to a single Function Group. There can be one or more GlobalSupervision per Function Group. But a GlobalSupervision does not span across multiple Function Groups.

[SWS_PHM_00218]{DRAFT} **Initialization of Global Supervision Status** ⌈The Global Supervision Status shall be initialized with kDEACTIVATED.⌋*(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.

[SWS_PHM_00220]{DRAFT} **Switch Global Supervision Status from kDEACTIVATED to kOK** ⌈If the Global Supervision Status is kDEACTIVATED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kOK and no supervision is in Elementary Supervision Status kFAILED or kEXPIRED, then the Platform Health Management shall change the Global Supervision Status to kOK.⌋*(RS_PHM_00111)* See transition (2) in figure 7.4.

[SWS_PHM_00221]{DRAFT} **Keep Global Supervision Status kOK** ⌈If the Global Supervision Status is kOK, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kOK and no supervision is in Elementary Supervision Status kFAILED or kEXPIRED, then the Platform Health Management shall keep the Global Supervision Status kOK.⌋*(RS_PHM_00111)*

[SWS_PHM_00222]{DRAFT} **Switch Global Supervision Status from kOK to kDEACTIVATED** ⌈If the Global Supervision Status is kOK or kFAILED or kEXPIRED AND the Elementary Supervision Status of all Alive, Deadline and Logical Supervisions is kDEACTIVATED, then the Platform Health Management shall set the Global Supervision Status to kDEACTIVATED and stop measuring Expired Supervision Time.⌋*(RS_PHM_00111)* See transitions (6), (14) and (15) in figure 7.4.

[SWS_PHM_00223]{DRAFT} **Switch Global Supervision Status from kOK to kFAILED** ⌈If the Global Supervision Status is kOK, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kFAILED and no supervision is in Elementary Supervision Status kEXPIRED, then the Platform Health Management shall change the Global Supervision Status to kFAILED.⌋*(RS_PHM_00111)* See transition (7) in figure 7.4.

[SWS_PHM_00224]{DRAFT} **Switch Global Supervision Status from kOK to kEXPIRED for SM/EM/OS supervision** ⌈If the Global Supervision Status is kOK, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED and in case the GlobalSupervision corresponds to Operating System, Execution Management or State Management the expiredSupervisionTolerance is configured to a value larger than zero, then the Platform

Health Management shall change the Global Supervision Status to kEXPIRED and start measuring Expired Supervision Time.⌋*(RS_PHM_00111, RS_PHM_00112)* See transition (8) in figure 7.4.

Note: expiredSupervisionTolerance and hence the Expired Supervision Time are applicable in case of Global Supervision Status corresponding to Operating System, Execution Management or State Management only.

[SWS_PHM_00225]{DRAFT} **Switch Global Supervision Status from kOK to kSTOPPED** ⌈If the Global Supervision Status is kOK, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED, the expiredSupervisionTolerance is configured to zero and the GlobalSupervision corresponds to Operating System, Execution Management or State Management, then the Platform Health Management shall change the Global Supervision Status to kSTOPPED.⌋*(RS_PHM_00111, RS_PHM_00112)* See transition (9) in figure 7.4.

[SWS_PHM_00226]{DRAFT} **Keep Global Supervision Status kFAILED** ⌈If the Global Supervision Status is kFAILED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kFAILED and no supervision is in Elementary Supervision Status kEXPIRED, then the Platform Health Management shall keep the Global Supervision Status kFAILED.⌋ *(RS_PHM_00111)*

[SWS_PHM_00227]{DRAFT} **Switch Global Supervision Status from kFAILED to kOK** ⌈If the Global Supervision Status is kFAILED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kOK and no supervision is in Elementary Supervision Status kFAILED or kEXPIRED, then the Platform Health Management shall change the Global Supervision Status to kOK.⌋*(RS_PHM_00111)* See transition (10) in figure 7.4.

[SWS_PHM_00228]{DRAFT} **Switch Global Supervision Status from kFAILED to kEXPIRED** ⌈If the Global Supervision Status is kFAILED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED and in case the GlobalSupervision corresponds to Operating System, Execution Management or State Management the expiredSupervisionTolerance is configured to a value larger than zero, then the Platform Health Management shall change the Global Supervision Status to kEXPIRED and start measuring Expired Supervision Time.⌋*(RS_PHM_00111, RS_PHM_00112)* See transition (11) in figure 7.4.

[SWS_PHM_00229]{DRAFT} **Switch Global Supervision Status from kFAILED to kSTOPPED** ⌈If the Global Supervision Status is kFAILED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED, the expiredSupervisionTolerance is configured to zero and the GlobalSupervision corresponds to Operating System, Execution Management or State Management, then the Platform Health Management shall change

the Global Supervision Status to kSTOPPED.⌋(*RS_PHM_00111, RS_PHM_00112*) See transition (12) in figure 7.4.

[SWS_PHM_00230]{DRAFT} **Keep Global Supervision Status kEXPIRED** ⌈If the Global Supervision Status is kEXPIRED,

- the GlobalSupervision corresponds to Operating System, Execution Management or State Management and the measured Expired Supervision Time is less than the configured expiredSupervisionTolerance OR
- the GlobalSupervision DOES NOT correspond to Operating System, Execution Management or State Management and the Elementary Supervision Status of at least one corresponding Alive, Deadline or Logical Supervision is kEXPIRED,

then the Platform Health Management shall keep the Global Supervision Status kEXPIRED.⌋(*RS_PHM_00111, RS_PHM_00112*)

[SWS_PHM_00231]{DRAFT} **Switch Global Supervision Status from kEXPIRED to kSTOPPED** ⌈If the Global Supervision Status is kEXPIRED,GlobalSupervision corresponds to Operating System, Execution Management or State Management, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED 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 kSTOPPED.⌋(*RS_PHM_00111, RS_PHM_00112*) See transition (13) in figure 7.4.

Note: Transition (13) in figure 7.4 is only applicable for GlobalSupervision that does correspond to Operating System, Execution Management or State Management.

[SWS_PHM_00232]{DRAFT} **Keep Global Supervision Status kSTOPPED** ⌈If the Global Supervision Status is kSTOPPED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED and the GlobalSupervision corresponds to Operating System, Execution Management or State Management, then the Platform Health Management shall keep the Global Supervision Status kSTOPPED.⌋(*RS_PHM_00111*)

[SWS_PHM_00233]{DRAFT} **Switch Global Supervision Status from kEXPIRED to kOK** ⌈If the Global Supervision Status is kEXPIRED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kOK and no supervision is in Elementary Supervision Status kFAILED or kEXPIRED, then the Platform Health Management shall change the Global Supervision Status to kOK.⌋(*RS_PHM_00111*) See transition (16) in figure 7.4.

[SWS_PHM_00234]{DRAFT} **Switch Global Supervision Status from kEXPIRED to kFAILED** ⌈If the Global Supervision Status is kEXPIRED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kFAILED and no supervision is in Elementary Supervision Status kEXPIRED, then the Platform Health Management shall change the Global Supervision Status to kFAILED.⌋(*RS_PHM_00111*) See transition (17) in figure 7.4.

Note: Transitions (15), (16) and (17) in figure 7.4 is not applicable in case of Global-Supervision corresponding to Operating System, Execution Management or State Management as Elementary Supervision Status of supervisions corresponding to these is not allowed to leave the state kEXPIRED until watchdog action is taken (see [SWS_PHM_00105]).

[SWS_PHM_00237]{DRAFT} **Switch Global Supervision Status from kDEACTIVATED to kFAILED** ⌈If the Global Supervision Status is kDEACTIVATED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kFAILED and no supervision is in Elementary Supervision Status kEXPIRED, then the Platform Health Management shall change the Global Supervision Status to kFAILED.⌋*(RS_PHM_00111)* See transition (3) in figure 7.4.

[SWS_PHM_00238]{DRAFT} **Switch Global Supervision Status from kDEACTIVATED to kEXPIRED** ⌈If the Global Supervision Status is kDEACTIVATED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED and in case the GlobalSupervision corresponds to Operating System, Execution Management or State Management the expiredSupervisionTolerance is configured to a value larger than zero, then the Platform Health Management shall change the Global Supervision Status to kEXPIRED and start measuring Expired Supervision Time.⌋*(RS_PHM_00111, RS_PHM_00112)* See transition (4) in figure 7.4.

[SWS_PHM_00239]{DRAFT} **Switch Global Supervision Status from kDEACTIVATED to kSTOPPED** ⌈If the Global Supervision Status is kDEACTIVATED, the Elementary Supervision Status of at least one Alive, Deadline or Logical Supervision is kEXPIRED, the expiredSupervisionTolerance is configured to zero and the GlobalSupervision corresponds to Operating System, Execution Management or State Management, then the Platform Health Management shall change the Global Supervision Status to kSTOPPED.⌋*(RS_PHM_00111, RS_PHM_00112)* See transition (5) in figure 7.4.

Note: How to distinguish whether a GlobalSupervision corresponds to Execution Management/State Management/Operating System is not standardized.

7.6 Recovery actions

The scope of Platform Health Management is to monitor the safety relevant Processes 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.

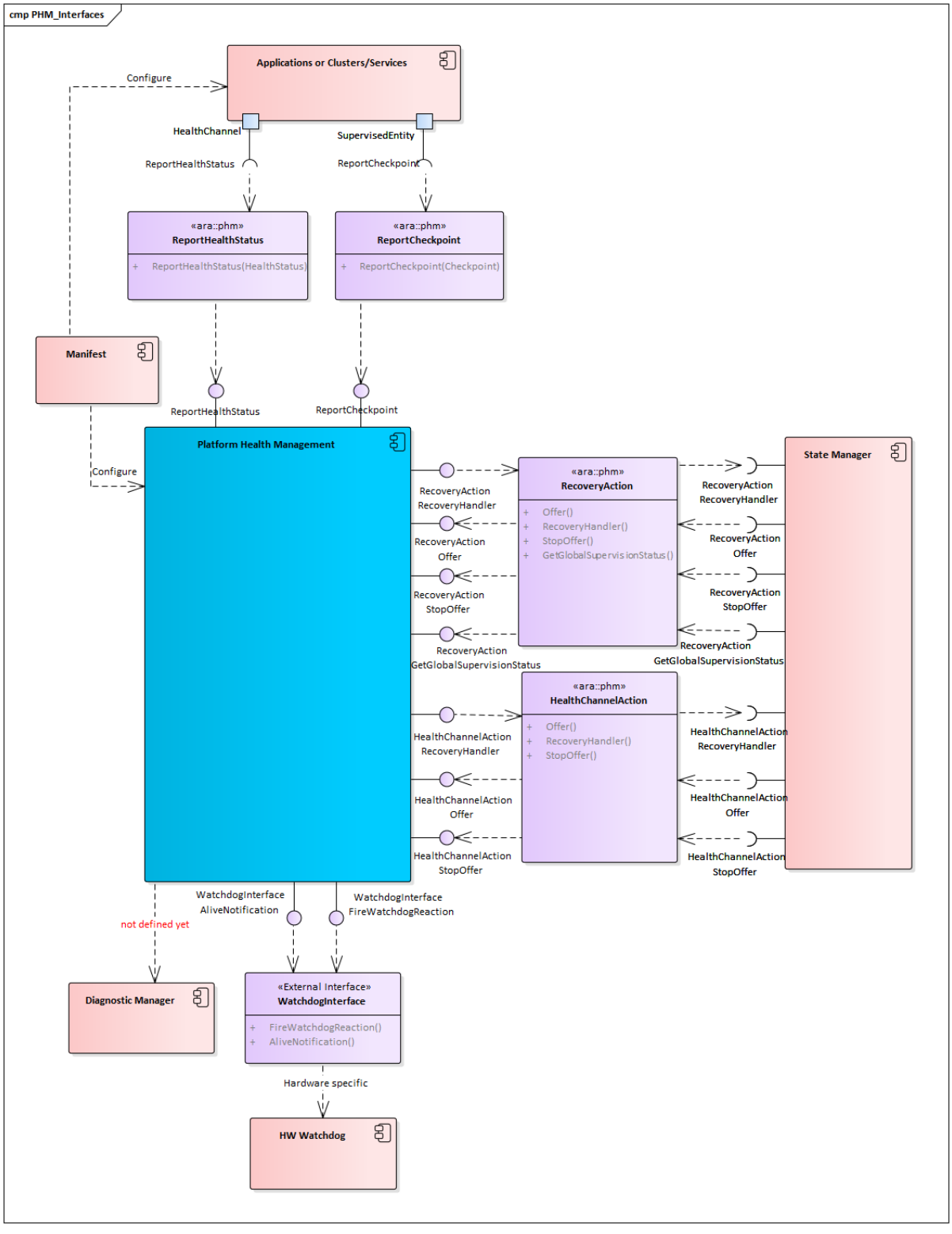

Figure 7.5: Platform Health Management and the environment

7.6.1 Notificaton to State Management

The Platform Health Management debounces the failures of Supervised Entitys, see the Elementary Supervision Status kFAILED 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 kEXPIRED, 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 kEXPIRED.⌋ *(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 kSTOPPED 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 kSTOPPED, 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 Alive Supervision, Deadline Supervision and Logical Supervision generating an instance of Elementary Supervision Status.

A specific instance of a Supervised Entity or Health Channel identifies itself at run time via an InstanceSpecifier. The API usage of the ara::core::InstanceSpecifier is specified in SWS_CORE_10200 and chapter "'InstanceSpecifier data type"' in [8]. The modelling relation of the InstanceSpecifier and its usage in PHM is explained in detail in the chapter "'Supervised Entities and Checkpoints"' in [12].

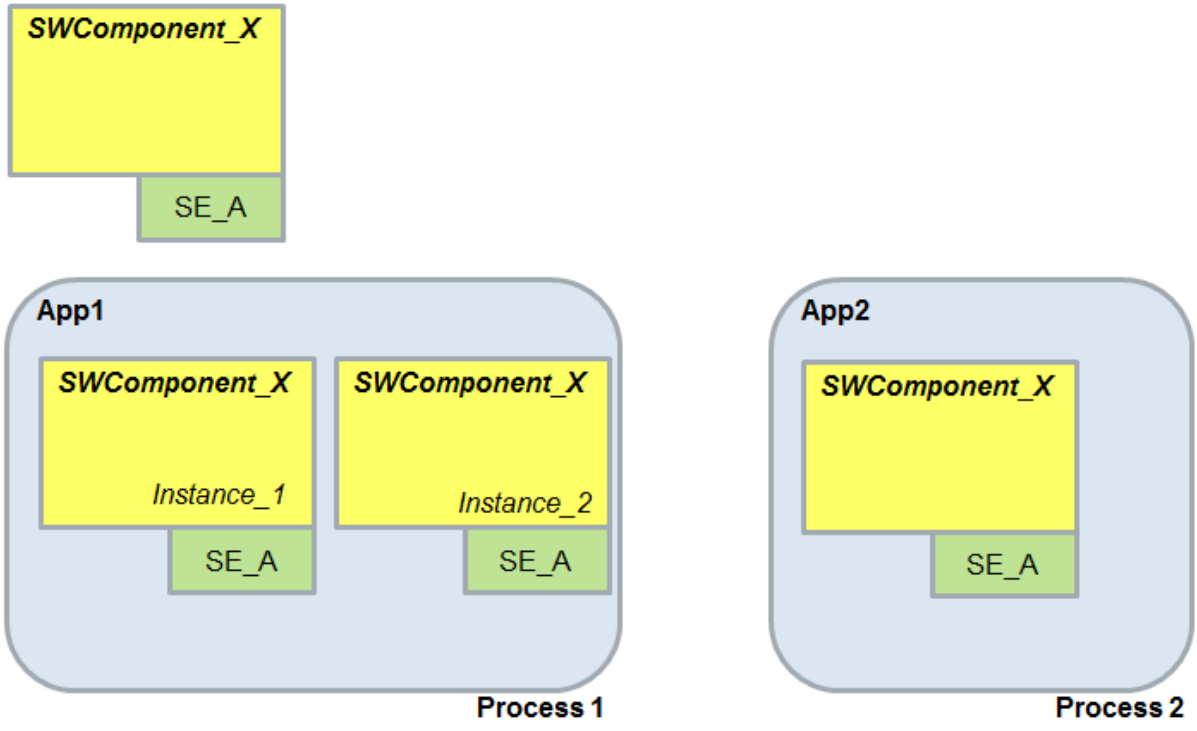

Figure 7.6: Example of multiple instance of the same Supervised Entity

Figure 7.6 shows an example of a single Supervised Entity (called SE_A) belonging to a unique SW Component (SWComponent_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.

7.8 Functional cluster life-cycle

7.8.1 Startup

[SWS_PHM_01252]{DRAFT} **Handling of Watchdog after Startup** ⌈Platform Health Management shall call WatchdogInterface::AliveNotification before reporting kRunning to Execution Management using the method ara::exec::ExecutionClient::ReportExecutionState.⌋*(RS_HM_09249, RS_HM_09244, RS_HM_09245, RS_HM_09246)*

The intention is to take over the control of the HW watchdog as early as possible.

More information on the machine startup sequence can be found in [13].

7.8.2 Shutdown

It is the integrators responsibility to make correct use of the shutdown mechanism. Details for ensuring safe execution are given in [14]. Details on the sequence of machine shutdown can be found in [13].

[SWS_PHM_01253]{DRAFT} **Termination of Supervisions at SIGTERM** ⌈Platform Health Management shall stop all configured supervisions (eg: delete all supervision objects) after receiving SIGTERM.⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_PHM_01254]{DRAFT} **Global Supervision Status at SIGTERM** ⌈Platform Health Management shall change all Global Supervision Statuses to DEACTIVATED after receiving SIGTERM.⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

7.8.2.1 Handling of watchdog during shutdown

Handling of watchdog during and after Shutdown of Platform Health Management will not be specified.

Note: Platform Health Management will no more be able to handle the servicing of the watchdog once it is shutdown.

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>

```

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

} // namespace health_channels

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

with all namespace names converted to lower-case letters. 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 : std::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 : std::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
};
```

△

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`.