Document Title	Specification of Platform Health Management
Document Category	Software specification (SWS)
Document UID	851
Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Added DaemonUnreachableViolation and aggregated several violations to UnresolvableInstanceSpecifierViolation • Added support of multiple Supervised Entities for same process • Extended support of disabling supervisions for Suspend To RAM
2025-11-27	R25-11	AUTOSAR Release Management	• Description of security issues extended • Chapter of Reporting extended
2024-11-27	R24-11	AUTOSAR Release Management	• Removed Health Channels from specification • Changed return type of RecoveryHandler API to ara::core::Future • Set SupervisedEntity class and RecoveryAction APIs to final • Update of threadsafety and exception safety information on APIs • Introduction of violation messages to ReportCheckpoint() and constructors
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

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 9
2 Acronyms and Abbreviations 10
3 Related documentation 12
3.1 Input documents & related standards and norms . . . 12
3.2 Further applicable specification . . . 12
4 Constraints and assumptions 13
4.1 Known limitations . . . 13
4.2 Applicability to car domains . . . 14
5 Dependencies to other Functional Clusters 15
5.1 Provided Interfaces . . . 15
5.2 Required Interfaces . . . 16
5.3 Additional dependencies on Execution Management . . . 17
6 Requirements Tracing 18
7 Functional specification 21
7.1 General description . . . 21
7.2 Supervision of Supervised Entities . . . 21
7.2.1 Start and Stop of Supervisions . . . 25
7.2.1.1 Stopping of Alive Supervision for Self Terminating Process . . . 26
7.2.2 Supervision of processes started before Platform Health Management . . . 28
7.2.3 Deactivation of Supervision during runtime . . . 28
7.3 Supervision Modes . . . 29
7.3.1 Effect of changing Mode . . . 30
7.4 Determination of Supervision Status . . . 32
7.4.1 Determination of Elementary Supervision Status . . . 32
7.4.2 Determination of Global Supervision Status . . . 38
7.5 Recovery actions . . . 46
7.5.1 Notificaton to State Management . . . 47
7.5.2 Handling of Hardware Watchdog . . . 50
7.5.3 Configuration Parameters . . . 51
7.6 Multiple processes and multiple instances . . . 52
7.7 Functional cluster life-cycle . . . 54
7.7.1 Startup . . . 54
7.7.2 Shutdown . . . 54
7.7.2.1 Handling of watchdog during shutdown . . . 54
7.8 Reporting . . . 55
7.8.1 Security Events . . . 55
7.8.2 Log Messages . . . 55

7.8.3 Violation Messages . . . 57
7.8.4 Production Errors . . . 59
7.8.4.1 PHM_E_WATCHDOG_RESET . . . 59
7.8.5 Lost Daemon Connection . . . 59
8 API specification 61
8.1 PortInterface to API class binding . . . 61
8.2 Header: ara/phm/supervised_entities/{<si-namespace-derived-directory-path-lower>}/{<phmssi-sn>}.h . . . 62
8.2.1 Namespaces . . . 63
8.2.1.1 Namespace: ara::phm::supervised_entities::{<hierarchical-namespace-list-lower-skeleton>} . . . 63
8.2.2 Non-Member Types . . . 63
8.2.3 Global Variables . . . 64
8.3 Header: ara/phm/phm_error_domain.h . . . 64
8.3.1 Non-Member Types . . . 64
8.3.2 Non-Member Functions . . . 65
8.3.2.1 Other . . . 65
8.3.3 Class: PhmErrorDomain . . . 66
8.3.3.1 Public Member Types . . . 66
8.3.3.2 Public Member Functions . . . 67
8.3.3.2.1 Special Member Functions . . . 67
8.3.3.2.2 Member Functions . . . 67
8.3.4 Class: PhmException . . . 69
8.3.4.1 Public Member Functions . . . 69
8.3.4.1.1 Constructors . . . 69
8.4 Header: ara/phm/recovery_action.h . . . 70
8.4.1 Non-Member Types . . . 70
8.4.2 Class: RecoveryAction . . . 70
8.4.2.1 Public Member Functions . . . 71
8.4.2.1.1 Special Member Functions . . . 71
8.4.2.1.2 Constructors . . . 72
8.4.2.1.3 Member Functions . . . 73
8.5 Header: ara/phm/supervised_entity.h . . . 75
8.5.1 Non-Member Types . . . 75
8.5.2 Class: SupervisedEntity . . . 76
8.5.2.1 Public Member Functions . . . 77
8.5.2.1.1 Special Member Functions . . . 77
8.5.2.1.2 Constructors . . . 78
8.5.2.1.3 Member Functions . . . 79
8.6 Sequence Diagrams . . . 80
8.6.1 Suspend-to-RAM activation . . . 80
8.6.2 Disable supervision not allowed . . . 81
8.6.3 Resume STR . . . 82

8.6.4 Checkpoint ignored after Disable . . . 83
9 Service Interfaces 85
10 Configuration 86
10.1 Default Values . . . 86
10.2 Semantic Constraints . . . 86
A Mentioned Manifest Elements 87
B Demands and constraints on Base Software (normative) 99
C Platform Extension API (normative) 100
C.1 Header: apext/phm/watchdog_interface.h . . . 100
C.1.1 Class: WatchdogInterface . . . 100
C.1.1.1 Public Member Functions . . . 100
C.1.1.1.1 Member Functions . . . 100
D Not implemented requirements 103
E Change History of AUTOSAR traceable items 104
E.1 Traceable item history of this document according to AUTOSAR Release R21-11 . . . 104
E.1.1 Added Specification Items in R21-11 . . . 104
E.1.2 Changed Specification Items in R21-11 . . . 105
E.1.3 Deleted Specification Items in R21-11 . . . 106
E.2 Traceable item history of this document according to AUTOSAR Release R22-11 . . . 107
E.2.1 Added Specification Items in R22-11 . . . 107
E.2.2 Changed Specification Items in R22-11 . . . 108
E.2.3 Deleted Specification Items in R22-11 . . . 110
E.3 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 110
E.3.1 Added Specification Items in R23-11 . . . 110
E.3.2 Changed Specification Items in R23-11 . . . 111
E.3.3 Deleted Specification Items in R23-11 . . . 111
E.4 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 111
E.4.1 Added Specification Items in R24-11 . . . 111
E.4.2 Changed Specification Items in R24-11 . . . 112
E.4.3 Deleted Specification Items in R24-11 . . . 112
E.4.4 Added Constraints in R24-11 . . . 112
E.4.5 Changed Constraints in R24-11 . . . 112
E.4.6 Deleted Constraints in R24-11 . . . 112
E.5 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 113
E.5.1 Added Specification Items in R25-11 . . . 113

E.5.2 Changed Specification Items in R25-11 . . . 113
E.5.3 Deleted Specification Items in R25-11 . . . 113
E.5.4 Added Constraints in R25-11 . . . 113
E.5.5 Changed Constraints in R25-11 . . . 113
E.5.6 Deleted Constraints in R25-11 . . . 113
E.6 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 114
E.6.1 Added Specification Items in R26-11 . . . 114
E.6.2 Changed Specification Items in R26-11 . . . 114
E.6.3 Deleted Specification Items in R26-11 . . . 114
E.6.4 Added Constraints in R26-11 . . . 114
E.6.5 Changed Constraints in R26-11 . . . 114
E.6.6 Deleted Constraints in R26-11 . . . 114

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].

`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 that are only relevant within this specification. A general list of acronyms and abbreviations is available in [6].

Acronym:	Description:
E2E	AUTOSAR End to End communication protection mechanism
PHM	Platform Health Management
SE	Supervised Entity

Table 2.1: Acronyms and abbreviations used in the scope of this Document

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	The current status 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 `Processes`, which need to be controlled consistently. Depending on the state of the `Function Group`, `Processes` are started or terminated. Function Groups and their state are controlled by StateManagement, see [7] for more details.

▽

△

Function Group State	The element of State Management that characterizes the current status of a set of (functionally coherent) user level Applications. The set of `Function Groups` and their `Function Group States` is machine specific and are configured in the Machine Manifest. See [7] for more details.
Global Supervision Status	Status that summarizes the `Elementary Supervision Status` of a set of supervisions within a `Function Group`.
Health Monitoring	Supervision of the software behaviour for correct timing and sequence.
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	`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".
Suspend-To-RAM (S3)	An ACPI power mode in which the system is suspended to memory while preserving context in RAM; see [8].

Table 2.2: Technical terms used in the Scope of this Document

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 Road vehicles -– Functional Safety
https://www.iso.org

[6] Glossary
AUTOSAR_FO_TR_Glossary

[7] Specification of State Management
AUTOSAR_AP_SWS_StateManagement

[8] Advanced Configuration and Power Interface

[9] General Specification of Adaptive Platform
AUTOSAR_AP_SWS_General

[10] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[11] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[12] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[13] Specification of Adaptive Platform Core
AUTOSAR_AP_SWS_Core

[14] Guidelines for using Adaptive Platform interfaces
AUTOSAR_AP_EXP_InterfacesGuidelines

3.2 Further applicable specification

AUTOSAR provides a general specification [9] which is also applicable for this functional cluster. It shall be considered an additional and required specification for implementing this functional cluster.

4 Constraints and assumptions

4.1 Known limitations

- Daisy chaining (i.e. forwarding Supervision Status or Checkpoint 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.
- 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 [7] 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 defines the dependencies of this functional cluster to other functional clusters. AUTOSAR decided not to standardize interfaces which are exclusively used between functional clusters to allow efficient implementations which might depend e.g., on the used operating system. The goal of this chapter is to provide an informative guideline for the interactions between functional clusters without specifying syntactical details. This ensures compatibility between documents specifying different functional clusters and supports parallel implementation of different functional clusters. Details of internal interfaces are up to the platform provider. Additional internal interfaces, parameters, and return values can be added. A detailed technical architecture documentation of the overall AUTOSAR Adaptive Platform is provided in [11].

5.1 Provided Interfaces

This section provides an overview of the public interfaces provided by this functional cluster towards other functional clusters.

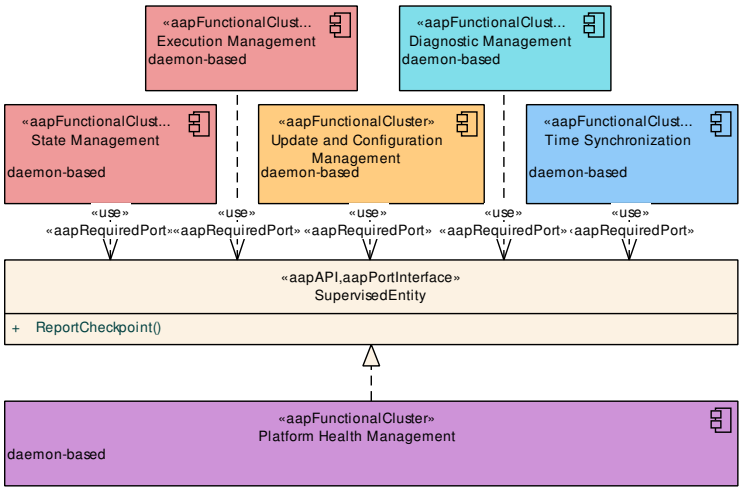


Figure 5.1: Interfaces provided by Platform Health Management to other Functional Clusters

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*
`RecoveryAction`	State Management	`Platform Health Management` uses this interface to trigger failure recovery.
`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`.
	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

This section provides an overview of the public interfaces required by this functional cluster from other functional clusters.

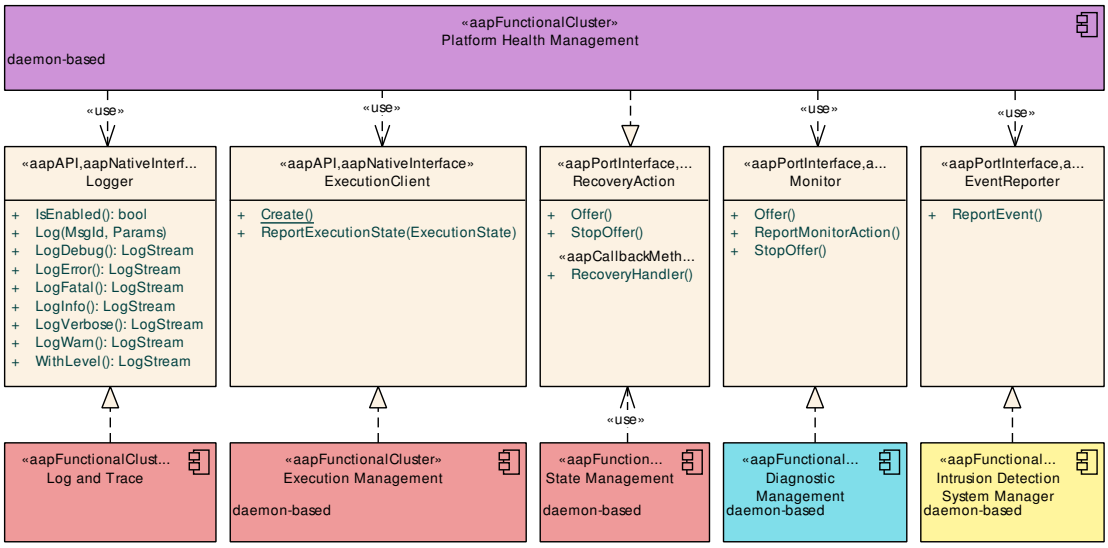


Figure 5.2: Interfaces required by Platform Health Management from other Functional Clusters

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*
Diagnostic Management	Monitor	`Platform Health Management` uses this interface to report production errors.
Execution Management	ExecutionClient	`Platform Health Management` uses this interface to report the state of its daemon process to `Execution Management`.
Intrusion Detection System Manager	EventReporter	`Platform Health Management` uses this interface to report security events.
Log and Trace	Logger	`Platform Health Management` shall use this interface to log standardized messages.

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 AUTOSAR RS PlatformHealthManagement [2] and AUTOSAR RS HealthMonitoring [3] 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_00114]	Compatibility with the ISO 14882 C++ standard	[SWS_PHM_01005]
[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_00122]	Type names	[SWS_PHM_00424]
[RS_AP_00127]	Usage of ara::core types	[SWS_PHM_00424] [SWS_PHM_01245] [SWS_PHM_01246]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment	[SWS_PHM_00424]
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_PHM_01145] [SWS_PHM_01211]
[RS_AP_00156]	Naming conventions for L&T Context ID	[SWS_PHM_01370] [SWS_PHM_01371] [SWS_PHM_01372] [SWS_PHM_01373] [SWS_PHM_01381]
[RS_AP_00159]	usage of “noexcept” specifier	[SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01130] [SWS_PHM_01141] [SWS_PHM_01142] [SWS_PHM_01143] [SWS_PHM_01144] [SWS_PHM_01149] [SWS_PHM_01151] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01243] [SWS_PHM_01244] [SWS_PHM_01247] [SWS_PHM_01248] [SWS_PHM_01249] [SWS_PHM_01251] [SWS_PHM_01380]
[RS_AP_00173]	InstanceSpecifierAlreadyInUse Violation	[SWS_PHM_01123] [SWS_PHM_01141]
[RS_AP_00177]	AraNotInitializedViolation	[SWS_PHM_01123]
[RS_AP_00180]	UnresolvableInstanceSpecifier Violation	[SWS_PHM_01123] [SWS_PHM_01141]
[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] [SWS_PHM_01365] [SWS_PHM_01367]
[RS_HM_09159]	Health Monitoring shall be able to report supervision errors.	[SWS_PHM_01138] [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]
[RS_HM_09169]	Health Monitoring shall be able to trigger microcontroller reset.	[SWS_PHM_01359] [SWS_PHM_01360]
[RS_HM_09222]	Health Monitoring shall provide a Logical Supervision	[SWS_PHM_01253] [SWS_PHM_01254]



△

Requirement	Description	Satisfied by
[RS_HM_09226]	Health Monitoring shall be able to wrongly trigger the serviced watchdogs.	[SWS_PHM_00104] [SWS_PHM_00105] [SWS_PHM_00106] [SWS_PHM_00107] [SWS_PHM_01359] [SWS_PHM_01360]
[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 Status.	[SWS_PHM_01137] [SWS_PHM_01358]
[RS_HM_09244]	Health Monitoring shall support timeout watchdogs.	[SWS_PHM_01252] [SWS_PHM_01362] [SWS_PHM_01363]
[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_00101] [SWS_PHM_00104] [SWS_PHM_00105] [SWS_PHM_00106] [SWS_PHM_00107] [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] [SWS_PHM_01359] [SWS_PHM_01360] [SWS_PHM_01365] [SWS_PHM_01367]
[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_01123] [SWS_PHM_01127] [SWS_PHM_01130] [SWS_PHM_01132] [SWS_PHM_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01229] [SWS_PHM_01255] [SWS_PHM_01256] [SWS_PHM_01257] [SWS_PHM_01258] [SWS_PHM_01259] [SWS_PHM_01341] [SWS_PHM_01380]
[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]
[RS_PHM_00105]	`Platform Health Management` shall support different allocations/ distributions of a Supervision through threads and processes.	[SWS_PHM_01383]
[RS_PHM_00106]	`Platform Health Management` shall support allocating of multiple `Supervised Entitys` to the same process.	[SWS_PHM_01383]

▽

△

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_01147] [SWS_PHM_01148] [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_00114]	`Platform Health Management` at highest safety integrity level	[SWS_PHM_00105] [SWS_PHM_00106] [SWS_PHM_00107] [SWS_PHM_01252]
[RS_PHM_00115]	If supervision of State Management fails then Platform Health Management shall trigger a watchdog reset.	[SWS_PHM_00105] [SWS_PHM_01359] [SWS_PHM_01360] [SWS_PHM_01362] [SWS_PHM_01363]
[RS_PHM_00116]	If supervision of Execution Management fails then Platform Health Management shall trigger a watchdog reset.	[SWS_PHM_00105] [SWS_PHM_01359] [SWS_PHM_01360]
[RS_PHM_00117]	Platform Health Management shall notify State Management in case an `AUTOSAR Adaptive Platform` functional cluster, `Adaptive Application` or service other than Execution Management and State Management fails.	[SWS_PHM_00101] [SWS_PHM_01147] [SWS_PHM_01148] [SWS_PHM_01361]
[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_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01255]
[RS_PHM_09241]	Health Monitoring shall support multiple instances of Checkpoints in a Supervised Entity occurrence.	[SWS_PHM_00424] [SWS_PHM_00425]

Table 6.1: Requirements Tracing

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). With the same means, functional clusters like State Management and Execution Management can be monitored. 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 described in this document.

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 [7]. 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.

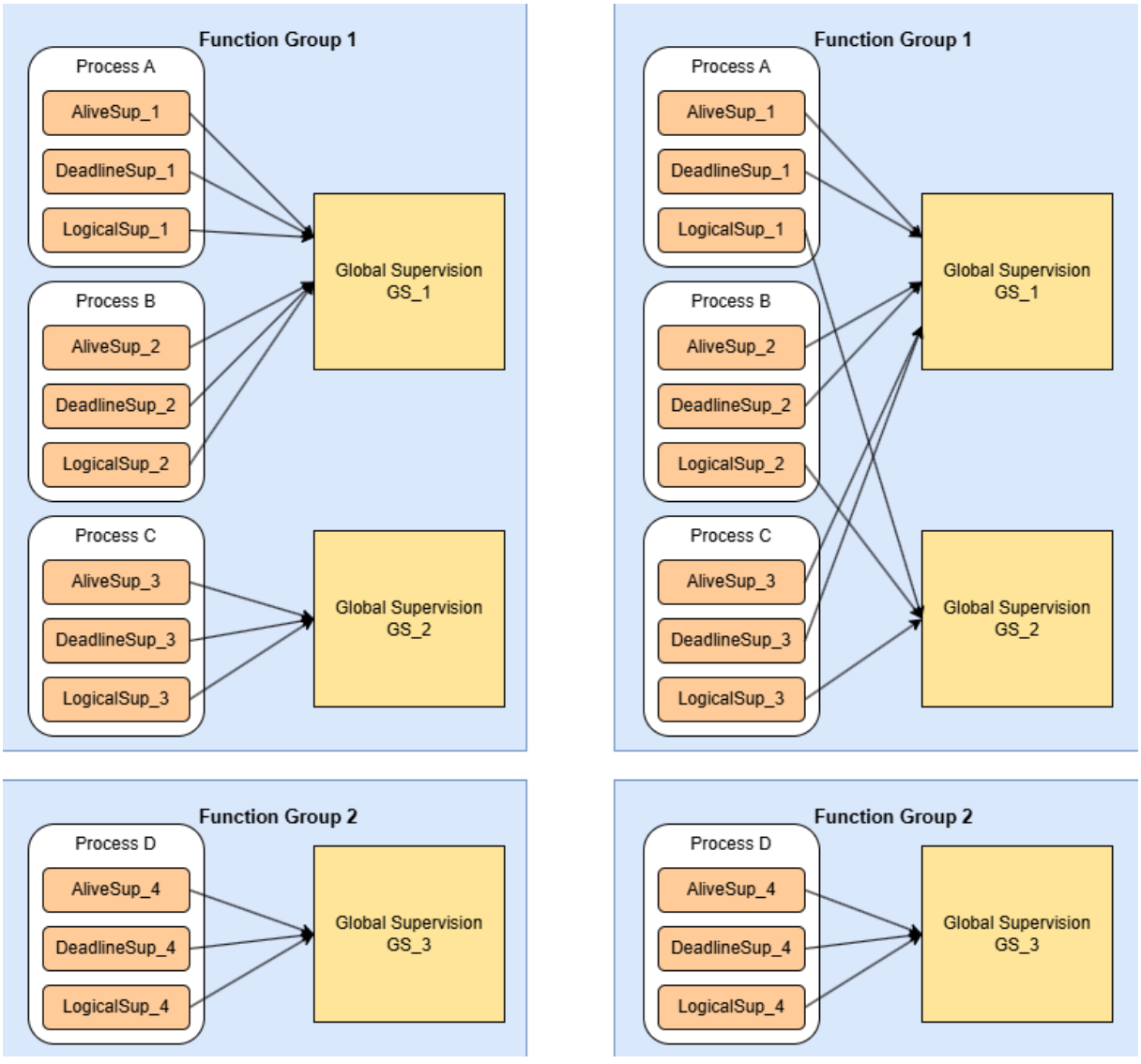


Figure 7.1: Allowed mappings of Elementary Supervisions to Global Supervisions

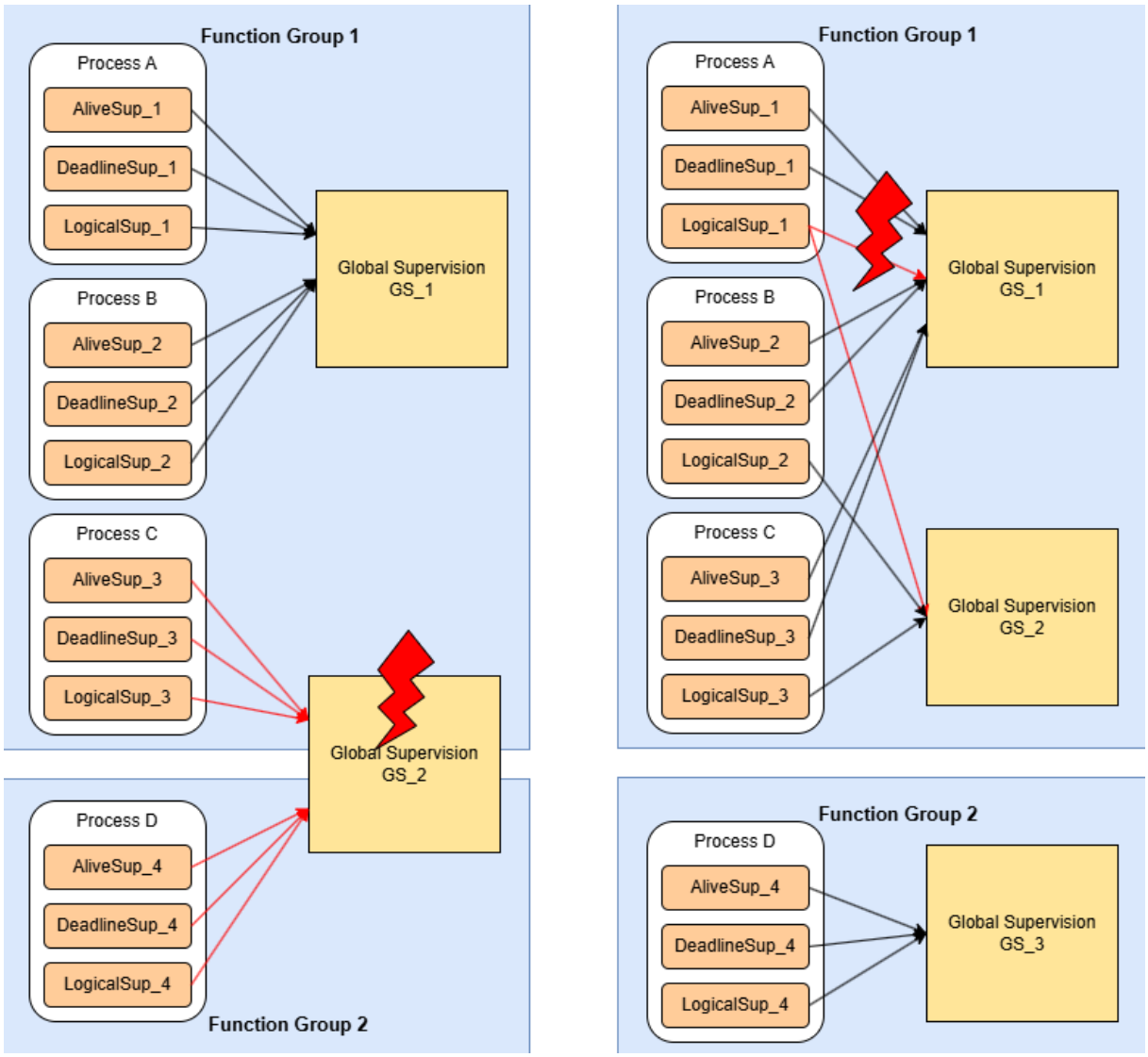


Figure 7.2: Mappings of Elementary Supervisions to Global Supervisions which are not supported

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] Reporting of Supervision Checkpoint mapped to No Supervision provision

Upstream requirements: RS_PHM_00101

⌈If a SupervisionCheckpoint reported to Platform Health Management via ara::phm::SupervisedEntity::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).⌋

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

[SWS_PHM_01229] Restricted access on reporting of Checkpoints

Upstream requirements: RS_PHM_00101, RS_PHM_00118

⌈The Platform Health Management shall ignore the execution of ara::phm::SupervisedEntity::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.⌋

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] Reporting access violation w.r.t. checkpoints to IdsM

Upstream requirements: RS_PHM_00119

⌈If access to resources was not granted by PHM due to wrong process reporting the `SupervisionCheckpoint` as described in [SWS_PHM_01229], the security event SEV_ACCESS_CONTROL_PHM_IAM_ACCESS_DENIED defined in [SWS_PHM_01340] shall be reported to IdsM.⌋

7.2.1 Start and Stop of Supervisions

Supervision of non reporting processes is not supported.

[SWS_PHM_01331] Start of Alive Supervision

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈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`.⌋

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] Checkpoints corresponding to Alive Supervision before kRunning

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈With respect to `Alive Supervision`, `Platform Health Management` shall ignore `Checkpoints` reported by a `Supervised Entity` Instance before the corresponding process reaches state `kRunning`.⌋

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 `SupervisionCheckpoint`s 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] Termination of Supervised Processes

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈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 `SupervisionCheckpoint`s of the corresponding process only).⌋

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] Time Source for Supervisions

Upstream requirements: RS_HM_09249

⌈All timing aspects related to `Platform Health Management` shall be measured in the context of the reporting process using the same time source.⌋

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] Stopping of Alive Supervision for Self-Terminating Process

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈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 Management that the process will be notified to terminate (by issuing SIGTERM), whichever is earlier.⌋

[SWS_PHM_01336] Timeout monitoring for termination of Self-Terminating Process

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈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, then Platform Health Management shall notify a failure of self termination to State Management via ara::phm::RecoveryAction::RecoveryHandler.⌋

[SWS_PHM_01337] Unintended termination of Self-Terminating Process

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈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.⌋

[SWS_PHM_01338] Avoid redundant Monitoring of Termination for Self-Terminating Process

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈If an Alive Supervision is configured for a Self Terminating Process and if after reporting of terminatingCheckpoint and before terminatingCheckpoint-TimeoutUntilTermination 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.⌋

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.

7.2.3 Deactivation of Supervision during runtime

There are situations in which deactivation of the monitoring is necessary because the overall system enters a degradation such as Suspend-To-RAM (S3). Deactivation is critical and must be taken into account in the safety case, as no further monitoring takes place during deactivation.

[SWS_PHM_01382] Register as a S2RSatellite (Suspend To Ram) ⌈PHM shall register a S2RSatellite (see `ara::sm::s2r::S2RSatellite::S2RSatellite`) while starting up.⌋

[SWS_PHM_01364] Temporarily disabling of Supervision ⌈A invocation of `ara::phm::SupervisedEntity::Disable` shall disable the supervision.

Note: A disabled Supervision is not longer monitored.⌋

[SWS_PHM_01386] Precondition to Disable a Supervision ⌈Disabling request will only be processed if the STR-Mode is activated i.e. PHM has received `ara::sm::s2r::S2RSatellite::EnterSuspendMode` invocation. Otherwise the errorcode `PhmErrc::kNotInSTRMode` shall be returned.⌋

In the SM interaction, PHM postpones entering suspend mode until all supervisions are disabled. This ensures that all safety supervisions are deactivated while the machine is in Suspend-To-RAM (S3) state.

[SWS_PHM_01384] Handling of EnterSuspendMode ⌈If `ara::sm::s2r::S2RSatellite::EnterSuspendMode` is invoked, PHM shall postpone the asynchronous return (i.e. fulfillment of the Promise) after all Supervisions are deactivated via `ara::phm::SupervisedEntity::Disable` and `apext::phm::WatchdogInterface::DisableWatchdog` shall be called. Note: `ara::phm::SupervisedEntity::Disable` has as precondition (see [SWS_PHM_01386]) the PHM has also entered the STR-mode.⌋

Further note, that SM has a timeout to supervise the return of `ara::sm::s2r::S2RSatellite::EnterSuspendMode`. If this timeout happens, the SM will resume from the suspend mode immediately.

In the SM interaction, PHM postpones returning from suspend mode until all supervisions have been re-enabled. The watchdog is activated immediately after resume is indicated to PHM to provide external supervision as early as possible.

Note: This is only necessary, if the hardware watchdog is NOT synchronized with the Suspend-To-RAM (S3) mode of the processor.

[SWS_PHM_01385] Handling of LeaveSuspendMode ⌈If ara::sm::s2r::S2RSatellite::LeaveSuspendMode is invoked, PHM shall immediately enable the watchdog by invoking apext::phm::WatchdogInterface::EnableWatchdog. The asynchronous response shall be postponed (i.e. fulfillment of the Promise) until all Supervisions are re-enabled via ara::phm::SupervisedEntity::Enable.⌋

[SWS_PHM_01365] Checkpoints are ignored while Supervision is disabled

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈Platform Health Management shall ignore Checkpoints reported by a Supervised Entity instance while the Supervision (Alive Supervision/Deadline Supervision/Logical Supervision) is disabled (which is the time between the call of ara::phm::SupervisedEntity::Disable and ara::phm::SupervisedEntity::Enable).⌋

[SWS_PHM_01366] Reactivation of Supervision ⌈A invocation of ara::phm::SupervisedEntity::Enable shall enable the Supervision. Further it shall restart the aliveReferenceCycle if the Supervision is configured as AliveSupervision.⌋

[SWS_PHM_01367] Checkpoints corresponding to Deadline Supervision/Logical Supervision while re-enabling

Upstream requirements: RS_HM_09125, RS_HM_09249

⌈With respect to Deadline Supervision/Logical Supervision, Platform Health Management shall keep ignoring Checkpoints reported by a Supervised Entity instance after re-enabling (i.e. a call to ara::phm::SupervisedEntity::Enable) until the Supervision is restarted (i.e. a CheckpointTransition.source is triggered).⌋

7.3 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.

7.3.1 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] Supervisions on termination of process

Upstream requirements: RS_PHM_00104

⌈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.⌋

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] Supervisions on Start of Process

Upstream requirements: RS_PHM_00104

⌈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.⌋

[SWS_PHM_00244] NoSupervision on Start of Process

Upstream requirements: RS_PHM_00104

⌈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`.⌋

Note: Even though it is supported to exclude (disable) Supervision in a particular `Supervision Mode`, dynamic change between Supervision inclusion (enable) and exclusion (disable) on `Function Group State` change wherein the process under consideration continues to execute on change in `Function Group State` is not supported in the normal behavior. 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.

However with the introduction of additional APIs, it is possible to `ara::phm::SupervisedEntity::Disable` or `ara::phm::SupervisedEntity::Enable` the supervision dynamically, independent of supervision mode.

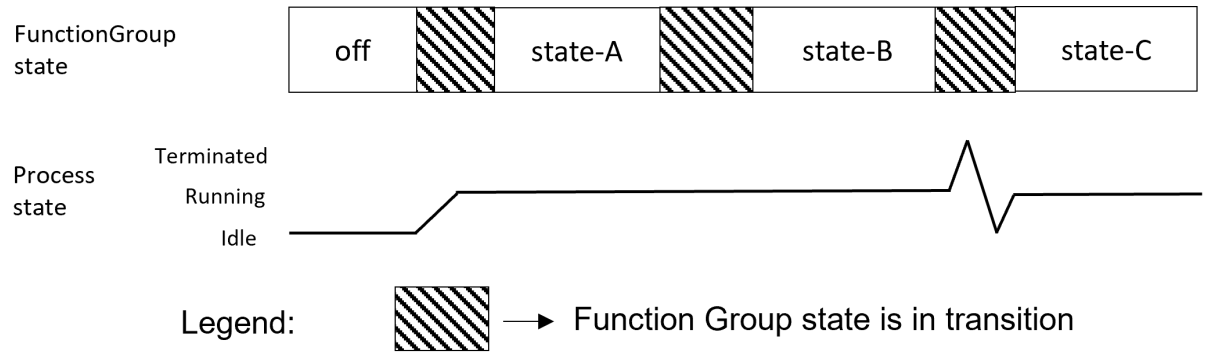


Figure 7.3: Supervision Exclusion and change of Function Group State

Figure 7.3 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] Supervisions on Restart of Process

Upstream requirements: RS_PHM_00104

⌈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]).⌋

[SWS_PHM_00243] Continuation of Supervisions

Upstream requirements: RS_PHM_00104

⌈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.⌋

[SWS_PHM_00245] Continuation of NoSupervision (Supervision Exclusion)

Upstream requirements: RS_PHM_00104

⌈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.⌋

7.4 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.

7.4.1 Determination of Elementary Supervision Status

The state machine of the `Elementary Supervision Status` 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] Tracking of Elementary Supervision Status

Upstream requirements: RS_PHM_00111

⌈The `Platform Health Management` shall track the `Elementary Supervision Status` of each `Alive Supervision`, `Deadline Supervision` and `Logical Supervision`.⌋

Figure 7.4 shows the state machine for `Elementary Supervision Status` of a supervision with all possible states.

[SWS_PHM_01343] States of state machine for Elementary Supervision Status

Upstream requirements: RS_PHM_00111

⌈The state machine of the `Elementary Supervision Status` shall have the states `kOK`, `kDeactivated`, `kExpired` and `kFailed`.⌋

See also figure 7.4 and `ara::phm::ElementarySupervisionStatus`.

Please note that the state `kFailed` is only relevant for `Alive Supervision`.

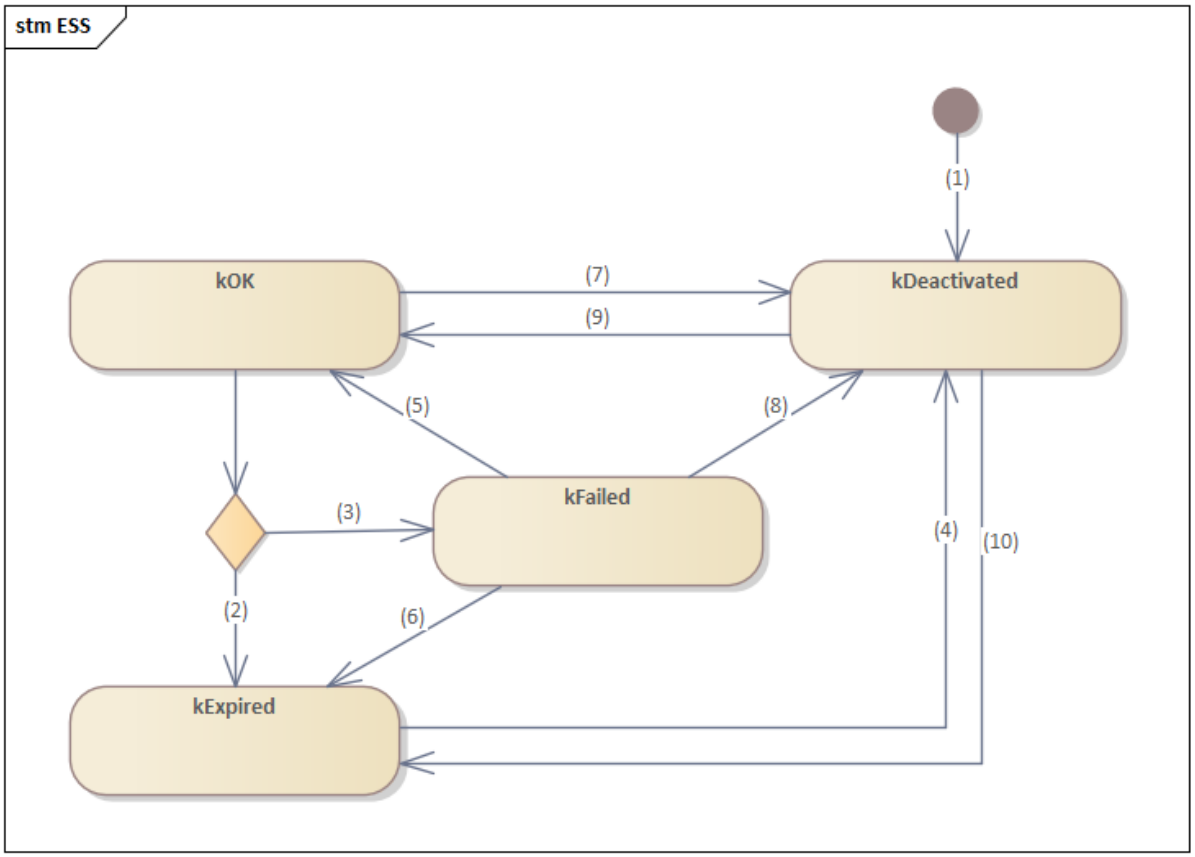


Figure 7.4: Elementary Supervision Status

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

[SWS_PHM_01344] Initialization of state machine for Elementary Supervision Status

Upstream requirements: RS_PHM_00111

⌈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).⌋

See transition (1) in figure 7.4.

[SWS_PHM_01345] Keep Elementary Supervision Status kOK

Upstream requirements: RS_PHM_00111

⌈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, then the Platform Health Management shall keep the supervision in the Elementary Supervision Status kOK .⌋

[SWS_PHM_01346] Switch Elementary Supervision Status from kOK to kExpired

Upstream requirements: RS_PHM_00111

⌈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.⌋

See transition (2) in figure 7.4.

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] Switch Elementary Supervision Status from kOK to kFailed

Upstream requirements: RS_PHM_00111

⌈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.⌋

See transition (3) in figure 7.4.

[SWS_PHM_01348] Keep Elementary Supervision Status kFailed

Upstream requirements: RS_PHM_00111

⌈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`.⌋

[SWS_PHM_01349] Switch Elementary Supervision Status from kFailed to kOK

Upstream requirements: RS_PHM_00111

⌈

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`.⌋

See transition (5) in figure 7.4.

[SWS_PHM_01350] Switch Elementary Supervision Status from kFailed to kExpired

Upstream requirements: RS_PHM_00111

⌈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.⌋

See transition (6) in figure 7.4.

[SWS_PHM_01351] Switch Elementary Supervision Status from kOK to kDeactivated

Upstream requirements: RS_PHM_00111, RS_PHM_00104

⌈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).⌋

See transition (7) in figure 7.4.

[SWS_PHM_01352] Switch Elementary Supervision Status from kFailed to kDeactivated

Upstream requirements: RS_PHM_00111, RS_PHM_00104

⌈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).⌋

See transition (8) in figure 7.4.

[SWS_PHM_01353] Keep Elementary Supervision Status kDeactivated

Upstream requirements: RS_PHM_00111, RS_PHM_00104

⌈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 .⌋

[SWS_PHM_01354] Switch Elementary Supervision Status from kDeactivated to kOK

Upstream requirements: RS_PHM_00111, RS_PHM_00104

⌈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 .⌋

See transition (9) in figure 7.4.

[SWS_PHM_01355] Switch Elementary Supervision Status from kExpired to kDeactivated

Upstream requirements: RS_PHM_00111, RS_PHM_00104

⌈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).⌋

See transition (4) in figure 7.4.

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] Keep Elementary Supervision Status kExpired

Upstream requirements: RS_PHM_00111, RS_PHM_00104

⌈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 .⌋

[SWS_PHM_01357] Switch Elementary Supervision Status from kDeactivated to kExpired

Upstream requirements: RS_PHM_00111

⌈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.⌋

See transition (10) in figure 7.4.

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

7.4.2 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 7.5 shows the values and transitions between them.

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

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

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

[SWS_PHM_00219] Calculation of Global Supervision Status

Upstream requirements: RS_PHM_00111

⌈The Platform Health Management shall calculate the Global Supervision Status of each configured GlobalSupervision.⌋

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] States of the state machine for Global Supervision Status

Upstream requirements: RS_PHM_00111

⌈The state machine of the Global Supervision Status shall have the states kOK , kDeactivated , kFailed , kExpired and

kStopped , see ara::phm::GlobalSupervisionStatus.⌋

See also figure 7.5.

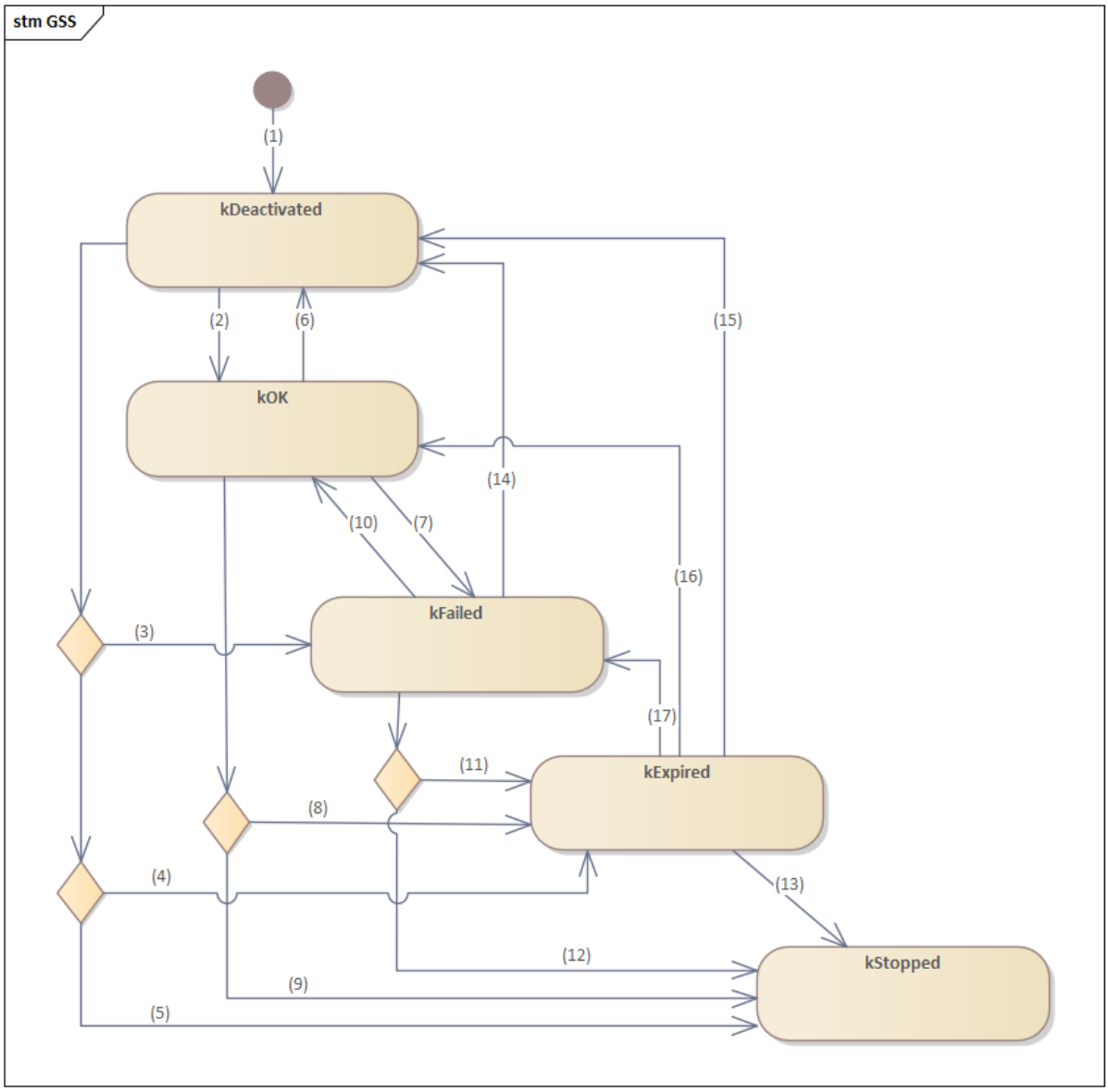


Figure 7.5: Global Supervision Status

[SWS_PHM_00217] One Global Supervision Status per Global Supervision

Upstream requirements: RS_PHM_00111

⌈The `Platform Health Management` shall have one `Global Supervision Status` per `GlobalSupervision` configured.⌋

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] Initialization of Global Supervision Status

Upstream requirements: RS_PHM_00111

⌈The `Global Supervision Status` shall be initialized with `kDeactivated` .⌋

See transition (1) in figure 7.5.

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] Switch Global Supervision Status from kDeactivated to kOK

Upstream requirements: RS_PHM_00111

⌈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` .⌋

See transition (2) in figure 7.5.

[SWS_PHM_00221] Keep Global Supervision Status kOK

Upstream requirements: RS_PHM_00111

⌈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` .⌋

[SWS_PHM_00222] Switch Global Supervision Status to kDeactivated

Upstream requirements: RS_PHM_00111

⌈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.⌋

See transitions (6), (14) and (15) in figure 7.5.

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] Switch Global Supervision Status from kOK to kFailed

Upstream requirements: RS_PHM_00111

⌈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 .⌋

See transition (7) in figure 7.5.

[SWS_PHM_00224] Switch Global Supervision Status from kOK to kExpired for SM/EM/OS supervision

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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.⌋

See transition (8) in figure 7.5.

Note: expiredSupervisionTolerance and 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] Switch Global Supervision Status from kOK to kStopped

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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 .⌋

See transition (9) in figure 7.5.

[SWS_PHM_00226] Keep Global Supervision Status kFailed

Upstream requirements: RS_PHM_00111

⌈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 .⌋

[SWS_PHM_00227] Switch Global Supervision Status from kFailed to kOK

Upstream requirements: RS_PHM_00111

⌈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` .⌋

See transition (10) in figure 7.5.

[SWS_PHM_00228] Switch Global Supervision Status from kFailed to kExpired

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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.⌋

See transition (11) in figure 7.5.

[SWS_PHM_00229] Switch Global Supervision Status from kFailed to kStopped

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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` .⌋

See transition (12) in figure 7.5.

[SWS_PHM_00230] Keep Global Supervision Status kExpired

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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 .⌋

[SWS_PHM_00231] Switch Global Supervision Status from kExpired to kStopped

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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 .⌋

See transition (13) in figure 7.5.

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] Keep Global Supervision Status kStopped

Upstream requirements: RS_PHM_00111

⌈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 .⌋

[SWS_PHM_00233] Switch Global Supervision Status from kExpired to kOK

Upstream requirements: RS_PHM_00111

⌈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 .⌋

See transition (16) in figure 7.5.

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] Switch Global Supervision Status from kExpired to kFailed

Upstream requirements: RS_PHM_00111

⌈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 .⌋

See transition (17) in figure 7.5.

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] Switch Global Supervision Status from kDeactivated to kFailed

Upstream requirements: RS_PHM_00111

⌈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 .⌋

See transition (3) in figure 7.5.

[SWS_PHM_00238] Switch Global Supervision Status from kDeactivated to kExpired

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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.⌋

See transition (4) in figure 7.5.

[SWS_PHM_00239] Switch Global Supervision Status from kDeactivated to kStopped

Upstream requirements: RS_PHM_00111, RS_PHM_00112

⌈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 .⌋

See transition (5) in figure 7.5.

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

7.5 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.

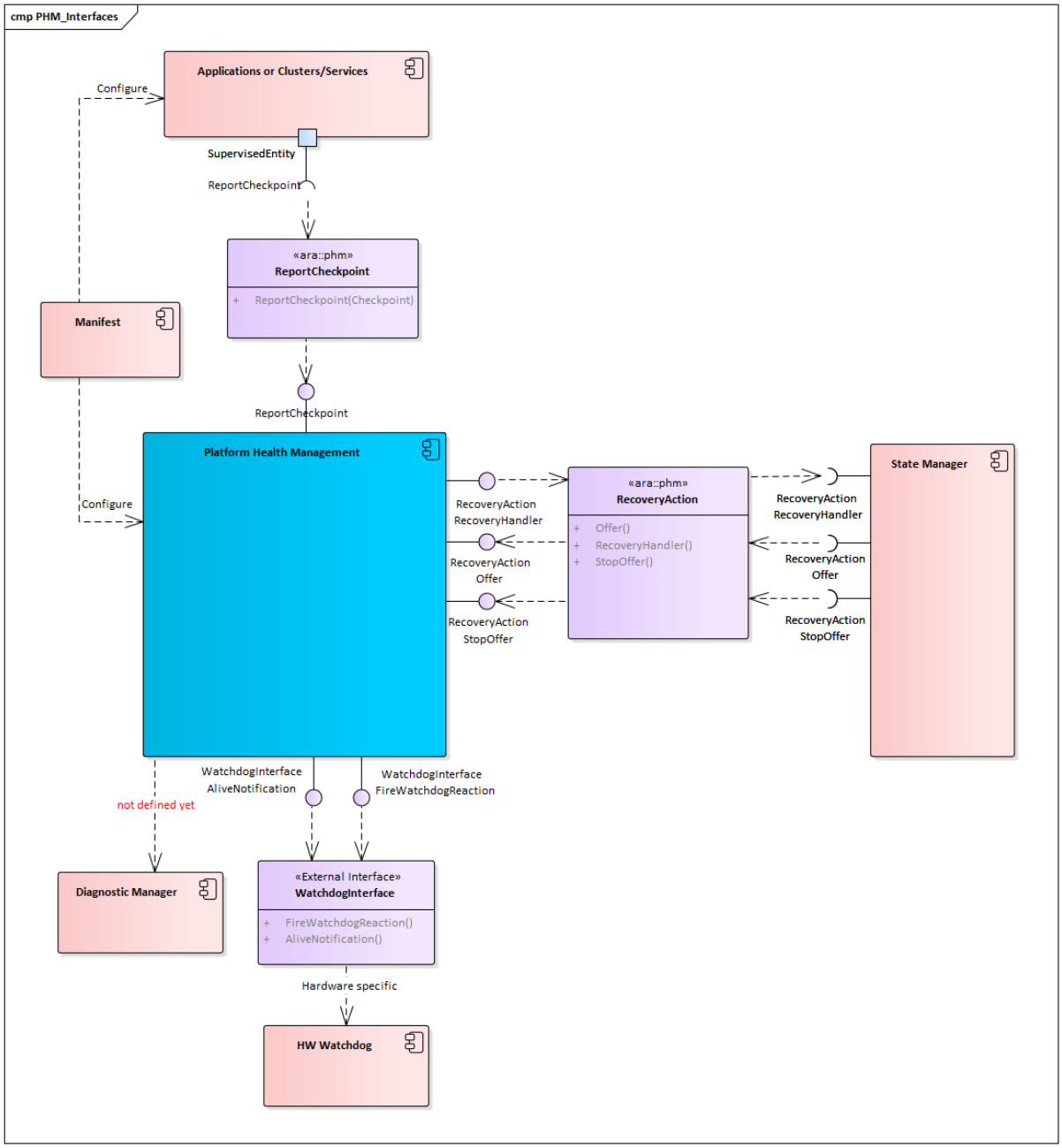


Figure 7.6: Platform Health Management and the environment

7.5.1 Notificaton to State Management

The `Platform Health Management` debounces the failures of `Supervised Entitys`, see the `Elementary Supervision Status` `kFailed` in chapter 7.4. 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. In case a failure cannot be handled, State Management can request a watchdog reaction via corresponding error code to Platform Health Management.

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 ara::phm::RecoveryAction::RecoveryHandler with a configurable timeout. If no response by State Management is received in time, the PHM will do its own countermeasures by wrongly triggering or stop triggering the serviced watchdog.

[SWS_PHM_00101] Notification to State Management due to Supervision failure

Upstream requirements: RS_HM_09249, RS_PHM_00117

⌈If the status of the mapped GlobalSupervision via RecoveryNotificationToPPortPrototypeMapping switches to state kExpired , the Platform Health Management shall notify State Management via the method ara::phm::RecoveryAction::RecoveryHandler. The parameter executionError shall contain the corresponding Function Group and the current ApApplicationError. The parameter supervision shall contain the TypeOfSupervision which causes the transition to state kExpired .⌋

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 ApApplicationError is defined within the StartupConfig, wherefore the executionError depends on the current used StartupConfig.

[SWS_PHM_01361] Default value for ExecutionError

Upstream requirements: RS_PHM_00117

⌈If GlobalSupervision detects a failure of a Process and the Process does not have an executionError configured in the StartupConfig, Platform Health Management shall report the executionError value as 1 to State Management.⌋

[SWS_PHM_00104] Reaction on timeout for notification to State Management

Upstream requirements: RS_HM_09249, RS_HM_09226

⌈The Platform Health Management shall stop calling apext::phm::WatchdogInterface::AliveNotification and call apext::phm::WatchdogInterface::FireWatchdogReaction if

- a failure is detected AND
- a notification of this failure is sent to State Management via the method ara::phm::RecoveryAction::RecoveryHandler AND
- the time between failure detection and reception of an acknowledgment response by State Management is longer than RecoveryNotification.recoveryNotificationTimeout.

⌋

Note 1: Possible reasons that the acknowledgment response is not received within given time interval: `ara::phm::RecoveryAction::RecoveryHandler` is not offered or IPC is not working.

Note 2: If the method `ara::phm::RecoveryAction::RecoveryHandler` returns without an error, no further action is taken.

[SWS_PHM_01363] Production Error reporting on Watchdog Reaction

Upstream requirements: RS_PHM_00115, RS_HM_09244

⌈If `Platform Health Management` calls `apext::phm::WatchdogInterface::FireWatchdogReaction`, it should report FAILED to the `PHM_E_WATCHDOG_RESET` production error.⌋

Note:
The criteria for triggering the watchdog reset ([SWS_PHM_00104]) and for setting this production error `PHM_E_WATCHDOG_RESET` to FAIL are the same.

The watchdog reset can terminate `Processes` so quickly and with high priority that in general the production error can not be persisted to memory.

In order to effectively capture this production error, it may be necessary to evaluate the ECU hardware reset status register following the next ECU startup, and the production error should be logged if the reset was caused by the watchdog.

In addition, the failure causes pertaining to this production error can be stored in the retention RAM of the ECU hardware prior to the watchdog reset, and this information can be accessed during the next startup, along with the hardware reset status register, to report the production error.

Although loosing the DLT message is not desirable, but this is acceptable because logging DLT message is of low priority compared to the pending watchdog reset (this is why the requirement is recommended and not mandatory).

[SWS_PHM_01147] Enable handler

Upstream requirements: RS_PHM_00111, RS_PHM_00117

⌈`Platform Health Management` shall enable potential invocations of `ara::phm::RecoveryAction::RecoveryHandler` when `ara::phm::RecoveryAction::Offer` is called.⌋

[SWS_PHM_01148] Disable handler

Upstream requirements: RS_PHM_00111, RS_PHM_00117

⌈`Platform Health Management` shall disable invocations of `ara::phm::RecoveryAction::RecoveryHandler` when `ara::phm::RecoveryAction::StopOffer` is called.⌋

7.5.2 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 7.5.1 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] Alive Notification to Hardware Watchdog

Upstream requirements: RS_HM_09249, RS_HM_09226, RS_PHM_00114

⌈As long as no Global Supervision Status corresponding to State Management or Execution Management has reached state kStopped , Notification to State Management has not failed and no error code kSMCanNotHandleRecovery was received, Platform Health Management shall call apext::phm::WatchdogInterface::AliveNotification periodically.⌋

[SWS_PHM_00105] Recovery Action for Failures in Execution Management or State Management

Upstream requirements: RS_HM_09249, RS_HM_09226, RS_PHM_00115, RS_PHM_00116, RS_PHM_00114

⌈If the Global Supervision Status corresponding to State Management or Execution Management switches to kStopped , Platform Health Management shall stop calling apext::phm::WatchdogInterface::AliveNotification and call apext::phm::WatchdogInterface::FireWatchdogReaction.⌋

[SWS_PHM_00107] Reaction on a return of kSMCanNotHandleRecovery for notification to State Management

Upstream requirements: RS_HM_09249, RS_HM_09226, RS_PHM_00114

⌈If the method ara::phm::RecoveryAction::RecoveryHandler returns the error kSMCanNotHandleRecovery, the Platform Health Management shall stop calling apext::phm::WatchdogInterface::AliveNotification and call apext::phm::WatchdogInterface::FireWatchdogReaction.⌋

[SWS_PHM_01359] Recovery Action without enabling of recoveryhandler

Upstream requirements: RS_HM_09249, RS_HM_09226, RS_HM_09169, RS_PHM_00115, RS_PHM_00116

⌈If the status of the mapped GlobalSupervision via `RecoveryNotificationToPPortPrototypeMapping` switches to state `kExpired` before the corresponding `ara::phm::RecoveryAction::Offer` is called successfully, `Platform Health Management` shall stop calling `apext::phm::WatchdogInterface::AliveNotification` and call `apext::phm::WatchdogInterface::FireWatchdogReaction`.⌋

[SWS_PHM_01360] Recovery Action after recoveryhandler StopOffer() was called

Upstream requirements: RS_HM_09249, RS_HM_09226, RS_HM_09169, RS_PHM_00115, RS_PHM_00116

⌈If the status of the mapped GlobalSupervision via `RecoveryNotificationToPPortPrototypeMapping` switches to state `kExpired` after the corresponding `ara::phm::RecoveryAction::StopOffer` is called successfully, `Platform Health Management` shall stop calling `apext::phm::WatchdogInterface::AliveNotification` and call `apext::phm::WatchdogInterface::FireWatchdogReaction`.⌋

7.5.3 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 a response by State Management after detection of failure.

7.6 Multiple processes and multiple instances

During the application deployment phase, a single Supervised Entity may be instanciated several times: this happens for example when the same C++ object class representing a Supervised Entity is explicitly instanciated inside the code or when the same executable containing the Supervised Entity 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 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 [13]. The modelling relation of the InstanceSpecifier and its usage in PHM is explained in detail in the chapter "'Supervised Entities and Checkpoints"' in [12].

[SWS_PHM_01383] Multiple Supervised Entities of the same or different type belonging to a process

Upstream requirements: RS_PHM_00106, RS_PHM_00105

⌈Multiple Supervised Entitys of the same or different types i.e. logical, deadline or alive supervision can be created for the same process.⌋

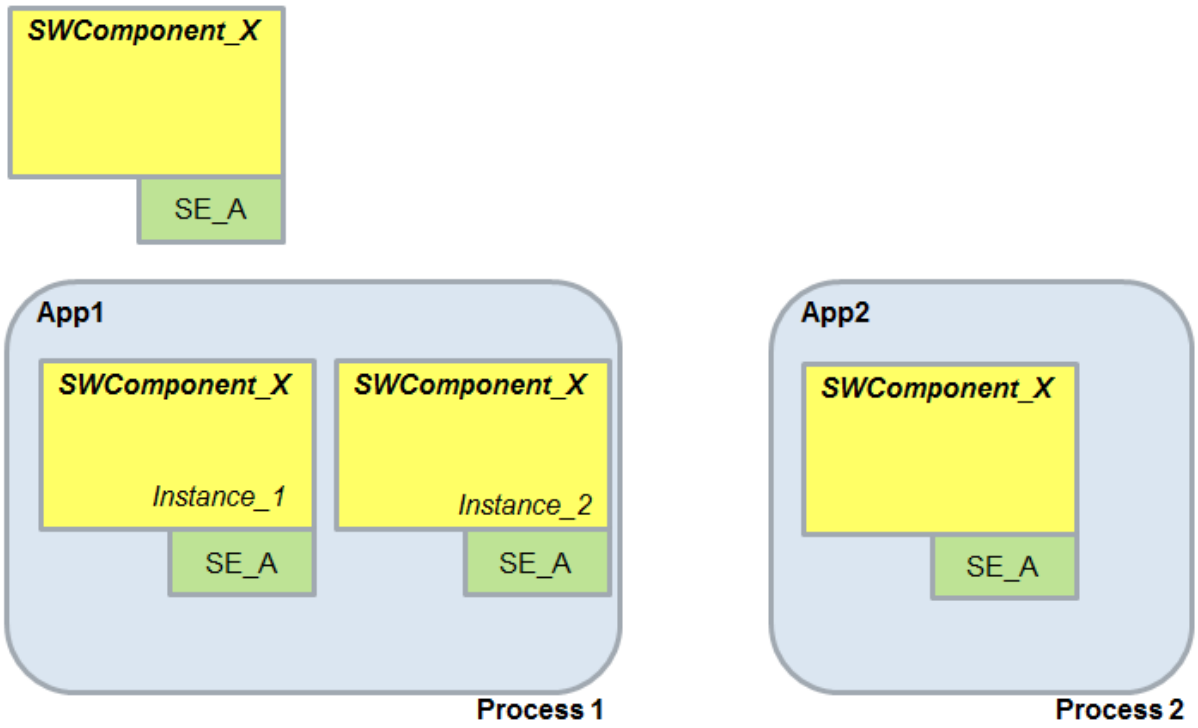


Figure 7.7: Example of multiple instance of the same Supervised Entity

Figure 7.7 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.

7.7 Functional cluster life-cycle

This section defines behavior of this functional cluster during its life-cycle. Please note that there is a general behavior for ara::core::Initialize and ara::core::Deinitialize defined in [13] by [SWS_CORE_15005] and [SWS_CORE_90022].

7.7.1 Startup

[SWS_PHM_01252] Handling of Watchdog after Startup

Upstream requirements: RS_HM_09249, RS_HM_09244, RS_HM_09245, RS_HM_09246, RS_PHM_00114

⌈`Platform Health Management` shall call `apext::phm::WatchdogInterface::AliveNotification` before reporting `kRunning` to Execution Management using the method `ara::exec::ExecutionClient::ReportExecutionState`.⌋

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].

7.7.2 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] Termination of Supervisions at SIGTERM

Upstream requirements: RS_HM_09222, RS_HM_09125, RS_HM_09235

⌈`Platform Health Management` shall stop all configured supervisions (eg: delete all supervision objects) after receiving SIGTERM.⌋

[SWS_PHM_01254] Global Supervision Status at SIGTERM

Upstream requirements: RS_HM_09222, RS_HM_09125, RS_HM_09235

⌈`Platform Health Management` shall change all `Global Supervision Statuses` to `kDeactivated` after receiving SIGTERM.⌋

7.7.2.1 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.

7.8 Reporting

7.8.1 Security Events

This section lists all security events defined by this functional cluster.

[SWS_PHM_01340] Security events for ap_phm ⌈

Name	Description	ID
SEV_ACCESS_CONTROL_PHM_IAM_ACCESS_DENIED	Access of an application to a resource provided by Platform Health Management was denied.	65

⌋

[SWS_PHM_01375] Security event context data definition: SEV_ACCESS_CONTROL_PHM_IAM_ACCESS_DENIED ⌈

SEV Name	SEV_ACCESS_CONTROL_PHM_IAM_ACCESS_DENIED	
ID	65	
Description	Access of an application to a resource provided by Platform Health Management was denied.	
Context Data Version	1	
Context Data	Data Type	Allowed Values
UserId	uint32	

⌋

7.8.2 Log Messages

[SWS_PHM_01371] Log Message SupervisionStateChanged

Status: DRAFT
Upstream requirements: RS_AP_00156

⌈

Dlt-Message	SupervisionStateChanged		
Description	Message sent by Platform Health Management when the Execution state of a process changes. The well defined states of a process supervision determine if process is running ok, if a supervision failure is detected or supervision ends.		
MessageId	0x80009001		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	ArgumentDescription	ArgumentType	ArgumentUnit
executionStateOf Process	Execution state of Process.	uint8 [encoding UTF-8]	
modeledProcess Id	Meta-model identifier of the process that has been started, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
hasChanged From	has changed from	uint8 [encoding UTF-8]	

▽

△

prevExecution State	Execution state	uint32	
to	to	uint8 [encoding UTF-8]	
currExecution State	Execution state	uint32	

⌋

[SWS_PHM_01372] Log Message RecoveryInitiated

Status: DRAFT
Upstream requirements: RS_AP_00156

⌈

Dlt-Message	RecoveryInitiated		
Description	Message sent by Platform Health Management when the Global Supervision Status has reached status “EXPIRED“ and the RecoveryHandler API invoked State Management.		
MessageId	0x80009002		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
phmInitiated RecoveryAction	Platform Health Management has initiated Recovery Action due to supervision failure of Process	uint8 [encoding UTF-8]	
modeledProcess Id	Meta-model identifier of the process that has been started, i.e., its short name path with ’/’ as a separator.	uint8 [encoding UTF-8]	

⌋

[SWS_PHM_01373] Log Message WatchdogReset

Status: DRAFT
Upstream requirements: RS_AP_00156

⌈

Dlt-Message	WatchdogReset		
Description	Message sent by Platform Health Management when the Platform Health Management initiates a watchdog reset		
MessageId	0x80009003		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
phmRequested Watchdog	Platform Health Management has initiated watchdog reset due to supervision failure of Process	uint8 [encoding UTF-8]	
modeledProcess Id	Meta-model identifier of the process that has been started, i.e., its short name path with ’/’ as a separator.	uint8 [encoding UTF-8]	

⌋

[SWS_PHM_01370] Log Message SupervisionStarted

Status: DRAFT
Upstream requirements: RS_AP_00156

⌈

Dlt-Message	SupervisionStarted		
Description	Message sent by Platform Health Management immediately after successfully starting supervision of this process.		
MessageId	0x80009000		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
phmSupervision Started	Platform Health Management has started supervision of Process.	uint8 [encoding UTF-8]	
modeledProcess Id	Meta-model identifier of the process that has been started, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	

⌋

7.8.3 Violation Messages

This section lists all violation messages (i.e., DLT messages logged for Violations according to [AP_SWS_GENERAL_00021]) defined by this functional cluster.
Please note that concrete implementations might additionally implement Non-Standardized Violations (see also [AP_SWS_GENERAL_00003]).

[AP_SWS_GENERAL_13006] Violation Message InstanceSpecifierAlreadyInUse Violation

Dlt-Message	InstanceSpecifierAlreadyInUseViolation		
Description	Violation message that is sent in case a constructor in the ara framework was called with an Instance Specifier already in use in this process.		
MessageId	0x80001ff9		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcess Id	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

[AP_SWS_GENERAL_13007] Violation Message AraNotInitializedViolation

Dlt-Message	AraNotInitializedViolation		
Description	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.		
MessageId	0x80001ff8		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object. (only present in case this violation is created by a function/constructor that has an InstanceSpecifier as an argument)	uint8 [encoding UTF-8]	
functionName	Name of the function/constructor that was called that checks for an initialized ara.	uint8 [encoding UTF-8]	

[AP_SWS_GENERAL_13215] Violation Message UnresolvableInstanceSpecifier Violation

Dlt-Message	UnresolvableInstanceSpecifierViolation		
Description	InstanceSpecifier either cannot be resolved in the model in the context of your executable or process, or it refers to a model element that is neither a PortPrototype of the expected type nor a FunctionalCluster InteractsWithFunctionalClusterMapping of the expected type.		
MessageId	0x80001fe8		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {Class Name}:{MethodName}, {filename}:{linenumber}, etc.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

[SWS_PHM_01381] Violation Message DaemonUnreachableViolation

Status: DRAFT
Upstream requirements: RS_AP_00156

⌈

Dlt-Message	DaemonUnreachableViolation
Description	This Violation indicates that communication between a PHM client and the PHM daemon cannot be established during initialization or can no longer be maintained during runtime.
MessageId	0x80009fff

▽

△

MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	

⌋

7.8.4 Production Errors

This section lists all production errors (i.e., Diagnostic Events) defined by this functional cluster.

7.8.4.1 PHM_E_WATCHDOG_RESET

[SWS_PHM_01362] Diagnostic events for Platform Health Management

Status: DRAFT
Upstream requirements: RS_PHM_00115, RS_HM_09244

⌈

Diagnostic Event (Error Name)	PHM_E_WATCHDOG_RESET
Description	PHM has triggered a watchdog reset.
Monitoring condition	Continuously
Failed condition	A supervision failure happened which actively triggered a watchdog reset.
Passed condition	A clean machine start which is not caused by a watchdog reset.

⌋

7.8.5 Lost Daemon Connection

A lost connection to the `Health Monitoring` daemon is treated as `DaemonUnreachableViolation`. `DaemonUnreachableViolation` indicates that communication between a PHM client and the PHM daemon cannot be established during initialization or can no longer be maintained during runtime. The rationale is that the use of `Health Monitoring` typically implies a safety case in which the application should not start, or continue operation, if the daemon is not present or not available. This is also consistent with the dependency of Execution Management on `Health Monitoring`. If Execution Management cannot connect to the PHM daemon during startup, startup of the overall system is not meaningful because the safety-relevant supervision is not available.

If the connection to the `Health Monitoring` daemon cannot be established during construction of a PHM client object, this is to be considered `DaemonUnreachableViolation` and the application should not start.

If an application loses the connection to the `Health Monitoring` daemon at runtime, for example while calling `ara::phm::SupervisedEntity::ReportCheckpoint`, this is also to be considered `DaemonUnreachableViolation`. In this case, the application aborts immediately after it detects that communication with the daemon is no longer possible.

If the supervised entities are running correctly and `Health Monitoring` crashes, the alive keeping of the watchdog will end and there will be a watchdog reset.

8 API specification

This chapter provides a reference of the APIs defined by this functional cluster. The API is described in the following chapters in tables. Table 8.1 explains the content that is described in such an API table.

Kind:	Defines the kind of the declaration that this API table describes. The following values are supported: • class (Declaration of a class) • function (Declaration of a member or non-member function) • struct (Declaration of a structure) • type alias (Declaration of a type alias) • enumeration (Declaration of an enumeration) • variable (Declaration of a variable)	
Port Interfaces:	States that the C++ API `class` is the related C++ API binding for the given modeled sub-class of `PortInterface`	
Header File:	Defines the header file to be included according to [AP_SWS_GENERAL_90001]	
Forwarding Header File:	Defines the forwarding header file to be included according to [AP_SWS_GENERAL_90001]	
Scope:	Defines the scope that may be a C++ `namespace` (in case of a `class` or non-member function) or a `class` declaration (in case of a member)	
Symbol:	C++ symbol name	
Thread Safety:	Defines whether a function is thread-safe, not thread-safe, or conditional according to [AP_SWS_GENERAL_13200] and [AP_SWS_GENERAL_13202]	
Syntax:	Description of C++ syntax	
Template Param:	Template parameter (0..*)	Template parameter(s) used to parameterize the template
Parameters (in):	Parameter declaration (0..*)	Parameter(s) that are passed to the function
Parameters (out):	Parameter declaration (0..*)	Parameter(s) that are returned to the caller
Return Value:	Return type	Type of the value that the function returns
Exception Safety:	Defines whether a function is exception-safe, not exception safe or conditionally exception safe	
Exceptions:	List of C++ `Exception`s that may be thrown by the function	
Violations:	List of violations that may raised by the function	
Errors:	Error type (0..*)	List of defined `ara::core::ErrorCode`s that may be returned by the function with their recoverability class defined in [RS_AP_00160]. APIs can be extended with vendor-specific error codes. These are not standardized by AUTOSAR
Description:	Brief description of the function	

Table 8.1: Explanation of an API table

8.1 PortInterface to API class binding

This table shows the `API class` binding for each `PortInterface` owned by this functional cluster and those functions taking an `ara::core::InstanceSpecifier` argument, designated to “construct” that class. These constructing functions may be any combination of special-member constructors, named constructor members or non-member factory constructors.

Port Interface	API Class / Function
PhmSupervisedEntityInterface	[SWS_PHM_01132] Definition of API class ara::phm::SupervisedEntity
	[SWS_PHM_01123] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity
PhmSupervisionRecoveryNotification-Interface	[SWS_PHM_01140] Definition of API class ara::phm::RecoveryAction
	[SWS_PHM_01141] Definition of API function ara::phm::RecoveryAction::RecoveryAction

Table 8.2: PortInterface (sub-class) to API class / function binding

8.2 Header: ara/phm/supervised_entities/{<si-namespace-derived-directory-path-lower>}/{<phmssi-sn>}.h

[SWS_PHM_01002] File name, includes and multiple inclusion guard ⌈

Kind:	Header File	
Syntax:	ara/phm/supervised_entities/{<si-namespace-derived-directory-path-lower>}/{<phmssi-sn>}.h	
Description:	For each modeled PhmSupervisedEntityInterface a header file shall be generated according to this directory and path/file name convention - a multiple inclusion guard shall be placed around the whole header file as per [AP_SWS_GENERAL_90002].	
Descriptors:	{<si-namespace-derived-directory-path-lower>}	as per [SWS_PHM_01005] whereby: for each inner namespace in the hierarchy, an inner directory shall be created to contain the header file
	{<phmssi-sn>}	The file name as given by PhmSupervisedEntityInterface.shortName converted to lower-case.
Example:		

```
// File=ara/phm/supervised_entities/n/n_p_1/n_p_2/si_checkpoint.h   (1)
#ifndef N_NPLUS1_NPLUS2_SI_CHECKPOINT_H_      (2)
#define N_NPLUS1_NPLUS2_SI_CHECKPOINT_H_      (2)

...
#endif // N_NPLUS1_NPLUS2_SI_CHECKPOINT_H_    (2)
```

⌋

8.2.1 Namespaces

8.2.1.1 Namespace: ara::phm::supervised_entities::{<hierarchical-namespace-list-lower-skeleton>}

[SWS_PHM_01005] Checkpoint Header File: service namespace

Upstream requirements: RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "ara/phm/supervised_ entities/{<si-namespace-derived-directory-path-lower>}/{<phmssi-sn>}.h"
Scope:	namespace ara::phm::supervised_entities
Syntax:	`namespace {<hierarchical-namespace-list-lower-skeleton>}`
Description:	The generator shall use the `SymbolProps` aggregated in the role `PortInterface`. `namespace`. For each `namespace` in the **ordered** list: `namespace`[N+1] shall be an inner namespace of `namespace`[N] converted to lower-case.

⌋

8.2.2 Non-Member Types

[SWS_PHM_00424] Definition of API enum ara::phm::supervised_entities::{<hierarchical-namespace-list-lower-skeleton>}::{<phmssi-sn>}

Upstream requirements: RS_PHM_00101, RS_PHM_09241, RS_AP_00130, RS_AP_00122, RS_AP_00127

⌈

Kind:	enumeration	
Header file:	#include "ara/phm/supervised_ entities/{<si-namespace-derived-directory-path-lower>}/{<phmssi-sn>}.h"	
Forwarding header file:	#include "ara/phm/phm_fwd.h"	
Scope:	namespace ara::phm::supervised_ entities::{<hierarchical-namespace-list-lower-skeleton>}	
Symbol:	{<phmssi-sn>}	
Underlying type:	std::uint32_t	
Syntax:	`enum class {<phmssi-sn>} : std::uint32_t {...};`	
Values:	{<phm-checkpoint-list>}	--
Description:	Defines the checkpoints for the ara::phm::PhmSupervisedEntityInterface	
Descriptors:	{<phm-checkpoint-list>}	*Shown as "..." in Syntax*. The list of enumerations (checkpoints) for the `PhmSupervisedEntityInterface`. For each checkpoint in {<phm-checkpoint-list>}, [SWS_PHM_00425] shall be applied.

⌋

8.2.3 Global Variables

[SWS_PHM_00425] Definition of API variable ara::phm::supervised_entities::{<hierarchical-namespace-list-lower-skeleton>}::{<symbol-phm-checkpoint>}

Upstream requirements: RS_PHM_00101, RS_PHM_09241

⌈

Kind:	variable	
Header file:	#include "ara/phm/supervised_entities/{<si-namespace-derived-directory-path-lower>}/{<phmssi-sn>}.h"	
Scope:	namespace ara::phm::supervised_entities::{<hierarchical-namespace-list-lower-skeleton>}	
Symbol:	{<symbol-phm-checkpoint>}	
Type:	--	
Syntax:	{<symbol-phm-checkpoint>} = {<phm-checkpoint-value>};	
Description:	For each enumeration in {<phm-checkpoint-list>} in [SWS_PHM_00424] there shall exist a C++ enumerator declaration.	
Descriptors:	{<symbol-phm-checkpoint>}	The checkpoint enumerator symbol name as given by PhmCheckpoint. shortName.
	{<phm-checkpoint-value>}	The checkpoint enumerator value as given by PhmCheckpoint. checkpointId.

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

⌋

8.3 Header: ara/phm/phm_error_domain.h

8.3.1 Non-Member Types

[SWS_PHM_01240] Definition of API enum ara::phm::PhmErrc

Upstream requirements: RS_AP_00119

⌈

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
	kSMCanNotHandle Recovery	= 3
		State Management cannot handle the recovery and PlatformHealth Management will take over by firing the watchdog.
	kNotInSTRMode	= 4
		Disable request was issued while the STR mode is not active.
	kServiceNotReady	= 5
		There is already an ongoing Enable or Disable request.
Description:	Defines an enumeration class for the Platform Health Management error codes.	

⌋

8.3.2 Non-Member Functions

8.3.2.1 Other

[SWS_PHM_01251] Definition of API function ara::phm::GetPhmDomain

Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

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

⌋

[SWS_PHM_01244] Definition of API function ara::phm::MakeErrorCode

Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	namespace ara::phm	
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:	exception safe
Thread Safety:	thread-safe
Description:	Creates an error code.

⌋

8.3.3 Class: PhmErrorDomain

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

Upstream requirements: RS_AP_00119

⌈

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:	As per `ara::phm::PhmErrorDomain` in [AP_SWS_GENERAL_90023]
Description:	Defines the error domain for Platform Health Management.

⌋

8.3.3.1 Public Member Types

[SWS_PHM_01245] Definition of API type ara::phm::PhmErrorDomain::Errc

Upstream requirements: RS_AP_00119, RS_AP_00127

⌈

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.

⌋

[SWS_PHM_01246] Definition of API type ara::phm::PhmErrorDomain::Exception

Upstream requirements: RS_AP_00119, RS_AP_00127

⌈

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.

⌋

8.3.3.2 Public Member Functions

8.3.3.2.1 Special Member Functions

[SWS_PHM_01247] Definition of API function ara::phm::PhmErrorDomain::PhmErrorDomain

Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/phm/phm_error_domain.h"
Scope:	`class ara::phm::PhmErrorDomain`
Syntax:	`PhmErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Creates a PhmErrorDomain instance.

⌋

8.3.3.2.2 Member Functions

[SWS_PHM_01249] Definition of API function ara::phm::PhmErrorDomain::Message

Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/phm/phm_error_domain.h"
Scope:	`class ara::phm::PhmErrorDomain`

▽

△

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:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the message associated with the error code.	

⌋

[SWS_PHM_01248] Definition of API function ara::phm::PhmErrorDomain::Name

Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	`class ara::phm::PhmErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	As per `ara::phm::PhmErrorDomain` in [AP_SWS_GENERAL_90023]
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the name of the error domain.	

⌋

[SWS_PHM_01250] Definition of API function ara::phm::PhmErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00119

⌈

Kind:	function	
Header file:	#include "ara/phm/phm_error_domain.h"	
Scope:	`class ara::phm::PhmErrorDomain`	
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:	thread-safe	
Description:	Throws the exception associated with the error code. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.3.4 Class: PhmException

[SWS_PHM_01242] Definition of API class ara::phm::PhmException

Upstream requirements: RS_AP_00119

⌈

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.

⌋

8.3.4.1 Public Member Functions

8.3.4.1.1 Constructors

[SWS_PHM_01243] Definition of API function ara::phm::PhmException::PhmException

Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

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

⌋

8.4 Header: ara/phm/recovery_action.h

8.4.1 Non-Member Types

[SWS_PHM_01138] Definition of API enum ara::phm::TypeOfSupervision

Upstream requirements: RS_HM_09159

⌈

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.	

⌋

8.4.2 Class: RecoveryAction

[SWS_PHM_01140] Definition of API class ara::phm::RecoveryAction

Upstream requirements: RS_HM_09159

⌈

Kind:	class
Port Interfaces:	`PhmSupervisionRecoveryNotificationInterface`
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.

⌋

8.4.2.1 Public Member Functions

8.4.2.1.1 Special Member Functions

[SWS_PHM_01150] Definition of API function ara::phm::RecoveryAction::RecoveryAction

Upstream requirements: RS_HM_09159

⌈

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

⌋

[SWS_PHM_01149] Definition of API function ara::phm::RecoveryAction::RecoveryAction

Upstream requirements: RS_HM_09159, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/phm/recovery_action.h"
Scope:	`class ara::phm::RecoveryAction`
Syntax:	`RecoveryAction (RecoveryAction &&ra)=delete;`
Description:	The move constructor operator for the RecoveryAction shall not be used. Rationale: Using the move constructor after a RecoveryAction instance has been offered may result in premature invocation of RecoveryHandler before the derived class is fully constructed. This can lead to undefined behavior or unhandled exceptions, compromising system stability and safety

⌋

[SWS_PHM_01152] Definition of API function ara::phm::RecoveryAction::operator=

Upstream requirements: RS_HM_09159

⌈

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

⌋

[SWS_PHM_01151] Definition of API function ara::phm::RecoveryAction::operator=

Upstream requirements: RS_HM_09159, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/phm/recovery_action.h"
Scope:	`class ara::phm::RecoveryAction`
Syntax:	`RecoveryAction & operator= (RecoveryAction &&ra)=delete;`
Description:	The move assignment operator for the RecoveryAction shall not be used. Rationale: Using the move assignment after a RecoveryAction instance has been offered may result in premature invocation of RecoveryHandler before the derived class is fully constructed. This can lead to undefined behavior or unhandled exceptions, compromising system stability and safety.

⌋

[SWS_PHM_01145] Definition of API function ara::phm::RecoveryAction::~RecoveryAction

Upstream requirements: RS_HM_09159, RS_AP_00134

⌈

Kind:	function
Header file:	#include "ara/phm/recovery_action.h"
Scope:	`class ara::phm::RecoveryAction`
Syntax:	`virtual ~RecoveryAction () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for RecoveryAction.

⌋

8.4.2.1.2 Constructors

[SWS_PHM_01141] Definition of API function ara::phm::RecoveryAction::RecoveryAction

Upstream requirements: RS_HM_09159, RS_AP_00159, RS_AP_00173, RS_AP_00180

⌈

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

▽

△

Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`InstanceSpecifierAlreadyInUseViolation`	Violation message that is sent in case a constructor in the ara framework was called with an InstanceSpecifier already in use in this process.
	`UnresolvableInstanceSpecifierViolation`	The instance specifier can't be resolved. It is not available in the context of the executable, refers to a model element other than a `PortPrototype`, or the `PortPrototype` is not typed `PhmSupervisionRecoveryNotificationInterface` or no matching process is found for the (runtime) process.
	`DaemonUnreachableViolation`	Communication with the PHM Daemon could not be established.
Description:	Creation of an RecoveryAction.	

⌋

8.4.2.1.3 Member Functions

[SWS_PHM_01143] Definition of API function ara::phm::RecoveryAction::Offer

Upstream requirements: RS_HM_09159, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/recovery_action.h"	
Scope:	`class ara::phm::RecoveryAction`	
Syntax:	`ara::core::Result< void > Offer () noexcept;`	
Return value:	ara::core::Result< void >	A Result, being either empty or containing any of the errors defined below.
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	PhmErrc::kOfferFailed	rollback_semantics
		Service could not be offered due to failure of communication with Phm daemon
Description:	Enables potential invocations of RecoveryHandler.	

⌋

[SWS_PHM_01142] Definition of API function ara::phm::RecoveryAction::RecoveryHandler

Upstream requirements: RS_HM_09159, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/phm/recovery_action.h"
Scope:	`class ara::phm::RecoveryAction`

▽

△

Syntax:	`virtual ara::core::Future< void > RecoveryHandler (const ara::exec::ExecutionErrorEvent &executionError, TypeOfSupervision supervision) noexcept=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:	ara::core::Future< void >	void if recovery is successful, otherwise it returns kSMCanNot HandleRecovery error
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	PhmErrc::kSMCanNot HandleRecovery	rollback_semantics
		State Management cannot handle the recovery and PlatformHealth Management will take over by firing the watchdog.
Description:	RecoveryHandler to be invoked by PHM. The handler invocation needs to be enabled before by a call of RecoveryAction::Offer.	

⌋

[SWS_PHM_01144] Definition of API function ara::phm::RecoveryAction::StopOffer

Upstream requirements: RS_HM_09159, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/phm/recovery_action.h"
Scope:	`class ara::phm::RecoveryAction`
Syntax:	`void StopOffer () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Disables invocations of RecoveryHandler.

⌋

8.5 Header: ara/phm/supervised_entity.h

8.5.1 Non-Member Types

[SWS_PHM_01358] Definition of API enum ara::phm::ElementarySupervisionStatus

Upstream requirements: RS_HM_09237

⌈

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.	

⌋

[SWS_PHM_01137] Definition of API enum ara::phm::GlobalSupervisionStatus

Upstream requirements: RS_HM_09237

⌈

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.	

⌋

8.5.2 Class: SupervisedEntity

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

Upstream requirements: RS_PHM_00101

⌈

Kind:	class	
Port Interfaces:	`PhmSupervisedEntityInterface`	
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 final {...};`	
Template param:	typename EnumT	An enum type that contains a list of checkpoint identifier
Description:	SupervisedEntity Class.	

⌋

8.5.2.1 Public Member Functions

8.5.2.1.1 Special Member Functions

[SWS_PHM_01212] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity

Upstream requirements: RS_PHM_00101, RS_PHM_09240

⌈

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

⌋

[SWS_PHM_01214] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity

Upstream requirements: RS_PHM_00101, RS_PHM_09240, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	`class ara::phm::SupervisedEntity`	
Syntax:	`SupervisedEntity (SupervisedEntity &&se) noexcept;`	
Parameters (in):	se	The SupervisedEntity object to be moved.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Move constructor for SupervisedEntity.	

⌋

[SWS_PHM_01215] Definition of API function ara::phm::SupervisedEntity::operator=

Upstream requirements: RS_PHM_00101, RS_PHM_09240, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	`class ara::phm::SupervisedEntity`	
Syntax:	`SupervisedEntity & operator= (SupervisedEntity &&se) noexcept;`	
Parameters (in):	se	The SupervisedEntity object to be moved.
Return value:	SupervisedEntity &	The moved SupervisedEntity object.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	

▽

△

Description:	Move assignment operator for SupervisedEntity.

⌋

[SWS_PHM_01213] Definition of API function ara::phm::SupervisedEntity::operator=

Upstream requirements: RS_PHM_00101, RS_PHM_09240

⌈

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

⌋

[SWS_PHM_01211] Definition of API function ara::phm::SupervisedEntity::~SupervisedEntity

Upstream requirements: RS_PHM_00101, RS_PHM_09240, RS_AP_00134

⌈

Kind:	function
Header file:	#include "ara/phm/supervised_entity.h"
Scope:	`class ara::phm::SupervisedEntity`
Syntax:	`~SupervisedEntity () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor of a SupervisedEntity.

⌋

8.5.2.1.2 Constructors

[SWS_PHM_01123] Definition of API function ara::phm::SupervisedEntity::SupervisedEntity

Upstream requirements: RS_PHM_00101, RS_AP_00159, RS_AP_00173, RS_AP_00177, RS_AP_00180

⌈

Kind:	function
Header file:	#include "ara/phm/supervised_entity.h"
Scope:	`class ara::phm::SupervisedEntity`

▽

△

Syntax:	`explicit SupervisedEntity (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	instance specifier of the supervised entity.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`AraNotInitialized-Violation`	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.
	`InstanceSpecifier-AlreadyInUse-Violation`	Violation message that is sent in case a constructor in the ara framework was called with an InstanceSpecifier already in use in this process.
	`Unresolvable-InstanceSpecifier-Violation`	It is not available in the context of the executable, refers to a model element other than a `PortPrototype`, or the `PortPrototype` is not typed `PhmSupervisedEntityInterface`, is not referenced by a `SupervisionCheckpoint` or none of the corresponding `SupervisionCheckpoint` references includes a reference to the current proces.
	`DaemonUnreachable-Violation`	Communication with the PHM Daemon could not be established.
Description:	Creation of a SupervisedEntity.	

⌋

8.5.2.1.3 Member Functions

[SWS_PHM_01380] Definition of API function ara::phm::SupervisedEntity::Disable

Status: DRAFT
Upstream requirements: RS_PHM_00101, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	`class ara::phm::SupervisedEntity`	
Syntax:	`ara::core::Future< void > Disable () noexcept;`	
Return value:	ara::core::Future< void >	void if successful, otherwise an error
Exception Safety:	exception safe	
Thread Safety:	not-threadsafe	
Errors:	PhmErrc::kNotIn STRMode	rollback_semantics
		Disable request was issued while the STR mode is not active.
	PhmErrc::kServiceNot Ready	rollback_semantics
		There is already an ongoing Enable or Disable request.
Description:	Disable supervision until Enable() is called. Disable request will only be processed if the STR-Mode is activated i.e. PHM has received the STR notification.	

⌋

[SWS_PHM_01130] Definition of API function ara::phm::SupervisedEntity::Enable

Status: DRAFT
Upstream requirements: RS_PHM_00101, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	`class ara::phm::SupervisedEntity`	
Syntax:	`ara::core::Future< void > Enable () noexcept;`	
Return value:	ara::core::Future< void >	void if successful, otherwise an error
Exception Safety:	exception safe	
Thread Safety:	not-threadsafe	
Errors:	PhmErrc::kServiceNot Ready	rollback_semantics
		There is already an ongoing Enable or Disable request.
Description:	Enable supervision, if possible in the current function group state.	

⌋

[SWS_PHM_01127] Definition of API function ara::phm::SupervisedEntity::ReportCheckpoint

Upstream requirements: RS_PHM_00101, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/phm/supervised_entity.h"	
Scope:	`class ara::phm::SupervisedEntity`	
Syntax:	`void ReportCheckpoint (EnumT checkpointId) noexcept;`	
Parameters (in):	checkpointId	checkpoint identifier.
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`InsufficientPermissionsViolation`	ReportCheckpoint was invoked by a process which is not modelled by the corresponding `SupervisionCheckpoint.process` reference.
	`DaemonUnreachableViolation`	Communication with the PHM Daemon could not be established.
Description:	Reports an occurrence of a Checkpoint.	

⌋

8.6 Sequence Diagrams

8.6.1 Suspend-to-RAM activation

This sequence diagram shows the activation path where applications enter `Suspend-To-RAM (S3)` after `ara::sm::s2r::S2RSatellite::S2RSatellite` creates

the S2RSatellite and `ara::sm::s2r::S2RSatellite::EnterSuspendMode` is invoked, PHM disables supervision, and State Management completes STR entry.

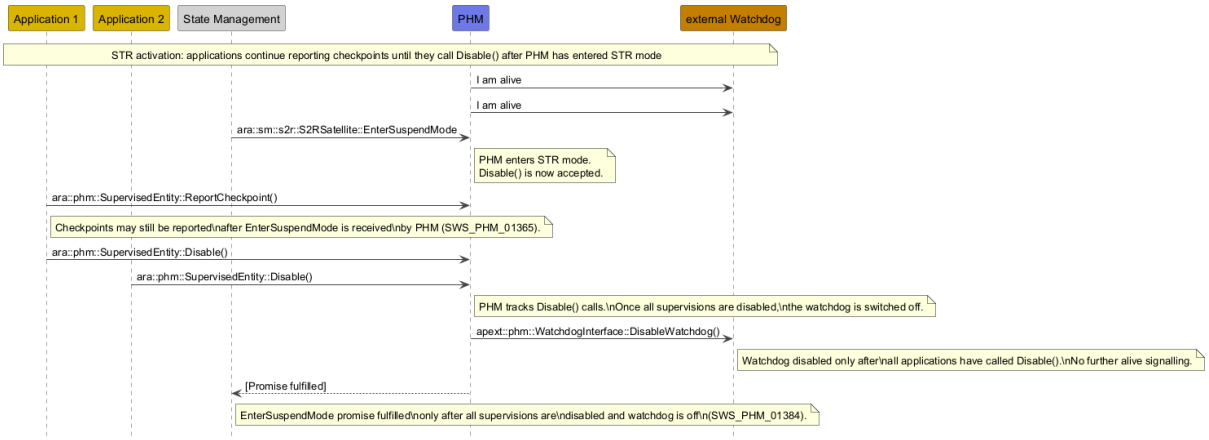


Figure 8.1: Suspend-to-RAM activation sequence

8.6.2 Disable supervision not allowed

This sequence diagram shows the rejection path when Disable() is invoked while PHM is not yet in STR mode.

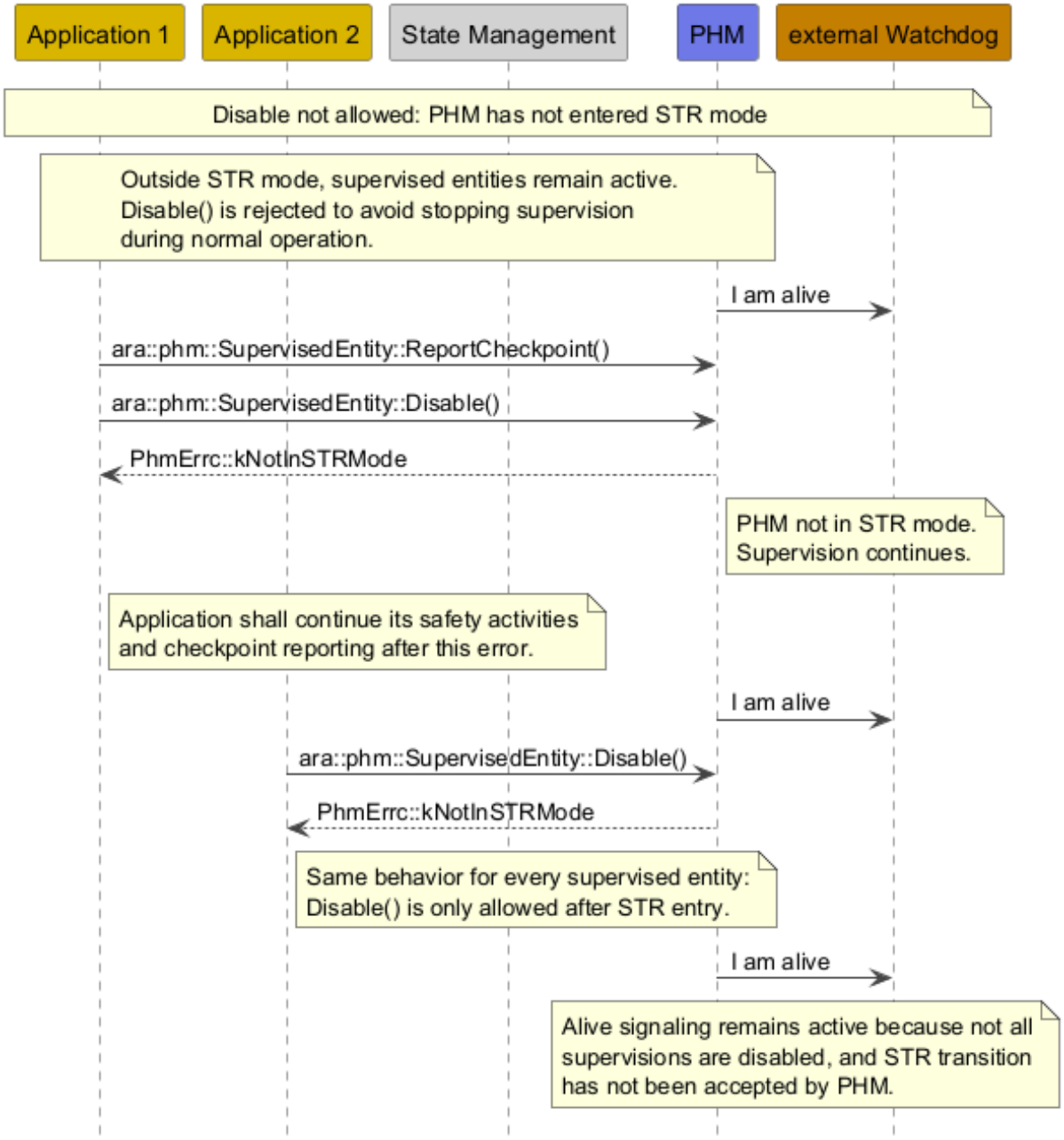


Figure 8.2: Disable supervision request rejected outside STR mode

8.6.3 Resume STR

This sequence diagram shows the resume path where `ara::sm::s2r::S2RSatellite::LeaveSuspendMode` is invoked, and PHM re-enables supervision and the watchdog after State Management leaves STR mode.

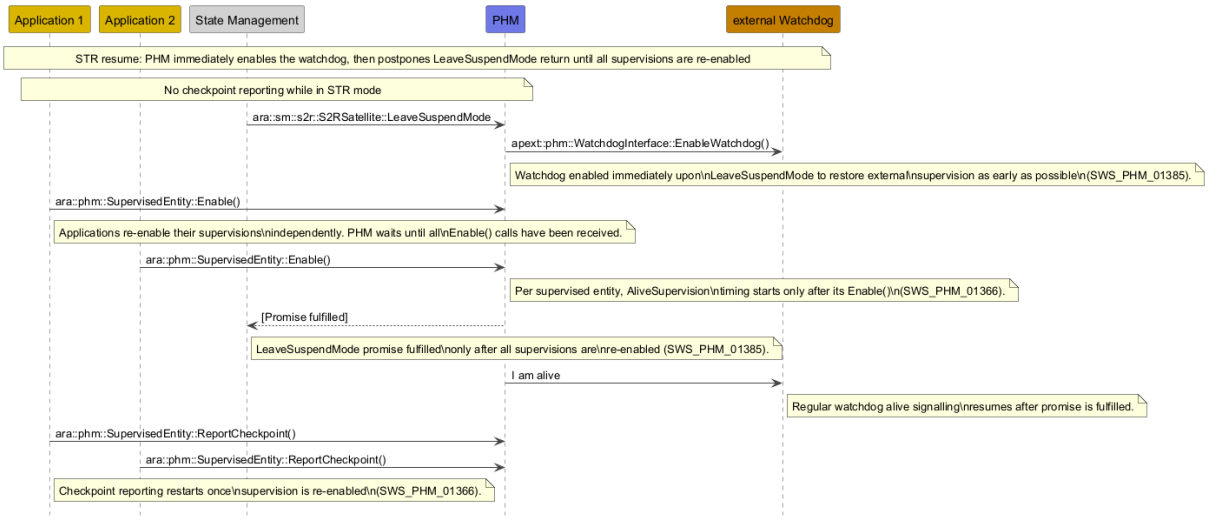


Figure 8.3: Suspend-to-RAM resume sequence

8.6.4 Checkpoint ignored after Disable

This sequence diagram shows that, once `ara::phm::SupervisedEntity::Disable` has been accepted for a supervised entity, subsequent `ara::phm::SupervisedEntity::ReportCheckpoint` calls of that entity are ignored until supervision is enabled again. It also highlights that the application shall stop safety activities before invoking Disable().

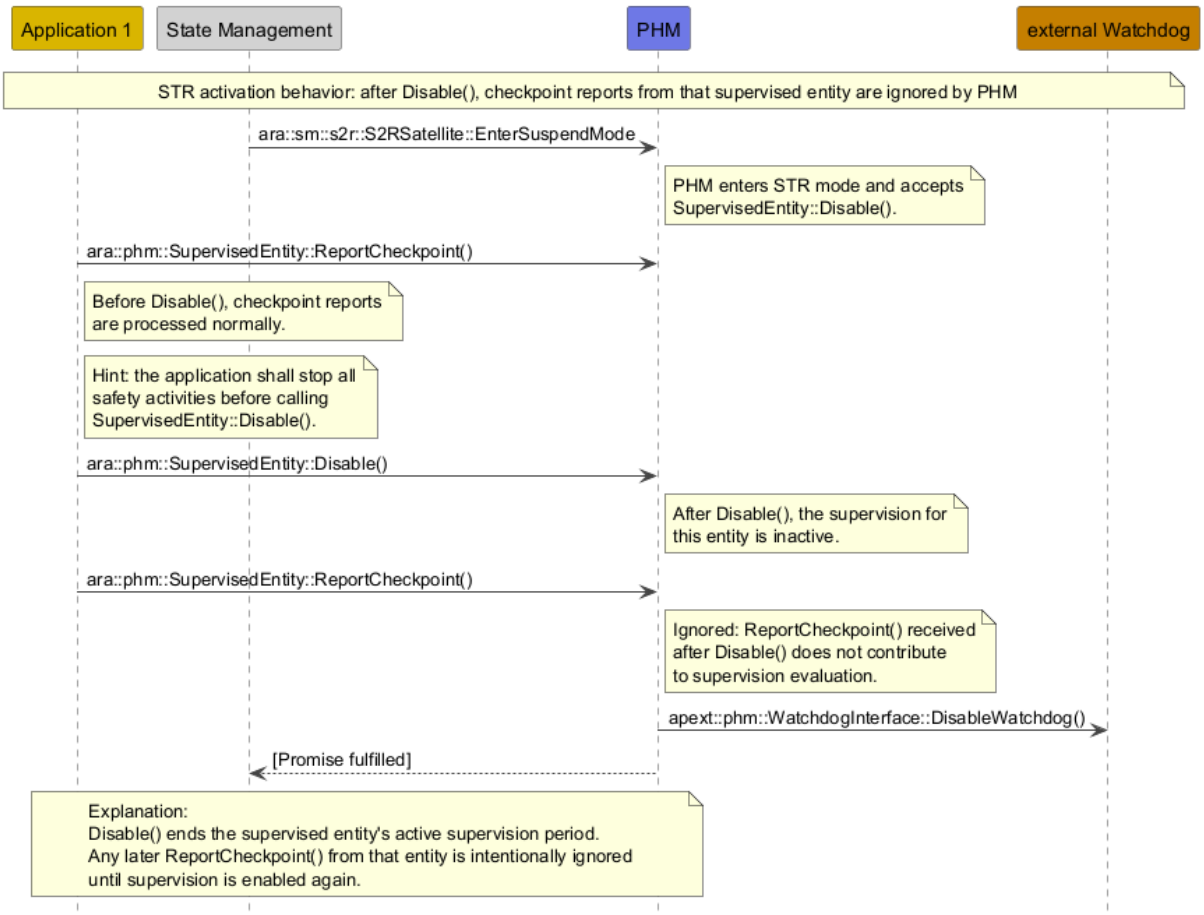


Figure 8.4: ReportCheckpoint ignored after Disable()

9 Service Interfaces

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

10 Configuration

The configuration model of this functional cluster is defined in [12]. This chapter defines the default values for attributes and semantic constraints for elements specified in [12] that are part of the configuration model of this functional cluster.

`Platform Health Management` is configured using `PlatformHealthManagementContribution`s and their contents. The reporting to `State Management` is configured using `PhmSupervisionRecoveryNotificationInterface`.

10.1 Default Values

This functional cluster does not define any default values for attributes specified in [12].

10.2 Semantic Constraints

This section defines semantic constraints for elements specified in [12] that are part of the configuration model of this functional cluster.

[SWS_PHM_CONSTR_00001] Configurable Namespace for PlatformHealthManagement ⌈`PlatformHealthManagementInterface.namespace` shall exist for `PhmSupervisedEntityInterface`.⌋

A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics. For further details, please refer chapters corresponding to below mentioned tables in [12].

Chapter is generated.

Class	**AliveSupervision**			
Note	Defines an AliveSupervision for one checkpoint. This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=checkpoint
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. **Tags:** atp.Splitkey=terminatingCheckpoint
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	**ApApplicationError**			
Note	This meta-class represents the ability to formally specify the semantics of an application error on the AUTOSAR adaptive platform **Tags:** atp.recommendedPackage=ApplicationErrors This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**ApApplicationError**			
errorCode	Integer	0..1	attr	This attribute has the ability to specify the error code value within the enclosing AdaptivePlatformApplication Error.
errorDomain	ApApplicationError Domain	0..1	ref	This reference represents the error domain of the Ap ApplicationError. **Tags:** atp.Splitkey=errorDomain

Table A.2: ApApplicationError

Class	**CheckpointTransition**			
Note	Defines one transition between two checkpoints. This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	GlobalSupervision.transition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
source	SupervisionCheckpoint	0..1	ref	Reference to the source checkpoint for this transition. **Tags:** atp.Splitkey=source
target	SupervisionCheckpoint	0..1	ref	Reference to the target checkpoint for this transition. **Tags:** atp.Splitkey=target

Table A.3: CheckpointTransition

Class	**DeadlineSupervision**			
Note	Defines an DeadlineSupervision for one transition. This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=transition

Table A.4: DeadlineSupervision

Class	**GlobalSupervision**			
Note	This element defines a collection of AliveSupervisions, DeadlineSupervisions, and LogicalSupervisions in order to provide an aggregated supervision state. This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	PlatformHealthManagementContribution.globalSupervision			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**GlobalSupervision**			
alive Supervision	AliveSupervision	*	aggr	Collection of AliveSupervisions in the context of this GlobalSupervision. **Tags:** atp.Splitkey=aliveSupervision, alive Supervision.shortName
deadline Supervision	DeadlineSupervision	*	aggr	Collection of DeadlineSupervisions in the context of this GlobalSupervision. **Tags:** atp.Splitkey=deadlineSupervision, deadline Supervision.shortName
expired Supervision Tolerance	TimeValue	0..1	attr	Defines the acceptable amount of time with EXPIRED supervision status before the GlobalSupervision is considered STOPPED.
logical Supervision	LogicalSupervision	*	aggr	Collection of LogicalSupervisions in the context of this GlobalSupervision. **Tags:** atp.Splitkey=logicalSupervision, logical Supervision.shortName
noCheckpoint Supervision	NoCheckpoint Supervision	*	aggr	Definition of No Checkpoint Supervision. **Tags:** atp.Splitkey=noCheckpointSupervision, no CheckpointSupervision.shortName
noSupervision	NoSupervision	*	aggr	Collection of NoSupervisions in the context of this Global Supervision. **Tags:** atp.Splitkey=noSupervision, noSupervision.short Name
supervision Mode	SupervisionMode	*	aggr	Collection of SupervisionModes in the context of this GlobalSupervision. **Tags:** atp.Splitkey=supervisionMode, supervision Mode.shortName
transition	CheckpointTransition	*	aggr	Collection of CheckpointTransitions in the context of this GlobalSupervision. **Tags:** atp.Splitkey=transition, transition.shortName

Table A.5: GlobalSupervision

Class	**LogicalSupervision**			
Note	Defines a LogicalSupervision graph consisting of transitions, initial- and final checkpoints. This Class is only used by the AUTOSAR Adaptive Platform.			
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:** atp.Splitkey=finalCheckpoint xml.sequenceOffset=20
initialCheckpoint	SupervisionCheckpoint	*	ref	Reference to the initial Checkpoint(s) for this Logical Supervision. **Tags:** atp.Splitkey=initialCheckpoint xml.sequenceOffset=10

▽

△

Class	LogicalSupervision			
transition	CheckpointTransition	*	ref	Reference to the transitions for this LogicalSupervision. **Tags:** atp.Splitkey=transition xml.sequenceOffset=30

Table A.6: LogicalSupervision

Class	NoCheckpointSupervision			
Note	Defines explicitly that NO supervision shall be applied for a set of SupervisionCheckpoints. This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=checkpoint

Table A.7: NoCheckpointSupervision

Class	NoSupervision			
Note	Defines explicitly that NO supervision shall be applied for a specific Supervised Entity instance. This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=process
targetPhm SupervisedEntity	RPortPrototype	0..1	iref	Instance reference to the RPortPrototype which represents the Supervised Entity instance. **Stereotypes:** atpUriDef **Tags:** atp.Splitkey=targetPhmSupervisedEntity, target PhmSupervisedEntity.contextRootSwComponent Prototype, targetPhmSupervisedEntity.contextComponent Prototype, targetPhmSupervisedEntity.targetRPort Prototype **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef

Table A.8: NoSupervision

Class	PhmCheckpoint			
Note	This meta-class provides the ability to implement a checkpoint for interaction with the Platform Health Management Supervised Entity. This Class is only used by the AUTOSAR Adaptive Platform.			
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.9: PhmCheckpoint

Class	**PhmSupervisedEntityInterface**			
Note	This meta-class provides the ability to implement a PortInterface for interaction with the Platform Health Management Supervised Entity. **Tags:** atp.recommendedPackage=PlatformHealthManagementInterfaces This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PlatformHealthManagementInterface*, *PortInterface*, *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. **Tags:** atp.Splitkey=checkpoint, checkpoint.shortName

Table A.10: PhmSupervisedEntityInterface

Class	**PhmSupervisionRecoveryNotificationInterface**			
Note	This meta-class represents a PortInterface that can be taken for implementing a PHM Supervision notification. **Tags:** atp.recommendedPackage=PlatformHealthManagementInterfaces This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PhmAbstractRecoveryNotificationInterface*, *PlatformHealthManagementInterface*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.11: PhmSupervisionRecoveryNotificationInterface

Class	**PlatformHealthManagementContribution**			
Note	This element defines a contribution to the Platform Health Management. **Tags:** atp.recommendedPackage=PlatformHealthManagementContributions This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
checkpoint	SupervisionCheckpoint	*	aggr	Collection of checkpoints in the context of a Platform HealthManagementContribution. **Tags:** atp.Splitkey=checkpoint, checkpoint.shortName xml.sequenceOffset=10
global Supervision	GlobalSupervision	*	aggr	Collection of GlobalSupervisions in the context of a PlatformHealthManagementContribution. **Tags:** atp.Splitkey=globalSupervision, globalSupervision.short Name xml.sequenceOffset=30

▽

△

Class	**PlatformHealthManagementContribution**			
healthChannel	HealthChannel	*	aggr	Collection of HealthChannels in the context of a Platform HealthManagementContribution. **Tags:** atp.Splitkey=healthChannel, healthChannel.shortName xml.sequenceOffset=40
supervision ModeCondition	SupervisionMode Condition	*	aggr	Collection of SupervisionModeConditions in the context of a PlatformHealthManagementContribution. **Tags:** atp.Splitkey=supervisionModeCondition, supervision ModeCondition.shortName xml.sequenceOffset=20

Table A.12: PlatformHealthManagementContribution

Class	***PlatformHealthManagementInterface*** (abstract)			
Note	This meta-class provides the abstract ability to define a PortInterface for the interaction with Platform Health Management. This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	*PhmAbstractRecoveryNotificationInterface*, PhmSupervisedEntityInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.13: PlatformHealthManagementInterface

Class	***PortInterface*** (abstract)			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractRawDataStreamInterface*, *AbstractSuspendToRamInterface*, *AbstractSynchronizedTimeBase Interface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallState SwitchInterface, *IdsmAbstractPortInterface*, LogAndTraceInterface, ModeSwitchInterface, Network ManagementPortInterface, *PersistencyInterface*, *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. This Attribute is only used by the AUTOSAR Adaptive Platform.

Table A.14: PortInterface

Class	***PortPrototype*** (abstract)
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.

▽

△

Class	**PortPrototype** (abstract)			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this `PortPrototype` with respect to client/ server communication. **Tags:** atp.Splitkey=clientServerAnnotation
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port. **Tags:** atp.Splitkey=delegatedPortAnnotation
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port. **Tags:** atp.Splitkey=ioHwAbstractionServerAnnotation
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port. **Tags:** atp.Splitkey=modePortAnnotation
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port. **Tags:** atp.Splitkey=nvDataPortAnnotation
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port. **Tags:** atp.Splitkey=parameterPortAnnotation
portPrototype Props	PortPrototypeProps	0..1	aggr	This attribute allows for the definition of further qualification of the semantics of a PortPrototype. **Tags:** atp.Splitkey=portPrototypeProps This Attribute is only used by the AUTOSAR Adaptive Platform.
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication. **Tags:** atp.Splitkey=senderReceiverAnnotation
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port. **Tags:** atp.Splitkey=triggerPortAnnotation

Table A.15: PortPrototype

Class	**Process**			
Note	This meta-class provides information required to execute the referenced `Executable`. **Tags:** atp.recommendedPackage=Processes This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=design
executable	Executable	*	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef **Tags:** atp.Splitkey=executable

▽

△

Class	**Process**			
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
posixUser	PosixUser	0..1	ref	This reference identifies the POSIX user ID under which the Process shall be launched. **Tags:** atp.Splitkey=posixUser
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory.
primaryGroup	PosixGroup	0..1	ref	This reference identifies the collection of primary group IDs under which the Process shall be launched. **Tags:** atp.Splitkey=primaryGroup
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process. This attribute is used to support the modeling of execution dependencies that utilize the condition of process state. Please note that the process states may not be modeled arbitrarily at any stage of the AUTOSAR workflow because the supported states are standardized in the context of the SWS Execution Management [10]. **Tags:** atp.Splitkey=processStateMachine, processState Machine.shortName
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations. **Tags:** atp.Splitkey=stateDependentStartupConfig
supplementary Group	PosixGroup	*	ref	This reference identifies the collection of supplementary group IDs under which the Process shall be launched. **Tags:** atp.Splitkey=supplementaryGroup

Table A.16: Process

Class	**RecoveryNotification**			
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 This Class is only used by the AUTOSAR Adaptive Platform.			
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**			
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 This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=process
recoveryAction	PPortPrototype	0..1	iref	This reference identifies the PortPrototype to be addressed as part of a PHM recovery. **Tags:** atp.Splitkey=recoveryAction, recovery Action.contextRootSwComponentPrototype, recovery Action.contextComponentPrototype, recovery Action.targetPPortPrototype **InstanceRef implemented by:** PPortPrototypeIn ExecutableInstanceRef
recovery Notification	RecoveryNotification	0..1	ref	This reference identifies the applicable Recovery Notification to be mapped. **Tags:** atp.Splitkey=recoveryNotification

Table A.18: RecoveryNotificationToPPortPrototypeMapping

Class	***Referrable*** (abstract)			
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, SomeipRequiredEventGroup, Tp ConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** atp.Splitkey=shortNameFragment xml.sequenceOffset=-90

Table A.19: Referrable

Class	**StartupConfig**			
Note	This meta-class represents a reusable startup configuration for processes.. **Tags:** atp.Status=obsolete atp.recommendedPackage=StartupConfigs This Class is only used by the AUTOSAR Adaptive Platform.			
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. **Tags:** atp.Splitkey=environmentVariable atp.Status=obsolete
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). **Tags:** atp.Status=obsolete
process Argument (ordered)	ProcessArgument	*	aggr	This aggregation represents the collection of command-line arguments applicable to the enclosing StartupConfig. **Tags:** atp.Status=obsolete
process ExecutionError	ApApplicationError	0..1	ref	this reference is used to identify the applicable execution error. **Tags:** atp.Splitkey=processExecutionError atp.Status=obsolete
scheduling Policy	String	0..1	attr	This attribute represents the ability to define the scheduling policy for the initial thread of the application. **Tags:** atp.Status=obsolete
scheduling Priority	Integer	0..1	attr	This is the scheduling priority requested by the application itself. **Tags:** atp.Status=obsolete
termination Behavior	TerminationBehavior Enum	0..1	attr	This attribute defines the termination behavior of the Process. **Tags:** atp.Status=obsolete
timeout	EnterExitTimeout	0..1	aggr	This aggregation can be used to specify the timeouts for launching and terminating the process depending on the StartupConfig. **Tags:** atp.Splitkey=timeout atp.Status=obsolete

Table A.20: StartupConfig

Class	SupervisionCheckpoint			
Note	This element contains an instance reference to a RPortPrototype representing a checkpoint for Platform Health Management. This Class is only used by the AUTOSAR Adaptive Platform.			
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 **Tags:** atp.Splitkey=phmCheckpoint, phm Checkpoint.contextRootSwComponentPrototype, phm Checkpoint.contextComponentPrototype, phm Checkpoint.contextRPortPrototype, phm Checkpoint.targetPhmCheckpoint **InstanceRef implemented by:** PhmCheckpointIn ExecutableInstanceRef
process	Process	0..1	ref	Reference to the Process this checkoint shall be monitored. **Tags:** atp.Splitkey=process

Table A.21: SupervisionCheckpoint

Class	***SwComponentType*** (abstract)			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	AdaptiveApplicationSwComponentType, *AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
port	PortPrototype	*	aggr	The `PortPrototype`s through which this `SwComponentType` can communicate. The aggregation of `PortPrototype` is subject to variability with the purpose to support the conditional existence of `PortPrototype`s. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=port, port.shortName, port.variation Point.shortLabel vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=portGroup, portGroup.shortName, port Group.variationPoint.shortLabel vh.latestBindingTime=preCompileTime



△

Class	***SwComponentType*** (abstract)			
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the `SwComponentType`. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, sw ComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10

Table A.22: SwComponentType

Class	**SymbolProps**			
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, *Port Interface*.namespace, SecurityEventDefinition.eventSymbolName, TraceSwitchConfig.namespace			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.23: SymbolProps

B Demands and constraints on Base Software (normative)

This functional cluster defines no demands or constraints for the Base Software on which the AUTOSAR Adaptive Platform is running on (usually a POSIX-compatible operating system).

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 Header: apext/phm/watchdog_interface.h

C.1.1 Class: WatchdogInterface

[SWS_PHM_01257] Definition of API class apext::phm::WatchdogInterface

Upstream requirements: RS_PHM_00101

⌈

Kind:	class
Header file:	#include "apext/phm/watchdog_interface.h"
Forwarding header file:	#include "apext/phm/phm_fwd.h"
Scope:	namespace apext::phm
Symbol:	WatchdogInterface
Syntax:	`class WatchdogInterface {...};`
Description:	class for interface to hardware watchdog Note: class WatchdogInterface although identifiable in code as an interface, it is a PlatformExtension - i.e. no modelled interface - nothing to do here

⌋

C.1.1.1 Public Member Functions

C.1.1.1.1 Member Functions

[SWS_PHM_01255] Definition of API function apext::phm::WatchdogInterface::AliveNotification

Upstream requirements: RS_PHM_00101, RS_PHM_09240

⌈

Kind:	function
Header file:	#include "apext/phm/watchdog_interface.h"
Scope:	`class apext::phm::WatchdogInterface`
Syntax:	`void AliveNotification ();`
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	thread-safe

▽

△

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

⌋

[SWS_PHM_01258] Definition of API function apext::phm::WatchdogInterface::DisableWatchdog

Upstream requirements: RS_PHM_00101

⌈

Kind:	function
Header file:	#include "apext/phm/watchdog_interface.h"
Scope:	`class apext::phm::WatchdogInterface`
Syntax:	`void DisableWatchdog ();`
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	Interface to disable the Watchdog.

⌋

[SWS_PHM_01259] Definition of API function apext::phm::WatchdogInterface::EnableWatchdog

Upstream requirements: RS_PHM_00101

⌈

Kind:	function
Header file:	#include "apext/phm/watchdog_interface.h"
Scope:	`class apext::phm::WatchdogInterface`
Syntax:	`void EnableWatchdog ();`
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	Interface to enable the Watchdog.

⌋

[SWS_PHM_01256] Definition of API function apext::phm::WatchdogInterface::FireWatchdogReaction

Upstream requirements: RS_PHM_00101

⌈

Kind:	function
Header file:	#include "apext/phm/watchdog_interface.h"
Scope:	`class apext::phm::WatchdogInterface`
Syntax:	`void FireWatchdogReaction ();`
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	thread-safe
Description:	Interface to fire an error reaction of the Watchdog.

⌋

D Not implemented requirements

This functional cluster implements all functional requirements specified in the corresponding requirement specifications.

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]	
[SWS_PHM_01137]	

▽

△

Number	Heading
[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
[SWS_PHM_01013]	Header file existence

▽

△

Number	Heading
[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
[SWS_PHM_01253]	Termination of Supervisions at SIGTERM
[SWS_PHM_01254]	Global Supervision Status at SIGTERM

▽

△

Number	Heading
[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

▽

△

Number	Heading
[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

E.4 Traceable item history of this document according to AUTOSAR Release R24-11

E.4.1 Added Specification Items in R24-11

[SWS_PHM_00107]

E.4.2 Changed Specification Items in R24-11

[SWS_PHM_00104] [SWS_PHM_00105] [SWS_PHM_00106] [SWS_PHM_00424] [SWS_PHM_00425] [SWS_PHM_01002] [SWS_PHM_01005] [SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01132] [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_01211] [SWS_PHM_01212] [SWS_PHM_01213] [SWS_PHM_01214] [SWS_PHM_01215] [SWS_PHM_01240] [SWS_PHM_01241] [SWS_PHM_01243] [SWS_PHM_01244] [SWS_PHM_01247] [SWS_PHM_01248] [SWS_PHM_01249] [SWS_PHM_01250] [SWS_PHM_01251] [SWS_PHM_01252] [SWS_PHM_01340]

E.4.3 Deleted Specification Items in R24-11

[SWS_PHM_00010] [SWS_PHM_00102] [SWS_PHM_00426] [SWS_PHM_00457] [SWS_PHM_01020] [SWS_PHM_01113] [SWS_PHM_01114] [SWS_PHM_01115] [SWS_PHM_01118] [SWS_PHM_01119] [SWS_PHM_01122] [SWS_PHM_01128] [SWS_PHM_01129] [SWS_PHM_01139] [SWS_PHM_01221] [SWS_PHM_01222] [SWS_PHM_01223] [SWS_PHM_01224] [SWS_PHM_01225] [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] [SWS_PHM_01328] [SWS_PHM_01329] [SWS_PHM_01330]

E.4.4 Added Constraints in R24-11

Number	Heading
[SWS_PHM_CONSTR_00001]	Configurable Namespace for PlatformHealthManagement

Table E.10: Added Constraints in R24-11

E.4.5 Changed Constraints in R24-11

none

E.4.6 Deleted Constraints in R24-11

none

E.5 Traceable item history of this document according to AUTOSAR Release R25-11

E.5.1 Added Specification Items in R25-11

[SWS_PHM_01130] [SWS_PHM_01255] [SWS_PHM_01256] [SWS_PHM_01257] [SWS_PHM_01359] [SWS_PHM_01360] [SWS_PHM_01361] [SWS_PHM_01363] [SWS_PHM_01364] [SWS_PHM_01365] [SWS_PHM_01366] [SWS_PHM_01367] [SWS_PHM_01370] [SWS_PHM_01371] [SWS_PHM_01372] [SWS_PHM_01373] [SWS_PHM_01375] [SWS_PHM_01380]

E.5.2 Changed Specification Items in R25-11

[SWS_PHM_00101] [SWS_PHM_00216] [SWS_PHM_01002] [SWS_PHM_01123] [SWS_PHM_01132] [SWS_PHM_01137] [SWS_PHM_01138] [SWS_PHM_01140] [SWS_PHM_01141] [SWS_PHM_01149] [SWS_PHM_01151] [SWS_PHM_01214] [SWS_PHM_01240] [SWS_PHM_01248] [SWS_PHM_01254] [SWS_PHM_01332] [SWS_PHM_01336] [SWS_PHM_01339] [SWS_PHM_01340] [SWS_PHM_01343] [SWS_PHM_01345] [SWS_PHM_01358]

E.5.3 Deleted Specification Items in R25-11

none

E.5.4 Added Constraints in R25-11

none

E.5.5 Changed Constraints in R25-11

none

E.5.6 Deleted Constraints in R25-11

none

E.6 Traceable item history of this document according to AUTOSAR Release R26-11

E.6.1 Added Specification Items in R26-11

[SWS_PHM_01258] [SWS_PHM_01259] [SWS_PHM_01362] [SWS_PHM_01381] [SWS_PHM_01382] [SWS_PHM_01383] [SWS_PHM_01384] [SWS_PHM_01385] [SWS_PHM_01386]

E.6.2 Changed Specification Items in R26-11

[SWS_PHM_01123] [SWS_PHM_01127] [SWS_PHM_01141] [SWS_PHM_01240] [SWS_PHM_01365] [SWS_PHM_01380]

E.6.3 Deleted Specification Items in R26-11

none

E.6.4 Added Constraints in R26-11

none

E.6.5 Changed Constraints in R26-11

none

E.6.6 Deleted Constraints in R26-11

none