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	R23-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2023-11-23	R23-11	AUTOSAR Release Management	• Addition of thread safety information to PHM APIs • Renaming of PHM security event • Added "k" prefix to enum TypeOfSupervision • Addition of explanations and examples
2022-11-24	R22-11	AUTOSAR Release Management	• Replaced Local Supervision with Elementary Supervision • Rework of state machine for Global Supervision Status • Removed API GetGlobalSupervisionStatus() from class RecoveryAction • Introduction of PhmErrorDomain functions and PhmException • Specification of Start and Stop of Supervisions

△

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

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.
Elementary Supervision Status	Status that represents the current state of an Alive Supervision, Deadline Supervision or Logical Supervision, based on the evaluation (correct/incorrect) of the supervision.
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 Elementary Supervision Status of a set of supervisions within a Function Group.
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).
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_AP_EXP_PlatformDesign

[2] Requirements on Platform Health Management
AUTOSAR_AP_RS_PlatformHealthManagement

[3] Requirements on Health Monitoring
AUTOSAR_FO_RS_HealthMonitoring

[4] Specification of Health Monitoring
AUTOSAR_FO_ASWS_HealthMonitoring

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

[6] Glossary
AUTOSAR_FO_TR_Glossary

[7] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[8] Specification of Adaptive Platform Core
AUTOSAR_AP_SWS_Core

[9] Specification of State Management
AUTOSAR_AP_SWS_StateManagement

[10] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[11] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[12] Specification of Intrusion Detection System Manager for Adaptive Platform
AUTOSAR_AP_SWS_IntrusionDetectionSystemManager

[13] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[14] Guidelines for using Adaptive Platform interfaces
AUTOSAR_AP_EXP_InterfacesGuidelines

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

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the interfaces provided by `Platform Health Management` to other Functional Clusters. Section 5.2 “Required Interfaces” lists the interfaces required by `Platform Health Management`.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [11].

5.1 Provided Interfaces

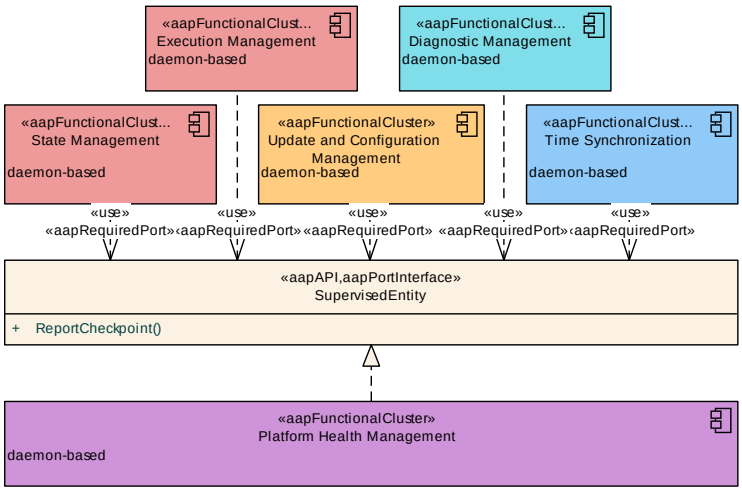


Figure 5.1: Interfaces provided by Platform Health Management to other Functional Clusters

Figure 5.1 shows interfaces provided by `Platform Health Management` to other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.1 provides a complete list of interfaces provided to other Functional Clusters within the AUTOSAR Adaptive Platform.

Interface	*Functional Cluster*	*Purpose*
`SupervisedEntity`	Diagnostic Management	`Diagnostic Management` should use this interface to enable supervision of its daemon process(es) by `Platform Health Management`.
	Execution Management	`Execution Management` shall use this interface to enable supervision of its process(es) by `Platform Health Management`.
	State Management	`State Management` shall use this interface to enable supervision of its process(es) by `Platform Health Management`.

△

Interface	*Functional Cluster*	*Purpose*
	Time Synchronization	`Time Synchronization` should use this interface to enable supervision of its daemon process by `Platform Health Management`
	Update and Configuration Management	This interface should be used to supervise the daemon process(es) of `Update and Configuration Management`.

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

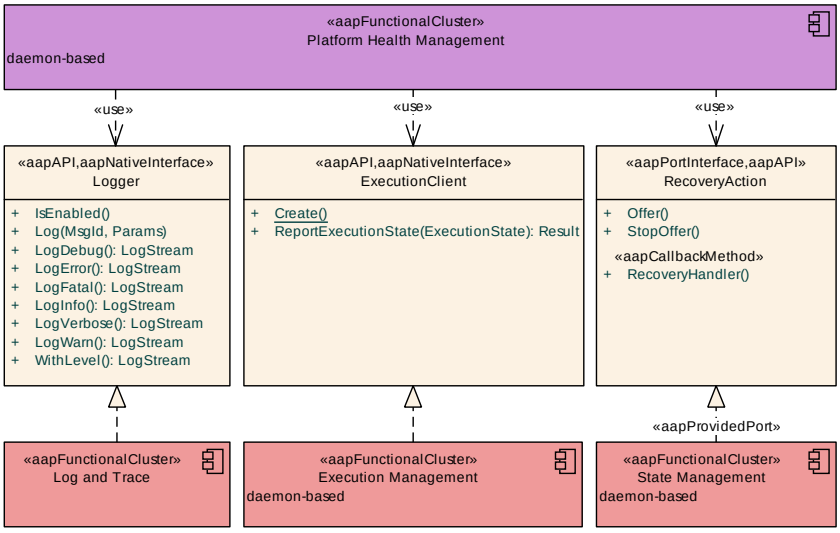


Figure 5.2: Interfaces required by Platform Health Management from other Functional Clusters

Figure 5.2 shows the interfaces required by `Platform Health Management` from other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.2 provides a complete list of required interfaces from other Functional Clusters within the AUTOSAR Adaptive Platform.

Functional Cluster	*Interface*	*Purpose*
Execution Management	ExecutionClient	`Platform Health Management` uses this interface to report the state of its daemon process to `Execution Management`.
Log and Trace	Logger	`Platform Health Management` shall use this interface to log standardized messages.
State Management	RecoveryAction	`Platform Health Management` uses this interface to trigger failure recovery.

Table 5.2: Interfaces required from other Functional Clusters

5.3 Additional dependencies on Execution Management

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

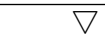
Following process state information is needed from Execution Management with respect to processes for which supervision is configured:

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

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] [SWS_PHM_01242] [SWS_PHM_01243] [SWS_PHM_01244] [SWS_PHM_01245] [SWS_PHM_01246] [SWS_PHM_01247] [SWS_PHM_01248] [SWS_PHM_01249] [SWS_PHM_01250] [SWS_PHM_01251]
[RS_AP_00127]	Usage of ara::core types.	[SWS_PHM_01245] [SWS_PHM_01246]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_PHM_01243] [SWS_PHM_01244] [SWS_PHM_01247] [SWS_PHM_01248] [SWS_PHM_01249] [SWS_PHM_01251]
[RS_HM_09125]	Health Monitoring shall provide an Alive Supervision	[SWS_PHM_01253] [SWS_PHM_01254] [SWS_PHM_01331] [SWS_PHM_01332] [SWS_PHM_01333] [SWS_PHM_01335] [SWS_PHM_01336] [SWS_PHM_01337] [SWS_PHM_01338]
[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_09222]	Health Monitoring shall provide a Logical Supervision	[SWS_PHM_01253] [SWS_PHM_01254]
[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_09235]	Health Monitoring shall provide a Deadline Supervision	[SWS_PHM_01253] [SWS_PHM_01254]
[RS_HM_09237]	Health Monitoring shall provide an interface to Supervised Entities informing them about their Supervision State.	[SWS_PHM_01137] [SWS_PHM_01358]
[RS_HM_09244]	Health Monitoring shall support timeout watchdogs.	[SWS_PHM_01252]
[RS_HM_09245]	Health Monitoring shall support window watchdogs.	[SWS_PHM_01252]
[RS_HM_09246]	Health Monitoring shall support question-answer watchdogs.	[SWS_PHM_01252]
[RS_HM_09249]	Health Monitoring shall support building safety-related systems.	[SWS_PHM_00010] [SWS_PHM_00101] [SWS_PHM_00102] [SWS_PHM_00104] [SWS_PHM_00105] [SWS_PHM_00106] [SWS_PHM_01252] [SWS_PHM_01331] [SWS_PHM_01332] [SWS_PHM_01333] [SWS_PHM_01334] [SWS_PHM_01335] [SWS_PHM_01336] [SWS_PHM_01337] [SWS_PHM_01338]
[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_01229] [SWS_PHM_01334] [SWS_PHM_01341]



△

Requirement	Description	Satisfied by
[RS_Ids_00810]	Basic SW security events	[SWS_PHM_01340]
[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_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_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_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_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01229] [SWS_PHM_01341]
[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_00240] [SWS_PHM_00241] [SWS_PHM_00242] [SWS_PHM_00243] [SWS_PHM_00244] [SWS_PHM_00245] [SWS_PHM_01351] [SWS_PHM_01352] [SWS_PHM_01353] [SWS_PHM_01354] [SWS_PHM_01355] [SWS_PHM_01356]

▽

△

Requirement	Description	Satisfied by
[RS_PHM_00111]	`Platform Health Management` shall determine Supervision status	[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_00237] [SWS_PHM_00238] [SWS_PHM_00239] [SWS_PHM_01342] [SWS_PHM_01343] [SWS_PHM_01344] [SWS_PHM_01345] [SWS_PHM_01346] [SWS_PHM_01347] [SWS_PHM_01348] [SWS_PHM_01349] [SWS_PHM_01350] [SWS_PHM_01351] [SWS_PHM_01352] [SWS_PHM_01353] [SWS_PHM_01354] [SWS_PHM_01355] [SWS_PHM_01356] [SWS_PHM_01357]
[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_00115]	If supervision of State Management fails then Platform Health Management shall trigger a watchdog reset.	[SWS_PHM_00105]
[RS_PHM_00116]	If supervision of Execution Management fails then Platform Health Management shall trigger a watchdog reset.	[SWS_PHM_00105]
[RS_PHM_00117]	Platform Health Management shall notify State Management in case an AUTOSAR Adaptive Platform functional cluster, application or service other than Execution Management and State Management fails.	[SWS_PHM_00101]
[RS_PHM_00118]	PHM shall only process a checkpoint reported from corresponding processes.	[SWS_PHM_01229]
[RS_PHM_00119]	A security event shall be raised if a checkpoint is reported from a non-corresponding process.	[SWS_PHM_01339]
[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]
[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]

▽

△

Requirement	Description	Satisfied by
[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]

Table 6.1: RequirementsTracing

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.There exists one Elementary Supervision Status per Alive, Deadline, Logical Supervision. The status of elementary supervisions within a Function Group is conglomerated in the corresponding Global Supervision Status.

One Elementary Supervision Status contributes to only one Global Supervision Status. Which Elementary Supervision Status contributes to which Global Supervision Status is determined by to which Global Supervision the corresponding supervision belongs to in the Manifest.

Scope of Global Supervision: Global Supervision corresponds to whole or part of a `Function Group`. A Global Supervision can contain all or a certain set of Elementary Supervisions corresponding to processes controlled within a single Function Group context. The mapping from Supervisions to Global Supervision is flexible. Through configuration, user can decide which Supervisions belong to which Global Supervision. But there are following restrictions:

- all Supervisions comprising a Global Supervision are corresponding to processes controlled within a single Function Group context and
- a Supervision can be part of only one Global Supervision.

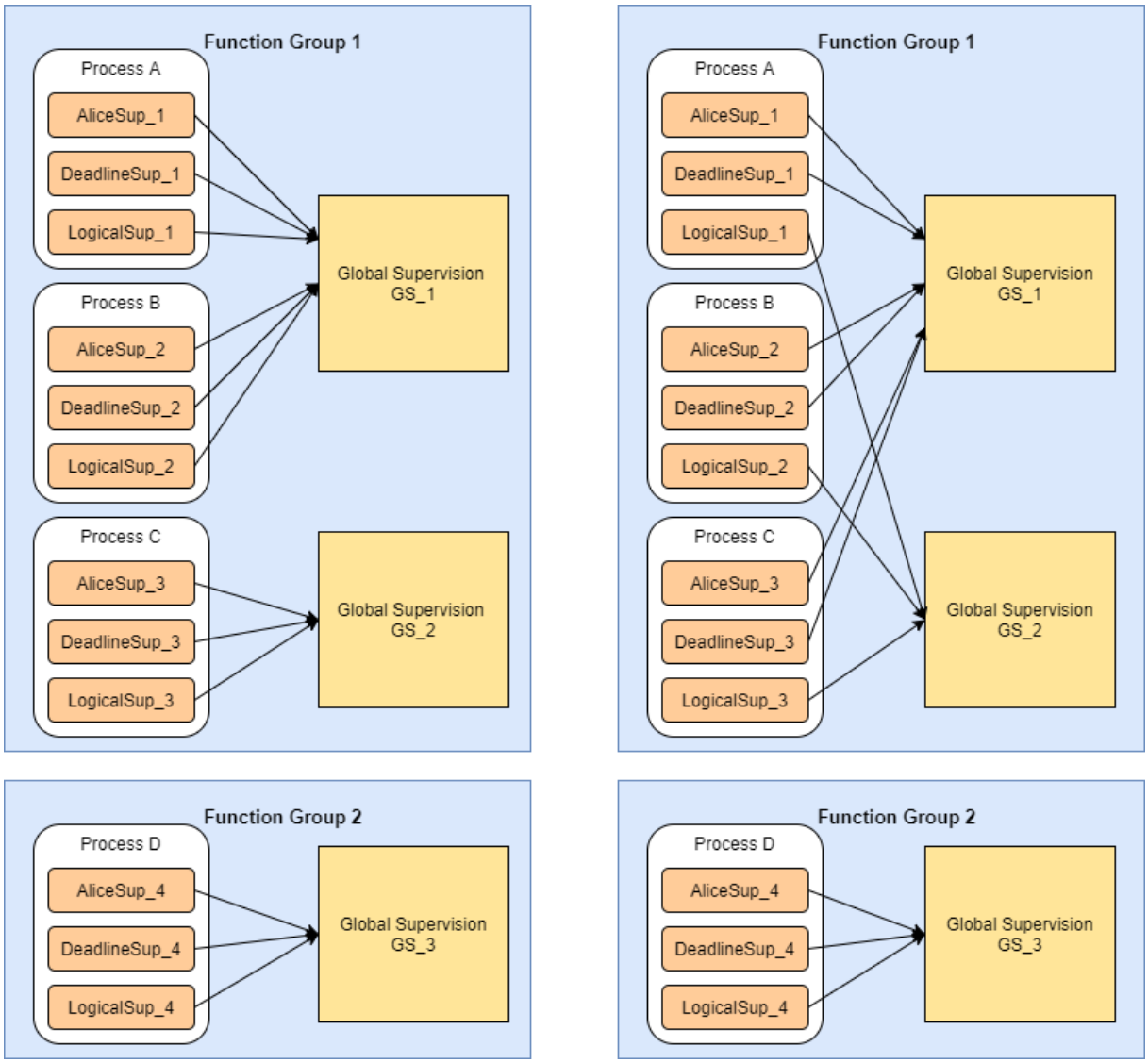


Figure 7.1: Allowed mappings of Elementary Supervisions to Global Supervisions

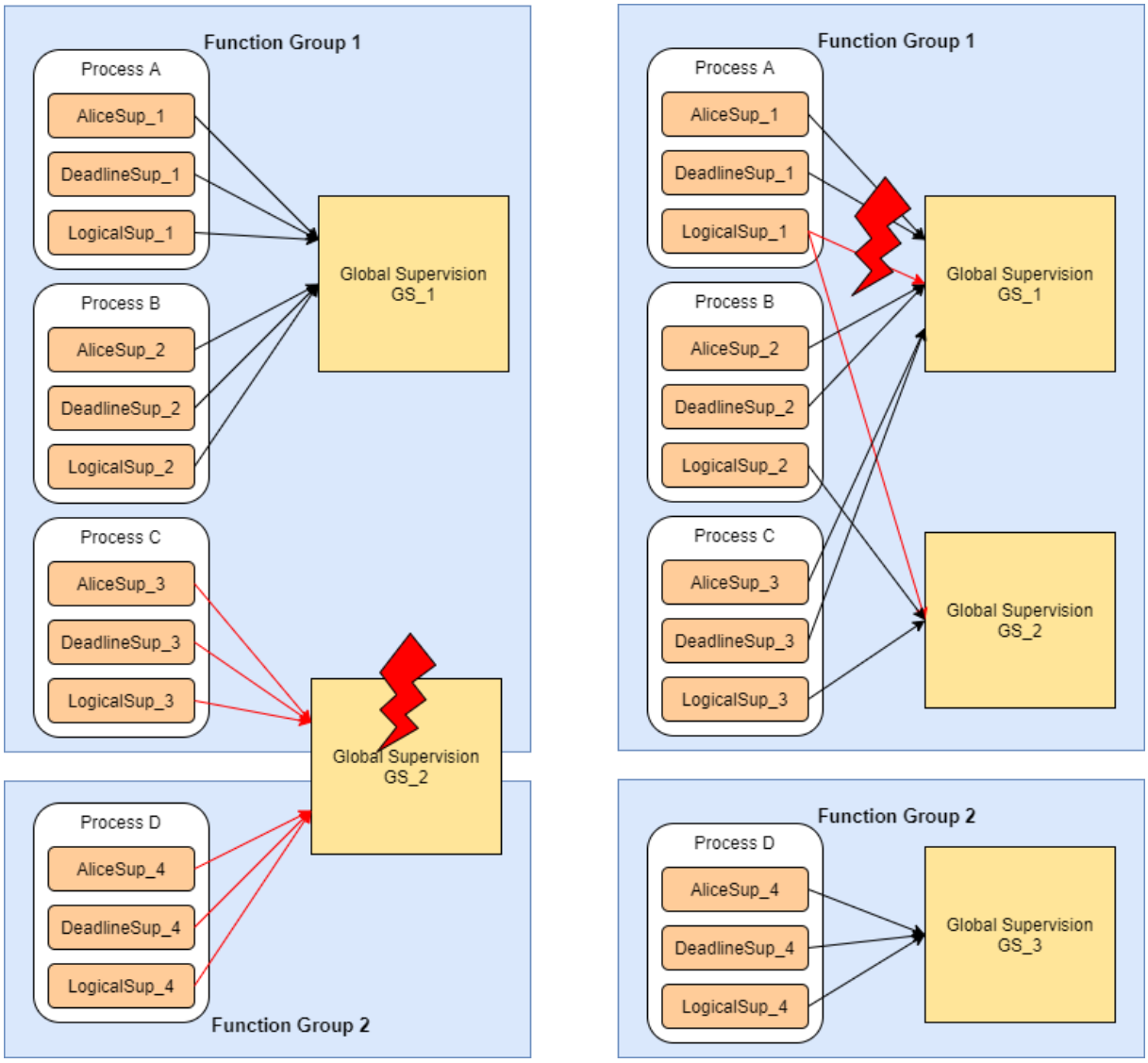


Figure 7.2: Mappings of Elementary Supervisions to Global Supervisions which are not supported

Example: Let Processes A, B and C be contained in Function Group 1 and Process D be contained in Function Group 2. Then the following mappings are allowed, see figure 7.1:

1. Supervisions corresponding to Process A and Process B comprising a Global Supervision GS_1, Supervisions corresponding to Process C comprising another Global Supervision GS_2.
2. All Supervisions corresponding to Processes in Function Group 2 are part of a single Global Supervision GS_3.
3. All Alive and Deadline Supervisions corresponding to Processes A, B and C comprise a Global Supervision GS_1, all Logical Supervisions corresponding to Processes A, B and C comprise another Global Supervision GS_2.

The following mappings are not allowed, see figure 7.2:

1. Supervisions corresponding to Processes C and D are part of a Global Supervision GS_2 since then the Global Supervision would span across multiple Function Groups.
2. Logical Supervision LogicalSup_1 corresponding to Process A is part of two Global Supervisions GS_1 and GS_2.

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_PHM_00118)*

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** ⌈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, THEN Security event SEV_ACCESSVIOLATION_PHM_CHECKPOINT defined in in [SWS_PHM_01340] shall be reported to IdsM (see [12]) with the following 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

⌋(*RS_PHM_00119*)

[SWS_PHM_01340] Security events for PHM ⌈

Name	*Description*	*ID*
SEV_ACCESSVIOLATION_PHM_CHECKPOINT	Access violation with respect to reporting of checkpoint	65

⌋(*RS_Ids_00810*)

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 terminating 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.

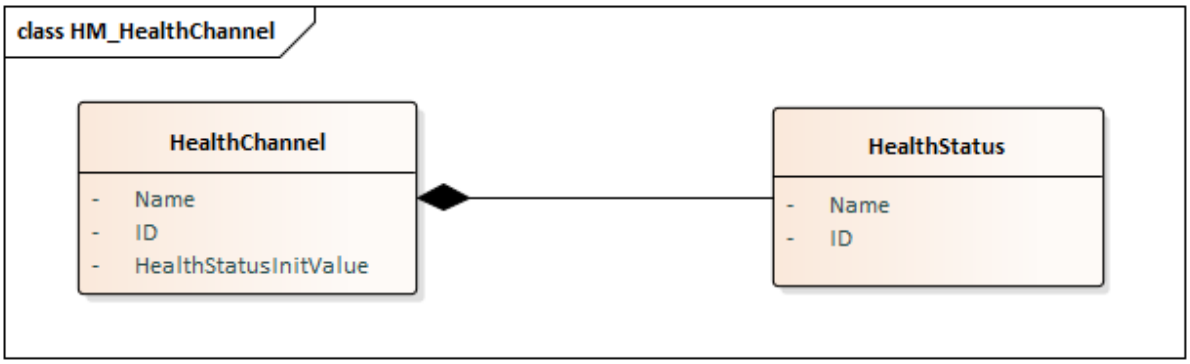


Figure 7.3: Health Channel configuration

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)*

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.

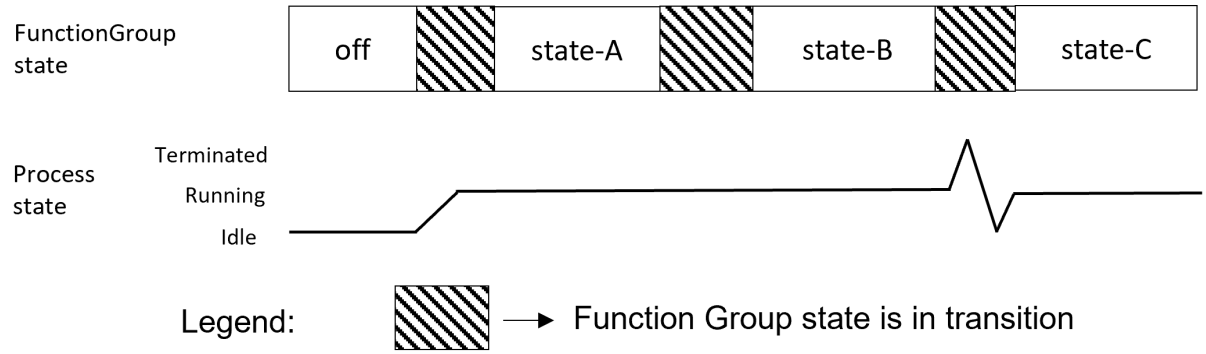


Figure 7.4: Supervision Exclusion and change of Function Group State

Figure 7.4 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 7.5 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 7.5 and ara::phm::ElementarySupervisionStatus.

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

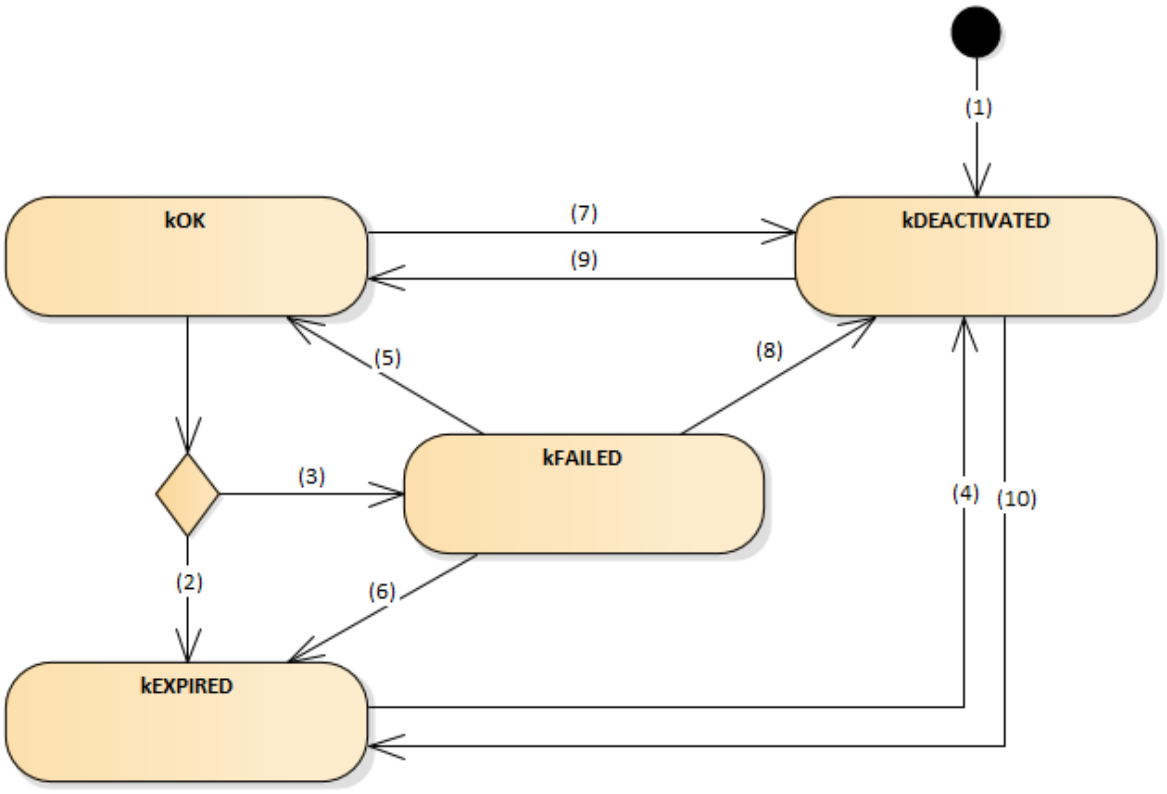


Figure 7.5: Elementary Supervision Status

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 7.5.

[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 7.5.

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 7.5.

[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 7.5.

[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 7.5.

[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 7.5.

[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 7.5.

[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 7.5.

[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 7.5.

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 7.5.

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 7.6 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 [13] 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 7.6.

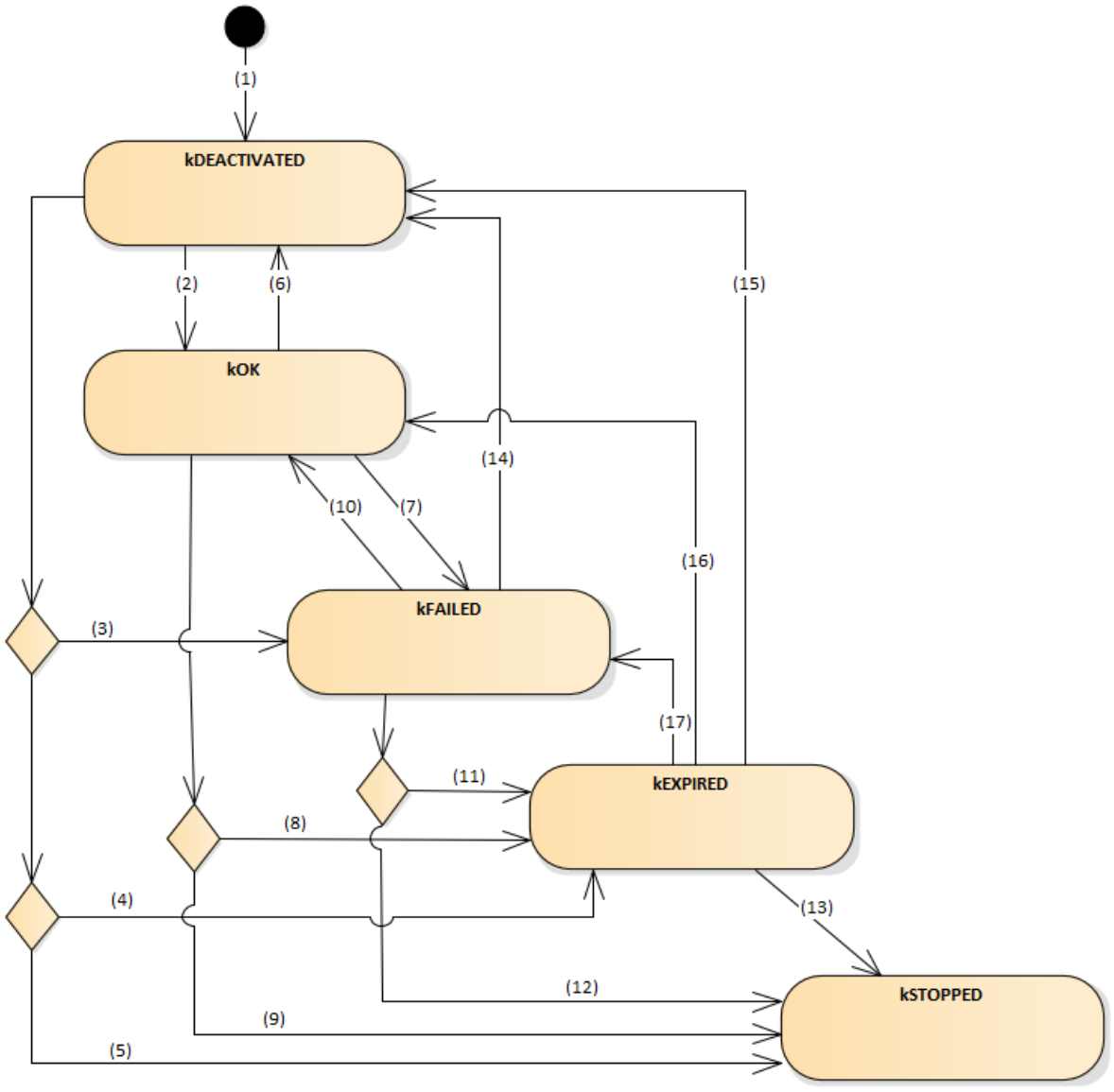


Figure 7.6: Global Supervision Status

[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 7.6.

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 7.6.

[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 7.6.

These transitions can occur when State Management has caused change in the state of the Function Group corresponding to the Global Supervision such that the Processes corresponding to the Supervised Entity instances whose Supervisions (Alive Supervisions, Deadline Supervisions and/or Logical Supervisions) are aggregated in the Global Supervision, are terminated. Typically, this can occur due to change in Function Group state to Off state.

[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 7.6.

[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 7.6.

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 7.6.

[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 7.6.

[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 7.6.

[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 7.6.

[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 7.6.

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 7.6.

This transition can occur when State Management has caused change in the state of the Function Group corresponding to the Global Supervision such that the Process corresponding to the Supervised Entity instance whose Elementary Supervision Status caused the Global Supervision Status to reach state kEXPIRED is terminated or restarted.

[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 7.6.

This transition can occur when State Management has caused change in the state of the Function Group corresponding to the Global Supervision such that the Process corresponding to the Supervised Entity instance whose Elementary Supervision Status caused the Global Supervision Status to reach state kEXPIRED is terminated or restarted. However, there exists another executing process whose corresponding Supervised Entity instance is in Elementary Supervision Status kFAILED and is not terminated or restarted.

Note: Transitions (15), (16) and (17) in figure 7.4 is not applicable in case of GlobalSupervision 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 7.6.

[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 7.6.

[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 7.6.

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.

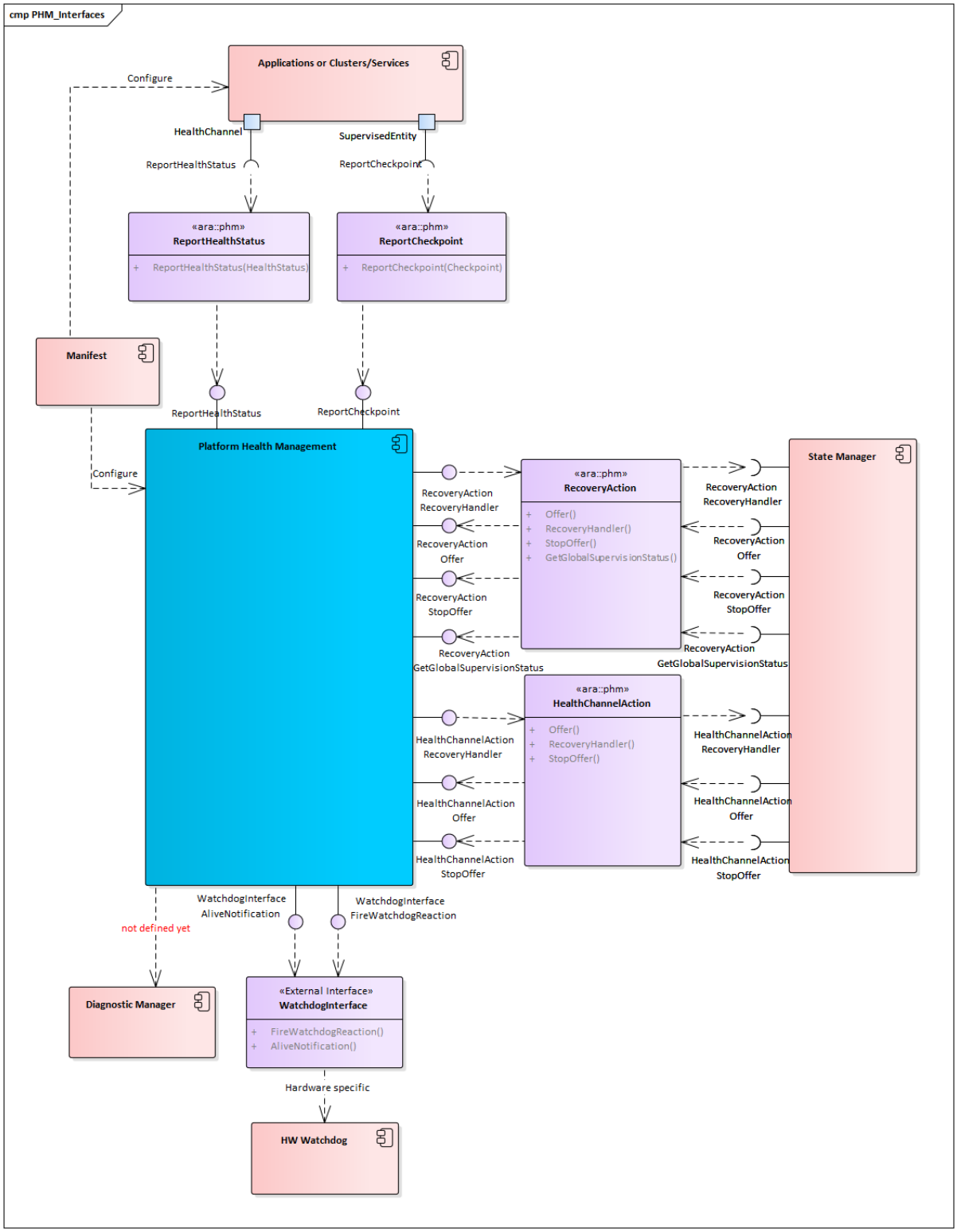


Figure 7.7: Platform Health Management and the environment

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, RS_PHM_00117)*

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*, *RS_PHM_00115*, *RS_PHM_00116*)

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 [13].

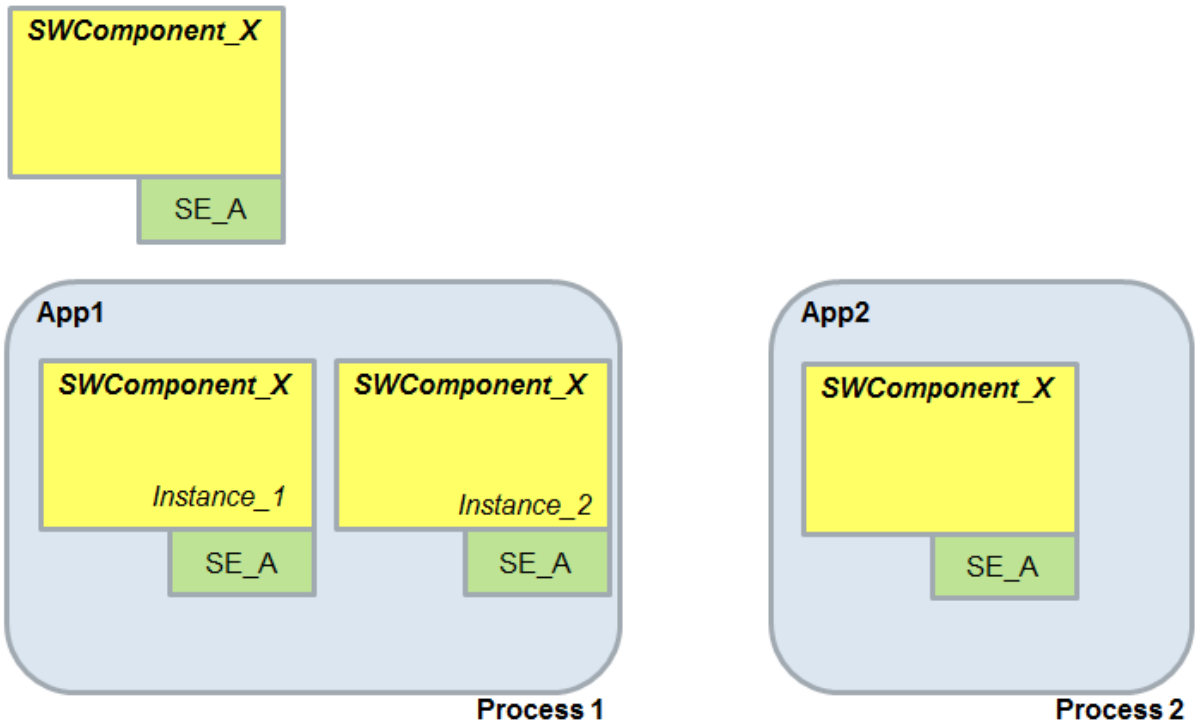


Figure 7.8: Example of multiple instance of the same Supervised Entity

Figure 7.8 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 [11].

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 [11].

[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 [13].⌋ *(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 [13].⌋*(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
};
```

[SWS_PHM_01129]{OBSOLETE} **Enumeration for `Health Channel`** ⌈The enumeration containing `Health Statuses` 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 ElementarySupervisionStatus

[SWS_PHM_01358]{DRAFT} **Definition of API enum ara::phm::ElementarySupervisionStatus** ⌈

Kind:	enumeration	
Header file:	#include "ara/phm/supervised_entity.h"	
Forwarding header file:	#include "ara/phm/phm_fwd.h"	
Scope:	namespace ara::phm	
Symbol:	ElementarySupervisionStatus	
Underlying type:	std::uint32_t	
Syntax:	`enum class ElementarySupervisionStatus : std::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.
Description:	Enumeration of elementary supervision status. Scoped Enumeration of uint32_t.	

⌋*(RS_HM_09237)*

8.2.2.2 GlobalSupervisionStatus

[SWS_PHM_01137]{DRAFT} **Definition of API enum ara::phm::GlobalSupervision Status** ⌈

Kind:	enumeration
Header file:	#include "ara/phm/supervised_entity.h"
Forwarding header file:	#include "ara/phm/phm_fwd.h"
Scope:	namespace ara::phm
Symbol:	GlobalSupervisionStatus
Underlying type:	std::uint32_t
Syntax:	`enum class GlobalSupervisionStatus : std::uint32_t {...};`

▽

△

Values:	kOK= 0	At least one Elementary Supervision corresponding to the Global Supervision is in status kOK and none in status kFailed or kExpired.
	kFailed= 1	At least one Elementary Supervision corresponding to the Global Supervision is in status kFailed but none in status kExpired.
	kExpired= 2	At least one Elementary 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 Elementary Supervision corresponding to the Global Supervision is in status kExpired and the time elapsed since reaching kExpired has exceeded the tolerance.
	kDeactivated= 4	All Elementary Supervisions corresponding to the Global Supervision are in status kDeactivated.
Description:	Enumeration of global supervision status. Scoped Enumeration of uint32_t.	

⌋(*RS_HM_09237*)

8.2.2.3 SupervisedEntity

[SWS_PHM_01132] Definition of API class ara::phm::SupervisedEntity ⌈

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

Kind:	class
Header file:	#include "ara/phm/health_channel.h"
Forwarding header file:	#include "ara/phm/phm_fwd.h"
Scope:	namespace ara::phm
Symbol:	HealthChannel
Syntax:	`template <typename EnumT>` `class HealthChannel {...};`

▽

△

Template param:	typename EnumT	An enum type that contains health status Identifier
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} **Definition of API class ara::phm::RecoveryAction** ⌈

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

⌋(*RS_PHM_00003*)

8.2.2.6 HealthChannelAction

[SWS_PHM_01139]{OBSOLETE} **Definition of API class ara::phm::HealthChannel Action** ⌈

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

⌋(*RS_PHM_00003*)

8.2.2.7 TypeOfSupervision

[SWS_PHM_01138]{DRAFT} **Definition of API enum ara::phm::TypeOfSupervision** ⌈

Kind:	enumeration	
Header file:	#include "ara/phm/recovery_action.h"	
Forwarding header file:	#include "ara/phm/phm_fwd.h"	
Scope:	namespace ara::phm	
Symbol:	TypeOfSupervision	
Underlying type:	std::uint32_t	
Syntax:	`enum class TypeOfSupervision : std::uint32_t {...};`	
Values:	kAliveSupervision= 0	Supervision is of type AliveSupervision.
	kDeadlineSupervision= 1	Supervision is of type DeadlineSupervision.
	kLogicalSupervision= 2	Supervision is of type LogicalSupervision.
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] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity ⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	class ara::phm::SupervisedEntity	
Symbol:	SupervisedEntity(const ara::core::InstanceSpecifier &instance)	
Syntax:	`explicit SupervisedEntity (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier of the supervised entity.
Exception Safety:	not exception safe	
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] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity ⌈

Kind:	function
Header file:	#include "ara/phm/supervised_entity.h"
Scope:	class ara::phm::SupervisedEntity
Symbol:	SupervisedEntity(const SupervisedEntity &se)
Syntax:	`SupervisedEntity (const SupervisedEntity &se)=delete;`
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] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity ⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	class ara::phm::SupervisedEntity	
Symbol:	SupervisedEntity(SupervisedEntity &&se)	
Syntax:	`SupervisedEntity (SupervisedEntity &&se) noexcept;`	
Parameters (in):	se	The SupervisedEntity object to be moved.
Exception Safety:	noexcept	
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] Definition of API function ara::phm::SupervisedEntity::ReportCheckpoint ⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	class ara::phm::SupervisedEntity	
Symbol:	ReportCheckpoint(EnumT checkpointId)	
Syntax:	`void ReportCheckpoint (EnumT checkpointId) noexcept;`	
Parameters (in):	checkpointId	checkpoint identifier.
Return value:	None	
Exception Safety:	noexcept	
Thread Safety:	thread-safe	
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] Definition of API function ara::phm::SupervisedEntity::~SupervisedEntity ⌈

Kind:	function
Header file:	#include "ara/phm/supervised_entity.h"
Scope:	class ara::phm::SupervisedEntity
Symbol:	~SupervisedEntity()

▽

△

Syntax:	`~SupervisedEntity () noexcept;`
Exception Safety:	noexcept
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] Definition of API function ara::phm::SupervisedEntity::operator= ⌈

Kind:	function
Header file:	#include "ara/phm/supervised_entity.h"
Scope:	class ara::phm::SupervisedEntity
Symbol:	operator=(const SupervisedEntity &se)
Syntax:	`SupervisedEntity & operator= (const SupervisedEntity &se)=delete;`
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] Definition of API function ara::phm::SupervisedEntity::operator= ⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	class ara::phm::SupervisedEntity	
Symbol:	operator=(SupervisedEntity &&se)	
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	
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} **Definition of API function ara::phm::HealthChannel::HealthChannel** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel.h"	
Scope:	class ara::phm::HealthChannel	
Symbol:	HealthChannel(const ara::core::InstanceSpecifier &instance)	
Syntax:	`explicit HealthChannel (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier of the health channel
Exception Safety:	not exception safe	
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} **Definition of API function ara::phm::HealthChannel::HealthChannel** ⌈

Kind:	function
Header file:	#include "ara/phm/health_channel.h"
Scope:	class ara::phm::HealthChannel
Symbol:	HealthChannel(const HealthChannel &channel)
Syntax:	`HealthChannel (const HealthChannel &channel)=delete;`
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} **Definition of API function ara::phm::HealthChannel::HealthChannel** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel.h"	
Scope:	class ara::phm::HealthChannel	
Symbol:	HealthChannel(HealthChannel &&channel)	
Syntax:	`HealthChannel (HealthChannel &&channel) noexcept;`	
Parameters (in):	channel	The HealthChannel object to be moved.
Exception Safety:	noexcept	
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} **Definition of API function ara::phm::HealthChannel::ReportHealthStatus** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel.h"	
Scope:	class ara::phm::HealthChannel	
Symbol:	ReportHealthStatus(EnumT healthStatusId)	
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	
Thread Safety:	thread-safe	
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} **Definition of API function ara::phm::HealthChannel::~HealthChannel** ⌈

Kind:	function
Header file:	#include "ara/phm/health_channel.h"
Scope:	class ara::phm::HealthChannel
Symbol:	~HealthChannel()
Syntax:	`~HealthChannel () noexcept;`
Exception Safety:	noexcept
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} **Definition of API function ara::phm::HealthChannel::operator=** ⌈

Kind:	function
Header file:	#include "ara/phm/health_channel.h"
Scope:	class ara::phm::HealthChannel
Symbol:	operator=(const HealthChannel &channel)

▽

△

Syntax:	`HealthChannel & operator= (const HealthChannel &channel)=delete;`
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} **Definition of API function ara::phm::HealthChannel::operator=** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel.h"	
Scope:	class ara::phm::HealthChannel	
Symbol:	operator=(HealthChannel &&channel)	
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	
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} **Definition of API function ara::phm::RecoveryAction::RecoveryAction** ⌈

Kind:	function	
Header file:	#include "ara/phm/recovery_action.h"	
Scope:	class ara::phm::RecoveryAction	
Symbol:	RecoveryAction(const ara::core::InstanceSpecifier &instance)	
Syntax:	`explicit RecoveryAction (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier to the PPortPrototype of a PhmRecoveryAction Interface
Exception Safety:	not exception safe	
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} **Definition of API function ara::phm::RecoveryAction::RecoveryAction** ⌈

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

⌋(*RS_PHM_00003*)

[SWS_PHM_01150]{DRAFT} **Definition of API function ara::phm::RecoveryAction::RecoveryAction** ⌈

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

⌋(*RS_PHM_00003*)

8.3.3.2 RecoveryAction::Operator=

[SWS_PHM_01151]{DRAFT} **Definition of API function ara::phm::RecoveryAction::operator=** ⌈

Kind:	function	
Header file:	#include "ara/phm/recovery_action.h"	
Scope:	class ara::phm::RecoveryAction	
Symbol:	operator=(RecoveryAction &&ra)	
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	
Description:	Move assignment operator for RecoveryAction.	

⌋(*RS_PHM_00003*)

[SWS_PHM_01152]{DRAFT} **Definition of API function ara::phm::RecoveryAction::operator=** ⌈

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

⌋(*RS_PHM_00003*)

8.3.3.3 RecoveryAction::~RecoveryAction

[SWS_PHM_01145]{DRAFT} **Definition of API function ara::phm::RecoveryAction::~RecoveryAction** ⌈

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

⌋(*RS_PHM_00003*)

8.3.3.4 RecoveryAction::RecoveryHandler

[SWS_PHM_01142]{DRAFT} **Definition of API function ara::phm::RecoveryAction::RecoveryHandler** ⌈

Kind:	function	
Header file:	#include "ara/phm/recovery_action.h"	
Scope:	class ara::phm::RecoveryAction	
Symbol:	RecoveryHandler(const ara::exec::ExecutionErrorEvent &executionError, TypeOfSupervision supervision)	
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 elementary supervision which failed.
Return value:	None	

▽

△

Exception Safety:	not exception safe
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} **Definition of API function ara::phm::RecoveryAction::Offer** ⌈

Kind:	function	
Header file:	#include "ara/phm/recovery_action.h"	
Scope:	class ara::phm::RecoveryAction	
Symbol:	Offer()	
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.
Exception Safety:	not exception safe	
Thread Safety:	no	
Errors:	ara::phm::PhmErrc::kOfferFailed	Returned if service could not be offered due to failure of communication with PHM daemon
Description:	Enables potential invocations of RecoveryHandler.	

⌋(*RS_PHM_00003*)

8.3.3.6 RecoveryAction::StopOffer

[SWS_PHM_01144]{DRAFT} **Definition of API function ara::phm::RecoveryAction::StopOffer** ⌈

Kind:	function
Header file:	#include "ara/phm/recovery_action.h"
Scope:	class ara::phm::RecoveryAction
Symbol:	StopOffer()
Syntax:	`void StopOffer ();`
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	no
Description:	Disables invocations of RecoveryHandler.

⌋(*RS_PHM_00003*)

8.3.4 HealthChannelAction API

8.3.4.1 HealthChannelAction::HealthChannelAction

[SWS_PHM_01231]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::HealthChannelAction** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel_action.h"	
Scope:	class ara::phm::HealthChannelAction	
Symbol:	HealthChannelAction(const ara::core::InstanceSpecifier &instance)	
Syntax:	`explicit HealthChannelAction (const ara::core::InstanceSpecifier &instance);`	
Parameters (in):	instance	instance specifier to the PPortPrototype of a PhmHealthChannelActionInterface
Exception Safety:	not exception safe	
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} **Definition of API function ara::phm::HealthChannelAction::HealthChannelAction** ⌈

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

⌋(*RS_PHM_00003*)

[SWS_PHM_01234]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::HealthChannelAction** ⌈

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

⌋(*RS_PHM_00003*)

8.3.4.2 HealthChannelAction::Operator=

[SWS_PHM_01235]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::operator=** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel_action.h"	
Scope:	class ara::phm::HealthChannelAction	
Symbol:	operator=(HealthChannelAction &&hca)	
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	
Description:	Move assignment operator for HealthChannelAction.	

⌋(*RS_PHM_00003*)

[SWS_PHM_01236]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::operator=** ⌈

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

⌋(*RS_PHM_00003*)

8.3.4.3 HealthChannelAction::~HealthChannelAction

[SWS_PHM_01232]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::~HealthChannelAction** ⌈

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

⌋(*RS_PHM_00003*)

8.3.4.4 HealthChannelAction::RecoveryHandler

[SWS_PHM_01237]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::RecoveryHandler** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel_action.h"	
Scope:	class ara::phm::HealthChannelAction	
Symbol:	RecoveryHandler(EnumT healthStatusId)	
Syntax:	`virtual void RecoveryHandler (EnumT healthStatusId)=0;`	
Parameters (in):	healthStatusId	The identifier representing the Health Status. The mapping is implementation specific.
Return value:	None	
Exception Safety:	not exception safe	
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} **Definition of API function ara::phm::HealthChannelAction::Offer** ⌈

Kind:	function	
Header file:	#include "ara/phm/health_channel_action.h"	
Scope:	class ara::phm::HealthChannelAction	
Symbol:	Offer()	
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.
Exception Safety:	not exception safe	
Thread Safety:	no	
Errors:	ara::phm::PhmErrc::kOfferFailed	Returned if service could not be offered due to failure of communication with PHM daemon
Description:	Enables potential invocations of RecoveryHandler.	

⌋(*RS_PHM_00003*)

8.3.4.6 HealthChannelAction::StopOffer

[SWS_PHM_01239]{OBSOLETE} **Definition of API function ara::phm::HealthChannelAction::StopOffer** ⌈

Kind:	function
Header file:	#include "ara/phm/health_channel_action.h"
Scope:	class ara::phm::HealthChannelAction
Symbol:	StopOffer()
Syntax:	`void StopOffer ();`
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	no
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] Definition of API enum ara::phm::PhmErrc ⌈

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

⌋(*RS_AP_00119*)

8.4.2 GetPhmDomain

[SWS_PHM_01251]{DRAFT} **Definition of API function ara::phm::GetPhmDomain**
⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	namespace ara::phm	
Symbol:	GetPhmDomain()	
Syntax:	`constexpr const ara::core::ErrorDomain & GetPhmDomain () noexcept;`	
Return value:	const ara::core::Error Domain &	The global PhmErrorDomain object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Returns the global PhmErrorDomain object.	

⌋(*RS_AP_00119*, *RS_AP_00132*)

8.4.3 MakeErrorCode

[SWS_PHM_01244]{DRAFT} **Definition of API function ara::phm::MakeErrorCode**
⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	namespace ara::phm	
Symbol:	MakeErrorCode(PhmErrc code, ara::core::ErrorDomain::SupportDataType data)	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (PhmErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	Error code number.
	data	Vendor defined data associated with the error.
Return value:	ara::core::ErrorCode	An ErrorCode object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Creates an error code.	

⌋(*RS_AP_00119*, *RS_AP_00132*)

8.4.4 PhmException Class

There are no standardized exceptions w.r.t `Platform Health Management`. But the API is added to be in sync with the base class from `ara::core`.

[SWS_PHM_01242]{DRAFT} **Definition of API class ara::phm::PhmException** ⌈

Kind:	class
Header file:	#include "ara/phm/phm_error_domain.h"
Forwarding header file:	#include "ara/phm/phm_fwd.h"
Scope:	namespace ara::phm
Symbol:	PhmException
Base class:	ara::core::Exception
Syntax:	`class PhmException : public ara::core::Exception {...};`
Description:	Exception type thrown by Platform Health Management.

⌋(*RS_AP_00119*)

8.4.4.1 PhmException::PhmException

[SWS_PHM_01243]{DRAFT} **Definition of API function ara::phm::PhmException::PhmException** ⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	class ara::phm::PhmException	
Symbol:	PhmException(ara::core::ErrorCode errorCode)	
Syntax:	`explicit PhmException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	noexcept	
Description:	Construct a new PlatformHealthManagement exception object containing an error code.	

⌋(*RS_AP_00119*, *RS_AP_00132*)

8.4.5 PhmErrorDomain Class

The error handling requires an `ara::core::ErrorDomain`, which can be used to check the errors returned via `ara::core::Result`.

[SWS_PHM_01241] Definition of API class ara::phm::PhmErrorDomain ⌈

Kind:	class
Header file:	#include "ara/phm/phm_error_domain.h"
Forwarding header file:	#include "ara/phm/phm_fwd.h"
Scope:	namespace ara::phm
Symbol:	PhmErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class PhmErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	0x8000'0000'0000'0602
Description:	Defines the error domain for Platform Health Management.

⌋(*RS_AP_00119*)

8.4.5.1 PhmErrorDomain::Errc

[SWS_PHM_01245]{DRAFT} **Definition of API type ara::phm::PhmErrorDomain::Errc** ⌈

Kind:	type alias
Header file:	#include "ara/phm/phm_error_domain.h"
Scope:	class ara::phm::PhmErrorDomain
Symbol:	Errc
Syntax:	`using Errc = PhmErrc;`
Description:	Alias for the error code value enumeration.

⌋(*RS_AP_00119*, *RS_AP_00127*)

8.4.5.2 PhmErrorDomain::Exception

[SWS_PHM_01246]{DRAFT} **Definition of API type ara::phm::PhmErrorDomain::Exception** ⌈

Kind:	type alias
Header file:	#include "ara/phm/phm_error_domain.h"
Scope:	class ara::phm::PhmErrorDomain
Symbol:	Exception
Syntax:	`using Exception = PhmException;`
Description:	Alias for the exception base class.

⌋(*RS_AP_00119*, *RS_AP_00127*)

8.4.5.3 PhmErrorDomain::PhmErrorDomain

[SWS_PHM_01247]{DRAFT} **Definition of API function ara::phm::PhmErrorDomain::PhmErrorDomain** ⌈

Kind:	function
Header file:	#include "ara/phm/phm_error_domain.h"
Scope:	class ara::phm::PhmErrorDomain
Symbol:	PhmErrorDomain()
Syntax:	`PhmErrorDomain () noexcept;`
Exception Safety:	noexcept
Thread Safety:	no
Description:	Creates a PhmErrorDomain instance.

⌋(*RS_AP_00119*, *RS_AP_00132*)

8.4.5.4 PhmErrorDomain::Name

[SWS_PHM_01248]{DRAFT} **Definition of API function ara::phm::PhmErrorDomain::Name** ⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	class ara::phm::PhmErrorDomain	
Symbol:	Name()	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Phm".
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Returns the name of the error domain.	

⌋(*RS_AP_00119*, *RS_AP_00132*)

8.4.5.5 PhmErrorDomain::Message

[SWS_PHM_01249]{DRAFT} **Definition of API function ara::phm::PhmErrorDomain::Message** ⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	class ara::phm::PhmErrorDomain	
Symbol:	Message(CodeType errorCode)	
Syntax:	`const char * Message (CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code.
Exception Safety:	noexcept	
Thread Safety:	no	
Description:	Returns the message associated with the error code.	

⌋(*RS_AP_00119*, *RS_AP_00132*)

8.4.5.6 PhmErrorDomain::ThrowAsException

There are no standardized exceptions w.r.t `Platform Health Management`. But these are added to be in sync with the base class from `ara::core()`.

[SWS_PHM_01250]{DRAFT} **Definition of API function ara::phm::PhmErrorDomain::ThrowAsException** ⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	class ara::phm::PhmErrorDomain	
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	no	
Description:	Throws the exception associated with the error code.	

⌋(*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 [13].

Chapter is generated.

Class	**AliveSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines an AliveSupervision for one checkpoint.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Aggregated by	GlobalSupervision.aliveSupervision			
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.
checkpoint	SupervisionCheckpoint	0..1	ref	Reference to a checkpoint in the context of Alive Supervision.
expectedAlive Indications	PositiveInteger	0..1	attr	Defines the amount of expected Alive Indications of the SupervisionCheckpoint within the aliveReferenceCycle.
failedReference Cycles Tolerance	PositiveInteger	0..1	attr	This attribute defines the acceptable amount of alive ReferenceCycles with incorrect/failed AliveSupervision.
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.
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.
terminating Checkpoint	SupervisionCheckpoint	0..1	ref	Reference to the SupervisionCheckpoint which is defined as the terminating checkpoint of this AliveSupervision.
terminating Checkpoint TimeoutUntil Termination	TimeValue	0..1	attr	Defines the time a process shall terminate after it has announced its start of termination by reporting terminatingCheckpoint.

Table A.1: AliveSupervision

Class	**DeadlineSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines an DeadlineSupervision for one transition.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Aggregated by	GlobalSupervision.deadlineSupervision			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
maxDeadline	TimeValue	0..1	attr	Defines the longest time span before which the deadline is considered to be met for transition.
minDeadline	TimeValue	0..1	attr	Defines the shortest time span after which the deadline is considered to be met for transition.
transition	CheckpointTransition	0..1	ref	Reference to the transition in the context of a Deadline Supervision.

Table A.2: DeadlineSupervision

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.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	PlatformHealthManagementContribution.globalSupervision			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
alive Supervision	AliveSupervision	*	aggr	Collection of AliveSupervisions in the context of this GlobalSupervision.
deadline Supervision	DeadlineSupervision	*	aggr	Collection of DeadlineSupervisions in the context of this GlobalSupervision.
logical Supervision	LogicalSupervision	*	aggr	Collection of LogicalSupervisions in the context of this GlobalSupervision.
noCheckpoint Supervision	NoCheckpoint Supervision	*	aggr	Definition of No Checkpoint Supervision.
noSupervision	NoSupervision	*	aggr	Collection of NoSupervisions in the context of this Global Supervision.
supervision Mode	SupervisionMode	*	aggr	Collection of SupervisionModes in the context of this GlobalSupervision. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=supervisionMode.shortName
transition	CheckpointTransition	*	aggr	Collection of CheckpointTransitions in the context of this GlobalSupervision.

Table A.3: GlobalSupervision

Class	***HealthChannel*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines the source of a health channel.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	HealthChannelExternalStatus, HealthChannelSupervision			
Aggregated by	PlatformHealthManagementContribution.healthChannel			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
recovery Notification	RecoveryNotification	*	ref	Defines the RecoveryNotification for this HealthChannel.

Table A.4: HealthChannel

Class	**HealthChannelExternalStatus**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines a health channel representing the status of an external health channel.			
Base	*ARObject*, *HealthChannel*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	PlatformHealthManagementContribution.healthChannel			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
healthChannel	RPortPrototype	0..1	iref	Refers to the HealthChannel. **Stereotypes:** atpUriDef **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef
notifiedStatus	HealthChannelExternal ReportedStatus	*	aggr	This is a list of statuses which shall trigger the Recovery Notification of this HealthChannelExternalStatus.
process	Process	0..1	ref	Defines the Process this Health Channel shall be monitored.

Table A.5: 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.6: ImplementationProps

Class	**LogicalSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines a LogicalSupervision graph consisting of transitions, initial- and final checkpoints.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Aggregated by	GlobalSupervision.logicalSupervision			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
finalCheckpoint	SupervisionCheckpoint	*	ref	Reference to the final Checkpoint(s) for this Logical Supervision. **Tags:** xml.sequenceOffset=20
initialCheckpoint	SupervisionCheckpoint	*	ref	Reference to the initial Checkpoint(s) for this Logical Supervision. **Tags:** xml.sequenceOffset=10
transition	CheckpointTransition	*	ref	Reference to the transitions for this LogicalSupervision. **Tags:** xml.sequenceOffset=30

Table A.7: LogicalSupervision

Class	**NoCheckpointSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines explicitly that NO supervision shall be applied for a set of SupervisionCheckpoints.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Aggregated by	GlobalSupervision.noCheckpointSupervision			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
checkpoint	SupervisionCheckpoint	*	ref	Reference to the set of SupervisionCheckpoints which shall not be considered for any kind of supervision.

Table A.8: NoCheckpointSupervision

Class	**NoSupervision**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	Defines explicitly that NO supervision shall be applied for a specific Supervised Entity instance.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *PhmSupervision*, *Referrable*			
Aggregated by	GlobalSupervision.noSupervision			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
process	Process	0..1	ref	Reference to the Process this NoSupervision applies to.

▽

△

Class	**NoSupervision**			
targetPhm Supervised Entity	RPortPrototype	0..1	iref	Instance reference to the RPortPrototype which represents the Supervised Entity instance. **Stereotypes:** atpUriDef **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef

Table A.9: 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.			
Base	*ARObject*, *AtpFeature*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, PhmSupervisedEntityInterface.checkpoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
checkpointId	PositiveInteger	0..1	attr	Defines the numeric value which is used to indicate the reporting of this Checkpoint to the Phm.

Table A.10: 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.recommendedPackage=PlatformHealthManagementInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *Port Interface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
status	PhmHealthChannel Status	*	aggr	Defines the possible set of status information available to the health channel.

Table A.11: PhmHealthChannelInterface

Class	**PhmHealthChannelStatus**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	The PhmHealthChannelStatus specifies one possible status of the health channel.			
Base	*ARObject*, *AtpFeature*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, PhmHealthChannelInterface.status			
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.

▽

△

Class	PhmHealthChannelStatus			
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.

Table A.12: PhmHealthChannelStatus

Class	PhmSupervisedEntityInterface			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class provides the ability to implement a PortInterface for interaction with the Platform Health Management Supervised Entity. **Tags:** atp.recommendedPackage=PlatformHealthManagementInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *Port Interface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
checkpoint	PhmCheckpoint	*	aggr	Defines the set of checkpoints which can be reported on this supervised entity.

Table A.13: 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*, FirewallStateSwitchInterface, *IdsmAbstractPort Interface*, LogAndTraceInterface, ModeSwitchInterface, NetworkManagementPortInterface, *Persistency Interface*, *PlatformHealthManagementInterface*, ServiceInterface, *StateManagementPortInterface*, TriggerInterface			
Aggregated by	ARPackage.element			
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

Table A.14: PortInterface

Class	**Process**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class provides information required to execute the referenced `Executable`. **Tags:** atp.recommendedPackage=Processes			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
design	ProcessDesign	0..1	ref	This reference represents the identification of the design-time representation for the Process that owns the reference.
executable	Executable	*	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the Process is affiliated with.
numberOf RestartAttempts	PositiveInteger	0..1	attr	This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory.
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process.
securityEvent	SecurityEventDefinition	*	ref	The reference identifies the collection of SecurityEvents that can be reported by the Process. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=candidate
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations.

Table A.15: Process

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.recommendedPackage=ProcessExecutionErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
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.

Table A.16: 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.recommendedPackage=RecoveryNotifications			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
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.

Table A.17: 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.recommendedPackage=RecoveryNotificationMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
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.
recoveryAction	PPortPrototype	0..1	iref	This reference identifies the PortPrototype to be addressed as part of a PHM recovery. **InstanceRef implemented by:** PPortPrototypeIn ExecutableInstanceRef
recovery Notification	RecoveryNotification	0..1	ref	This reference identifies the applicable Recovery Notification to be mapped.

Table A.18: 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, Bsw VariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescription Entity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, Pnc MappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, Someip RequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	***Referrable*** (abstract)			
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.19: Referrable

Class	**StartupConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class represents a reusable startup configuration for processes.. **Tags:** atp.recommendedPackage=StartupConfigs			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
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.
executionError	ProcessExecutionError	0..1	ref	this reference is used to identify the applicable execution error
permissionTo CreateChild Process	Boolean	0..1	attr	This attribute defines if Process is permitted to create child Processes. When setting this parameter to true two things should be kept in mind: 1) safety and security implication of this configuration, 2) the fact that Process will assume management responsibilities for child Processes (i.e. it will be responsible for terminating Processes that it creates).
process Argument (ordered)	ProcessArgument	*	aggr	This aggregation represents the collection of command-line arguments applicable to the enclosing StartupConfig.
scheduling Policy	String	0..1	attr	This attribute represents the ability to define the scheduling policy for the initial thread of the application.
scheduling Priority	Integer	0..1	attr	This is the scheduling priority requested by the application itself.
termination Behavior	TerminationBehavior Enum	0..1	attr	This attribute defines the termination behavior of the Process.
timeout	EnterExitTimeout	0..1	aggr	This aggregation can be used to specify the timeouts for launching and terminating the process depending on the StartupConfig.

Table A.20: 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.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	PlatformHealthManagementContribution.checkpoint			
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.
phmCheckpoint	PhmCheckpoint	0..1	iref	Instance reference to the PhmCheckpoint defined in the context of a PortInterface. **Stereotypes:** atpUriDef **InstanceRef implemented by:** PhmCheckpointIn ExecutableInstanceRef
process	Process	0..1	ref	Reference to the Process this checkoint shall be monitored.

Table A.21: SupervisionCheckpoint

Class	**SupervisionMode**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::PlatformHealthManagement			
Note	This element defines a SupervisionMode.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	GlobalSupervision.supervisionMode			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
active Supervision	PhmSupervision	*	ref	The reference defines which PhmSupervisions shall be active in this specific SupervisionMode.
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.
modeCondition	SupervisionMode Condition	0..1	ref	Reference to SupervisionModeCondition (Condition under which the configuration made under this SupervisionMode are to be applied).

Table A.22: 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			
Aggregated by	ARPackage.element			
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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portGroup.shortName, portGroup.variation Point.shortLabel 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.23: 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*			
Aggregated by	Allocator.namespace, ApApplicationErrorDomain.namespace, *AtomicSwComponentType*.symbolProps, *CppImplementationDataType*.namespace, ImplementationDataType.symbolProps, *PortInterface*.namespace, SecurityEventDefinition.eventSymbolName			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.24: 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 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)*

E Change History of AUTOSAR traceable items

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

E.1 Traceable item history of this document according to AUTOSAR Release R21-11

E.1.1 Added Specification Items in R21-11

Number	Heading
[SWS_PHM_00106]	Recovery Action for Failures in Execution or State Management
[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]	

▽

△

Number	Heading
[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]	
[SWS_PHM_00240]	Supervisions on termination of process
[SWS_PHM_00241]	Supervisions on Start of Process
[SWS_PHM_00242]	Supervisions on Restart of Process
[SWS_PHM_00243]	Continuation of Supervisions
[SWS_PHM_00244]	NoSupervision on Start of Process
[SWS_PHM_00245]	Continuation of NoSupervision (Supervision Exclusion)
[SWS_PHM_01240]	
[SWS_PHM_01241]	

Table E.1: Added Specification Items in R21-11

E.1.2 Changed Specification Items in R21-11

Number	Heading
[SWS_PHM_00101]	Notification to State Management due to Supervision failure
[SWS_PHM_00104]	Reaction on timeout for notification to State Management
[SWS_PHM_00105]	Recovery Action for Failures in Execution Management or State Management
[SWS_PHM_01005]	Namespace of generated header files for a Supervised Entity
[SWS_PHM_01113]	Namespace of generated header files for a Health Channel
[SWS_PHM_01127]	
[SWS_PHM_01128]	
[SWS_PHM_01132]	
[SWS_PHM_01136]	

▽

△

Number	Heading
[SWS_PHM_01137]	
[SWS_PHM_01142]	
[SWS_PHM_01143]	
[SWS_PHM_01146]	
[SWS_PHM_01149]	
[SWS_PHM_01150]	
[SWS_PHM_01151]	
[SWS_PHM_01152]	
[SWS_PHM_01212]	
[SWS_PHM_01213]	
[SWS_PHM_01214]	
[SWS_PHM_01215]	
[SWS_PHM_01222]	
[SWS_PHM_01223]	
[SWS_PHM_01224]	
[SWS_PHM_01225]	
[SWS_PHM_01227]	Consistency of Checkpoint Identifier
[SWS_PHM_01228]	Reporting of undefined Checkpoint Identifier
[SWS_PHM_01229]	Restricted access on reporting of Checkpoints
[SWS_PHM_01233]	
[SWS_PHM_01234]	
[SWS_PHM_01235]	
[SWS_PHM_01236]	
[SWS_PHM_01238]	
[SWS_PHM_01328]	Consistency of Health Status Identifier
[SWS_PHM_01329]	Reporting of undefined Health Status Identifier
[SWS_PHM_01330]	Restricted access on reporting of Health Status

Table E.2: Changed Specification Items in R21-11

E.1.3 Deleted Specification Items in R21-11

Number	Heading
[SWS_PHM_00103]	Timeout Monitoring for notification to State Management
[SWS_PHM_00321]	Underlying data types
[SWS_PHM_00458]	Creation of PHM service interface
[SWS_PHM_01010]	PHM Class

▽

△

Number	Heading
[SWS_PHM_01013]	Header file existence
[SWS_PHM_01018]	Header file namespace
[SWS_PHM_01101]	Folder structure for header files
[SWS_PHM_01116]	Definition of an identifier for a Supervised Entity
[SWS_PHM_01120]	Definition of an identifier for a Health Channel
[SWS_PHM_01121]	Definition of an identifier for a Health Channel Prototype
[SWS_PHM_01124]	Copy constructor for the use by SupervisedEntity and by HealthChannel
[SWS_PHM_01125]	The Platform Health Management shall provide a protected method ReportCheckpoint, provided by PHM
[SWS_PHM_01126]	The Platform Health Management shall provide a protected method ReportHealthStatus, provided by PHM
[SWS_PHM_01131]	Identifier Identifier Class Template
[SWS_PHM_01133]	Definition of an identifier for a Supervised Entity Prototype
[SWS_PHM_01134]	
[SWS_PHM_01135]	
[SWS_PHM_01160]	Restricted access on GetLocalSupervisionsStatus
[SWS_PHM_01161]	Restricted access on GetGlobalSupervisionStatus

Table E.3: Deleted Specification Items in R21-11

E.2 Traceable item history of this document according to AUTOSAR Release R22-11

E.2.1 Added Specification Items in R22-11

Number	Heading
[SWS_PHM_01242]	
[SWS_PHM_01243]	
[SWS_PHM_01244]	
[SWS_PHM_01245]	
[SWS_PHM_01246]	
[SWS_PHM_01247]	
[SWS_PHM_01248]	
[SWS_PHM_01249]	
[SWS_PHM_01250]	
[SWS_PHM_01251]	
[SWS_PHM_01252]	Handling of Watchdog after Startup

▽

△

Number	Heading
[SWS_PHM_01253]	Termination of Supervisions at SIGTERM
[SWS_PHM_01254]	Global Supervision Status at SIGTERM
[SWS_PHM_01331]	Start of Alive Supervision
[SWS_PHM_01332]	Checkpoints corresponding to Alive Supervision before kRunning
[SWS_PHM_01333]	Termination of Supervised Processes
[SWS_PHM_01334]	Time Source for Supervisions
[SWS_PHM_01335]	Stopping of Alive Supervision for Self-Terminating Process
[SWS_PHM_01336]	Timeout monitoring for termination of Self-Terminating Process
[SWS_PHM_01337]	Unintended termination of Self-Terminating Process
[SWS_PHM_01338]	Avoid redundant Monitoring of Termination for Self-Terminating Process
[SWS_PHM_01339]	Reporting access violation w.r.t. checkpoints to IdsM
[SWS_PHM_01341]	Reporting of Supervision Checkpoint mapped to No Supervision provision
[SWS_PHM_01342]	Tracking of Elementary Supervision Status
[SWS_PHM_01343]	States of state machine for Elementary Supervision Status
[SWS_PHM_01344]	Initialization of state machine for Elementary Supervision Status
[SWS_PHM_01345]	Keep Elementary Supervision Status `kOK`
[SWS_PHM_01346]	Switch Elementary Supervision Status from `kOK` to `kEXPIRED`
[SWS_PHM_01347]	Switch Elementary Supervision Status from `kOK` to `kFAILED`
[SWS_PHM_01348]	Keep Elementary Supervision Status `kFAILED`
[SWS_PHM_01349]	Switch Elementary Supervision Status from `kFAILED` to `kOK`
[SWS_PHM_01350]	Switch Elementary Supervision Status from `kFAILED` to `kEXPIRED`
[SWS_PHM_01351]	Switch Elementary Supervision Status from `kOK` to `kDEACTIVATED`
[SWS_PHM_01352]	Switch Elementary Supervision Status from `kFAILED` to `kDEACTIVATED`
[SWS_PHM_01353]	Keep Elementary Supervision Status `kDEACTIVATED`
[SWS_PHM_01354]	Switch Elementary Supervision Status from `kDEACTIVATED` to `kOK`
[SWS_PHM_01355]	Switch Elementary Supervision Status from `kEXPIRED` to `kDEACTIVATED`
[SWS_PHM_01356]	Keep Elementary Supervision Status `kEXPIRED`
[SWS_PHM_01357]	Switch Elementary Supervision Status from `kDEACTIVATED` to `kEXPIRED`
[SWS_PHM_01358]	

Table E.4: Added Specification Items in R22-11

E.2.2 Changed Specification Items in R22-11

Number	Heading
[SWS_PHM_00101]	Notification to State Management due to Supervision failure
[SWS_PHM_00105]	Recovery Action for Failures in Execution Management or State Management
[SWS_PHM_00106]	Recovery Action for Failures in Execution or State Management
[SWS_PHM_00216]	States of the state machine for Global Supervision Status
[SWS_PHM_00217]	One Global Supervision Status per Global Supervision
[SWS_PHM_00218]	Initialization of Global Supervision Status
[SWS_PHM_00220]	Switch Global Supervision Status from `kDEACTIVATED` to `kOK`
[SWS_PHM_00221]	Keep Global Supervision Status `kOK`
[SWS_PHM_00222]	Switch Global Supervision Status from `kOK` to `kDEACTIVATED`
[SWS_PHM_00223]	Switch Global Supervision Status from `kOK` to `kFAILED`
[SWS_PHM_00224]	Switch Global Supervision Status from `kOK` to `kEXPIRED` for SM/EM/OS supervision
[SWS_PHM_00225]	Switch Global Supervision Status from `kOK` to `kSTOPPED`
[SWS_PHM_00226]	Keep Global Supervision Status `kFAILED`
[SWS_PHM_00227]	Switch Global Supervision Status from `kFAILED` to `kOK`
[SWS_PHM_00228]	Switch Global Supervision Status from `kFAILED` to `kEXPIRED`
[SWS_PHM_00229]	Switch Global Supervision Status from `kFAILED` to `kSTOPPED`
[SWS_PHM_00230]	Keep Global Supervision Status `kEXPIRED`
[SWS_PHM_00231]	Switch Global Supervision Status from `kEXPIRED` to `kSTOPPED`
[SWS_PHM_00232]	Keep Global Supervision Status `kSTOPPED`
[SWS_PHM_00233]	Switch Global Supervision Status from `kEXPIRED` to `kOK`
[SWS_PHM_00234]	Switch Global Supervision Status from `kEXPIRED` to `kFAILED`
[SWS_PHM_00237]	Switch Global Supervision Status from `kDEACTIVATED` to `kFAILED`
[SWS_PHM_00238]	Switch Global Supervision Status from `kDEACTIVATED` to `kEXPIRED`
[SWS_PHM_00239]	Switch Global Supervision Status from `kDEACTIVATED` to `kSTOPPED`
[SWS_PHM_00424]	Enumeration for `Supervised Entity`
[SWS_PHM_00457]	
[SWS_PHM_01123]	
[SWS_PHM_01137]	
[SWS_PHM_01229]	Restricted access on reporting of Checkpoints
[SWS_PHM_01240]	
[SWS_PHM_01241]	

Table E.5: Changed Specification Items in R22-11

E.2.3 Deleted Specification Items in R22-11

Number	Heading
[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_00235]	
[SWS_PHM_00236]	
[SWS_PHM_01136]	
[SWS_PHM_01146]	
[SWS_PHM_01227]	Consistency of Checkpoint Identifier
[SWS_PHM_01228]	Reporting of undefined Checkpoint Identifier

Table E.6: Deleted Specification Items in R22-11

E.3 Traceable item history of this document according to AUTOSAR Release R23-11

E.3.1 Added Specification Items in R23-11

Number	Heading
[SWS_PHM_01340]	Security events for PHM

Table E.7: Added Specification Items in R23-11

E.3.2 Changed Specification Items in R23-11

Number	Heading
[SWS_PHM_00457]	Definition of API function ara::phm::HealthChannel::HealthChannel
[SWS_PHM_01123]	Definition of API function ara::phm::SupervisedEntity::SupervisedEntity
[SWS_PHM_01127]	Definition of API function ara::phm::SupervisedEntity::ReportCheckpoint
[SWS_PHM_01128]	Definition of API function ara::phm::HealthChannel::ReportHealthStatus
[SWS_PHM_01138]	Definition of API enum ara::phm::TypeOfSupervision
[SWS_PHM_01141]	Definition of API function ara::phm::RecoveryAction::RecoveryAction
[SWS_PHM_01142]	Definition of API function ara::phm::RecoveryAction::RecoveryHandler
[SWS_PHM_01143]	Definition of API function ara::phm::RecoveryAction::Offer
[SWS_PHM_01144]	Definition of API function ara::phm::RecoveryAction::StopOffer
[SWS_PHM_01231]	Definition of API function ara::phm::HealthChannelAction::HealthChannel Action
[SWS_PHM_01237]	Definition of API function ara::phm::HealthChannelAction::RecoveryHandler
[SWS_PHM_01238]	Definition of API function ara::phm::HealthChannelAction::Offer
[SWS_PHM_01239]	Definition of API function ara::phm::HealthChannelAction::StopOffer
[SWS_PHM_01240]	Definition of API enum ara::phm::PhmErrc
[SWS_PHM_01241]	Definition of API class ara::phm::PhmErrorDomain
[SWS_PHM_01250]	Definition of API function ara::phm::PhmErrorDomain::ThrowAsException
[SWS_PHM_01339]	Reporting access violation w.r.t. checkpoints to IdsM

Table E.8: Changed Specification Items in R23-11

E.3.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_PHM_00100]	Scope of Global Supervision

Table E.9: Deleted Specification Items in R23-11