Document Title	Specification of Watchdog Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	80
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R22-11

Document Change History

Date	Release	Changed by	Change Description
2022-11-24	R22-11	AUTOSAR Release Management	• Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Resolved inconsistency regarding determination of Supervised Entity ID values, between SWS WdgM and TPS • Set “Partition Restart / Shutdown” feature to obsolete • Removed the redundant parameter WdgMDemStoppedSupervisionReport • Extended to support supervision for Clustered Software Architecture (Classic Platform Flexibility), incl. support of multiple main functions
2020-11-30	R20-11	AUTOSAR Release Management	• Clarified the meaning of thresholds WdgMDeadlineMin and WdgMDeadlineMax • Updated the structure and tables of the error sections • Editorial/Minor Corrections

Document Change History

Date	Release	Changed by	Change Description
2019-11-28	R19-11	AUTOSAR Release Management	• Enhancement of Deadline Supervision to support timeout detection • Correction/Clarification of Supervision Algorithms and their configurations • Clarification of startup behaviour (incl. failed WdgIf_SetMode during init) • Corrected/Changed/Added Error Codes and other editorial issues • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Header File Cleanup • EcuPartition vs. OSApplication • Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Correction in development errors. • Renaming of default error to development errors.
2016-11-30	4.3.0	AUTOSAR Release Management	• Deprecated features removed • Service interfaces modified/corrected • Removed duplicate type definitions • Several minor fixes.
2015-07-31	4.2.2	AUTOSAR Release Management	• Debugging support marked as obsolete • Several minor fixes. • Fixed handling of development errors.
2014-10-31	4.2.1	AUTOSAR Release Management	• Introduced of the modeling of system services • Reformulated some requirements to constraints • Minor corrections

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2014-03-31	4.1.3	AUTOSAR Release Management	• Addition of the OS counters for deadline monitoring • Fixed data types for Supervised Entity and Checkpoint types (uint16) • Several minor corrections throughout the document
2013-10-31	4.1.2	AUTOSAR Release Management	• Minor fixes (mode switching, dependencies to other modules) • Quality corrections in the document (e.g. formatting of requirements) • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneral • New indexing scheme for requirements • Clarification in Deadline Supervision • Minor corrections in Specification of the Ports and Port Interfaces
2013-03-15	4.1.1	AUTOSAR Administration	• Include file structure changed • Added a method to read after restart which SE caused the reset: WdgM_GetFirstExpiredSEID. • New template with requirements traceability
2011-12-22	4.0.3	AUTOSAR Administration	• Streamlined the used terms • Reorganized structure of some chapters • Clarified ambigious statements and resolved contradicting ones • Corrected several bugs • Provided more details what WdgM functions do and in which sequence

Document Change History

Date	Release	Changed by	Change Description
2010-09-30	3.1.5	AUTOSAR Administration	• New concept of windowed watchdogs • New supervision functions, Logical Supervision and Deadline Supervision • Split of the supervision status into local and global supervision status • New concept for activation and deactivation of supervision • New concept of Defensive Behavior • New failure recovery concept for partition (application) restart • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Extended mode concept • Added GPT as activation source for operation during Startup, Shutdown, and Sleep • Restructured module configuration • Generated APIs from BSW UML model • Generated configuration from Meta Model • Document meta information extended • Small layout adaptations made

Document Change History

Date	Release	Changed by	Change Description
2007-01-24	2.1.15	AUTOSAR Administration	• New chapter "Specification of the ports and port interfaces" added from "AUTOSAR Services" document • New feature added : active reset as optional behavior • New behavior of Deinit function : triggering of the Watchdog Driver added • Default mode for the Watchdog Manager when SetMode service fails • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• 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

List of Figures

1 Introduction and Functional Overview

The Watchdog Manager is a basic software module at the service layer of the standardized basic software architecture of AUTOSAR.

The Watchdog Manager is able to supervise the program execution abstracting from the triggering of hardware watchdog entities.

The Watchdog Manager supervises the execution of a configurable number of so-called *Supervised Entities*. When it detects a violation of the configured temporal and/or logical constraints on program execution, it takes a number of configurable actions to recover from this failure.

The watchdog Manager provides three mechanisms:

1. *Alive Supervision* – for supervision of timing of ***periodic*** software
2. *Deadline Supervision* – for supervision of timing of ***aperiodic*** software
3. *Logical Supervision* – for supervision of the correctness of the execution sequence.

1.1 Supervised Entities and Checkpoints

The Watchdog Manager supervises the execution of software. The logical units of supervision are *Supervised Entities*. There is no fixed relationship between *Supervised Entities* and the architectural building blocks in AUTOSAR, i.e., SW-Cs, CDDs, RTE, BSW modules, but typically a *Supervised Entity* may represent one SW-C Prototype or one or more Runnable Entities within a SW-C Prototype, a BSW module instance or CDD instance depending on the choice of the developer.

Important places in a *Supervised Entity* are defined as *Checkpoints*. The code of *Supervised Entities* is interlaced with the calls of Watchdog Manager that report to the Watchdog Manager when they have reached a *Checkpoint*.

Each *Supervised Entity* has one or more *Checkpoints*. The *Checkpoints* and *Transitions* between the *Checkpoints* of a *Supervised Entity* form a *Graph*. This *Graph* is called *Internal Graph*. Moreover, *Checkpoints* from different *Supervised Entities* may also be connected by *External Transition*, forming an *External Graph*. There can be several *External Graphs* in each Watchdog Manager *Mode*.

A *Graph* may have one or more *Initial Checkpoints* and one or more *Final Checkpoints*. Any sequence of starting with any *Initial Checkpoint* and finishing with any *Final Checkpoint* is correct (assuming that the checkpoints belong to the same *Graph*). After the *Final Checkpoint*, any *Initial Checkpoint* can be reported.

Within the Watchdog Manager configurations, it is possible to configure the required timing of *Checkpoints* as well as the allowed *External* and *Internal Graphs*.

At runtime, Watchdog Manager verifies if the configured *Graphs* are executed. This is called *Logical Supervision*. Watchdog Manager also verifies the timing of *Checkpoints* and *Transitions*. The mechanism for periodic *Checkpoints* is called *Alive Supervision* and for aperiodic *Checkpoints* it is called *Deadline Supervision*.

The granularity of *Checkpoints* is not fixed by the Watchdog Manager. Few coarse-grained *Checkpoints* limit the detection abilities of the Watchdog Manager. For example, if an application SW-C only has one *Checkpoint* that indicates that a cyclic Runnable has been started, then the Watchdog Manager is only capable of detecting that this Runnable is re-started and check the timing constraints. In contrast, if that SW-C has *Checkpoints* at each block and branch in the Runnable the Watchdog Manager may also detect failures in the control flow of that SW-C. High granularity of *Checkpoints* causes a complex and large configuration of the Watchdog Manager.

1.2 Interaction of Supervision Mechanisms

The three supervision mechanisms supervise each *Supervised Entity*. A *Supervised Entity* may have one, two or three mechanisms enabled. Based on the results from each of enabled mechanisms, the status of the *Supervised Entity* (called *Local Supervision* Status) is computed.

When the status of each *Supervised Entity* is determined, then based on each *Local Supervision Status*, the status of the whole MCU is determined (called *Global Supervision Status*).

1.3 Supervision Functions

1.3.1 Alive Supervision

Periodic *Supervised Entities* have constraints on the number of times they are executed within a given time span. By means of *Alive Supervision*, Watchdog Manager checks periodically if the *Checkpoints* of a *Supervised Entity* have been reached within the given limits. This means that Watchdog Manger checks if a *Supervised Entity* is run not too frequently or not too rarely.

1.3.2 Deadline Supervision

Aperiodic or episodical *Supervised Entities* have individual constraints on the timing between two *Checkpoints*. By means of *Deadline Supervision*, Watchdog Manager checks the timing of transitions between two *Checkpoints* of a *Supervised Entity*. This means that Watchdog Manager checks if some steps in a *Supervised Entity* take a time that is within the configured minimum and maximum values. Watchdog Manager also detects no arrival to the second *Checkpoint*.

1.3.3 Logical Supervision

Logical Supervision is a fundamental technique for checking the correct execution of embedded system software. Please refer to the safety standards (IEC 61508 or ISO 26262) when *Logical Supervision* is required.

Logical Supervision focuses on control flow errors, which cause a divergence from the valid (i.e. coded/compiled) program sequence during the error-free execution of the application. An incorrect control flow occurs if one or more program instructions are processed either in the incorrect sequence or are not even processed at all. Control flow errors can lead to data corruption, microcontroller resets, or fail-silence violations.

For the control flow graph this implies that every time the *Supervised Entity* reports a new *Checkpoint*, it must be verified that there is a *Transition* configured between the previous *Checkpoint* and the reported one.

1.4 Watchdog Handling

Watchdog Manager communicates with Watchdog Interface to control the hardware watchdog.

In contrast to versions V1.x.y (before R4.0.1), the Watchdog Manager is no longer responsible for triggering the hardware watchdog via the Watchdog Interface and the Watchdog Driver. Instead, the Watchdog Manager reports via the Watchdog Interface a triggering condition to the Watchdog Driver. The Watchdog Driver is then responsible for triggering the hardware watchdog with the right timing for as long as the condition is true. The triggering condition is a counter value that the Watchdog Manager sets cyclically. The Watchdog Driver decrements this counter every time it triggers the hardware watchdog. When the counter reaches 0, the Watchdog Driver stops triggering the hardware watchdog. Therefore, when the Watchdog Manager fails to execute, this automatically causes a watchdog reset (after the time needed to decrement the counter plus the timeout value of HW watchdog).

When the *Supervised Entities* are not correctly evaluated due to a programming error or memory failure in the Watchdog Manager itself, it may still happen that the Watchdog Manager erroneously sets the triggering condition and no watchdog reset will be caused. Therefore, it may be needed to use *Supervised Entities* and *Checkpoints* (or some other internal supervision mechanism) within Watchdog Manager itself, while avoiding recursion in Watchdog Manager.

1.5 Error Handling

Depending on the *Local Supervision* Status of each *Supervised Entity* and on the *Global Supervision Status*, the Watchdog Manager initiates a number of mechanisms to recover from supervision failures. These range from local error recovery within the *Supervised Entity* to a global reset of the ECU.

1.5.1 Error Handling in the Supervised Entity

In case the *Supervised Entity* is an SW-C or a CDD, then the Watchdog Manager may inform the *Supervised Entity* about supervision failures via the RTE Mode mechanism. The *Supervised Entity* may then take its actions to recover from that failure.

The Watchdog Manager may register an entry with the Diagnostic Event Manager (DEM) when it detects a supervision failure. A *Supervised Entity* may take recovery actions based on that error entry.

1.5.2 Reset by Hardware Watchdog

The Watchdog Manager indicates to the Watchdog Interface when Watchdog Interface shall no longer trigger the hardware watchdog. After the timeout of the hardware watchdog, the hardware watchdog resets the ECU or the MCU. This leads to a re-initialization of the ECU and/or MCU hardware and the complete reinitialization of software.

1.5.3 Immediate MCU Reset

In case an immediate, global reaction to the supervision failure is necessary, the Watchdog Manager may directly cause an MCU reset. This will lead to a re-initialization of the MCU hardware and the complete software. Usually, a MCU reset will not re-initialize the rest of the ECU hardware.

Note that a MCU reset is not available on some types of micro controllers.

MCU reset and watchdog reset are two mostly equivalent mechanisms for system-level error reaction. In safety-related systems, it is recommended to use both of them in parallel. By this means, the two mechanisms make a “redundant shutdown path”.

2 Acronyms, Abbreviations and Terms

Abbreviation / Acronym	***Description***
AI	Alive Indication
BSW	Basic Software
BswM	Basic Software Mode Manager
DEM	Diagnostic Event Manager
DET	Default Error Tracer
EAI	Expected Alive Indications
EcuM	ECU State Manager
FiM	Function Inhibition Manager
HW	Hardware
ID	Identifier
MCU	Micro Controller Unit
OS	Operating System
SC	Supervision Cycle
SE	Supervised Entity
SRC	Supervision Reference Cycle
SW-C	Software Component
RTE	Runtime Environment
WdgM	Watchdog Manager

Term	***Description***
Alive Counter	An independent data resource in the Watchdog Manager in context of a *Checkpoint* to track and handle its amount of *Alive Indications*.
Alive Indication	An indication provided by a *Checkpoint* of a *Supervised Entity* to signal its aliveness to the Watchdog Manager.
Alive Supervision	Kind of supervision that checks if a *Supervised Entity* executed sufficiently often and not too often (including tolerances).
Checkpoint	A point in the control flow of a *Supervised Entity* where the activity is reported to the Watchdog Manager.
Deadline Supervision	Kind of supervision that checks if the execution time between two *Checkpoints* are lower than a given upper execution time limit.
Deadline Start Checkpoint	A *Checkpoint* for which *Deadline Supervision* is configured and which is a starting point for a particular *Deadline Supervision*.
Deadline End Checkpoint	A *Checkpoint* for which *Deadline Supervision* is configured and which is an ending point for a particular *Deadline Supervision*. It is possible that a *Checkpoint* is both a *Deadline Start Checkpoint* and *Deadline End Checkpoint* – if *Deadline Supervision* is chained.
Expired Supervision Cycle	A *Supervision Cycle* where the *Alive Supervision* has failed its two escalation steps (*Alive Counter* fails the expected amount of *Alive Indications* (including tolerances) more often than the allowed amount of failed reference cycles).
Failed Supervision Reference Cycle	A *Supervision Reference Cycle* that ends with a detected deviation (including tolerances) between the *Alive Counter* and the expected amount of *Alive*

Term	***Description***
	Indications.
Global Supervision Status	Status that summarizes the *Local Supervision Status* of all *Supervised Entities*.
Graph	A set of *Checkpoints* connected through *Transitions*, where at least one of *Checkpoints* is an *Initial Checkpoint*. There is a path (through *Transitions*) between any two *Checkpoints* of the *Graph*.
External Graph	*Graph* that may involve more than one Supervised Entity. Its configuration is mode-dependent.
Cross-Cluster External Graph	A special kind of *External Graph* that spans over multiple Software Clusters for Clustered Software Architecture. Its configuration is mode-dependent (controlled by Host Software Cluster) and has dedicated configuration structure additionally. Note: *External Graph* within one Software Cluster can be modelled without the configuration structure dedicated for clustered software architecture.
External Transition	An *External Transition* is a transition between two *Checkpoints*, where the *Checkpoints* belong to different *Supervised Entities*.
Local Supervision Status	Status that represents the current result of alive-supervision of a single *Supervised Entity*.
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).
Internal Graph	*Graph* that may not span over several *Supervised Entity*. Its configuration is mode-independent and can be disabled by disabling the corresponding *Supervised Entity*.
Internal Transition	An *Internal Transition* is a transition between two *Checkpoints* of a *Supervised Entity*.
Mode	A mode is a certain set of states of the various state machines that are running in the vehicle that are relevant to a particular entity, e.g. a SW-C, a BSW module, an application, a whole vehicle In its lifetime, an entity changes between a set of mutually exclusive modes. These changes are triggered by environmental data, e.g. signal reception, operation invocation. In the context of the Watchdog Manager a mode is defined by a set of configuration options. The set of *Supervised Entities* to be supervised may vary from mode to mode.
Supervised Entity	A software entity which is included in the supervision of the Watchdog Manager. Each *Supervised Entity* has exactly one identifier. A *Supervised Entity* denotes a collection of *Checkpoints* within an instance of Software Component Types or Basic Software Modules. There may be zero, one or more *Supervised Entities* in an instance of Software Component Types or Basic Software Modules.
Supervised Entity Identifier	An Identifier that identifies uniquely a *Supervised Entity* within an Application.
Supervision Counter	An independent data resource in context of a *Supervised Entity* which is updated by the Watchdog Manager during each *Supervision Cycle* and which is used by the *Alive Supervision* algorithm to perform the

Term	***Description***
	check against counted *Alive Indications*.
Supervision Cycle	The time base of *Supervision Reference Cycle* of Watchdog Manager, where the cyclic *Alive Supervision* is performed. And it's also the interval for updating *Global Supervision Status* and execution of resulting Recovery Actions. This is done in every call of the Main Function of belonging Watchdog Manager and mode-dependent (may vary when swiching mode).
Supervision Reference Cycle	The amount of *Supervision Cycles* to be used as reference by the *Alive Supervision* to perform the check of counted *Alive Indications* (individually for each *Supervised Entity*) and mode-dependent.

3 Related Documentation

3.1 Input Documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] Requirements on Mode Management
AUTOSAR_SRS_ModeManagement.pdf

[4] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[5] Specification of RTE
AUTOSAR_SWS_RTE.pdf

[6] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[7] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[8] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[9] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [9] (SWS BSW General), which is also valid for Watchdog Manager.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Watchdog Manager.

4 Constraints and Assumptions

4.1 Limitations and conditions of use

The main limitations of Watchdog Manager design are as follows. They may be removed in upcoming versions of this document:

- **{DRAFT}** A *Supervised Entity* cannot span over multiple EcucPartitions.
- **{DRAFT}** Handling of unconnected transition proxies for *Logical Supervision* based on *Cross-Cluster External Graph* by Watchdog Manager is unspecified in this release.
- As libraries cannot call BSWs, libraries cannot be supervised by Watchdog Manager.
- The nesting of *Deadline Supervision* (i.e. start 1, start 2, end 2, end 1) is not supported.
- The *Alive Supervision* function with more than one *Checkpoint* per *Supervised Entity* is not consistently specified within the document. For now, it is recommended to support only one *Alive Supervision Checkpoint* per *Supervised Entity*.

Further limitations:

- The Watchdog Manager does not encapsulate the Watchdog Driver initialization. The Watchdog Driver must be initialized by the ECU State Manager [6] in the startup process before the initialization of the Watchdog Manager.
- The Watchdog Manager is initialized after the OS has been started. Hence, it cannot be responsible for controlling the Watchdog Driver earlier in the startup process. Usually, it is sufficient to configure a large enough initial timeout in the Watchdog Driver to bridge the gap between Watchdog Driver and Watchdog Manager initialization. Alternatively, the Integrator may use ECU State Manager facilities (callouts).
- The Watchdog Manager is de-initialized before the OS shutdown. Hence, it cannot be responsible for controlling the Watchdog Driver later in the shutdown process. Usually, it is sufficient to configure a large enough final timeout that is set when the Watchdog Manager is de-initialized. This allows bridging the gap between Watchdog Manager de-initialization and system power-off or resetting. Alternatively, the Integrator may use ECU State Manager facilities (callouts).
- For ECUs which implement sleep modes, if the hardware watchdog remains active in these sleep modes, its triggering shall also be handled by the ECU State Manager.

- The error recovery mechanism “Immediate MCU Reset” is available only on microcontrollers that are able to perform a reset by using the hardware feature of the microcontroller.
- All of following conditions must be met for the expected operation of WdgM supervision:
  - Initialized Wdg Interface,
  - Initialized OS (because of possible usage of OsCounter)
  - Initialized WdgM (done by calling `WdgM_Init`)
  - Periodic invocation of `WdgM_MainFunction` preferably by AUTOSAR BSW scheduler; during startup the invocation may be done by another module.
    - Note: The deviations/jitters on the periodic call of `WdgM_MainFunction` will lead to a potential risk of delayed detection in both *Alive Supervision* and *Deadline Supervision* (timeout detection part) and false/missed detection in *Alive Supervision*.
    - Note: Any blocking of this periodic invocation will cause loss of *Deadline Supervision* (timeout detection part), *Alive Supervision*, all state transition of both L*ocal/Global Supervision Status* and resulting Error Handling mechanisms to recover from supervision failures, except the last resort "Reset by Hardware Watchdog" due to the loss of the Watchdog Handling (no trigger to the hardware instance via WdgIf).
- A *Supervised Entity* with all its *Checkpoints* may belong to only one OS-Application (at most). Because OS-application can run on one core only, therefore one specific *Supervised Entity* may run at one core.
- The *Deadline Supervision* (timeout detection part) and *Alive Supervision* is highly depending on the periodic invocation of `WdgM_MainFunction`: the periodicity shall be chosen carefully according to the requested value of the timeout detection.
- **{DRAFT}** The result of `WdgM_GetFirstExpiredSEID` in software architecture with multi-partition configuration may be not fully reliable, depending on implementation (at least, it cannot be achieved without reliable and common time stamping over partitions, but it will not to be standardized).
- Watchdog Manager cannot detect timeout of *Deadline Supervision* for the *Supervised Entities* which are running in Category 2 ISRs.
  - Rationale: A deadlock of Runnable Entities which are running in Category 2 ISR blocks the execution of `WdgM_MainFunction` on Task level.

4.2 Applicability to Car Domains

No restriction

5 Dependencies to Other Modules

- Watchdog Interface (WdgIf)
  The Watchdog Manager module is responsible for changing the mode of the Watchdog Driver and for reporting to the Watchdog Driver the condition to trigger the hardware watchdog. The services of the Watchdog Driver are accessed via the Watchdog Interface which allows addressing multiple watchdog instances.
- ECU State Manager (EcuM)
  The ECU State Manager is responsible for initializing, de-initializing of the Watchdog Manager module and for triggering the hardware watchdog in sleep modes.
- Micro Controller Unit Driver (Mcu)
  The Watchdog Manager module may perform an immediate reset of the ECU in case of a supervision failure. This reset service is provided by the MCU driver.
- Default Error Tracer (Det)
  If development error detection is enabled, the Watchdog Manager module informs the Default Error Tracer about detected development errors.
- Diagnostic Event Manager (Dem)
  The Watchdog Manager may notify the Diagnostic Event Manager about detected functional / production-code relevant errors.
- BSW Scheduler (SchM)
  The BSW Scheduler is responsible for calling the scheduled functions of the Watchdog Manager module. The Watchdog Manager module uses the services of the BSW Scheduler to implement critical sections.
- Runtime Environment (Rte)
  The Runtime Environment is responsible for propagating *Checkpoint* information from *Supervised Entities* in SW-Cs or in CDDs to the Watchdog Manager module. The Watchdog Manager module uses the services of the Runtime Environment to inform SW-Cs about changes in the supervision status. BSW Modules can call the Watchdog Manager module without using RTE.
- Operating system (OS)
  The Operating System is used by Watchdog Manager to provide the timestamp.
- {DRAFT} Software Cluster Connector (SwCluC)
  SwCluC (introduced by Classic Platform Flexility Concept) can establish internal connection of WdgM over Software Clusters, by means of Binary Manifests. Note that, inter-EcucPartition connection within a WdgM will be established without SwCluC, as it’s a part of BSW Multicore Distribution Concept and its way for implementation is not standardized (one of typical implementation method is master-satellite pattern).

5.1 File Structure

5.1.1 Code File Structure

For details refer to the chapter 5.1.6 “Code file structure” in *SWS_BSWGeneral.*

5.2 Version Check

For details refer to the chapter 5.1.8 “Version Check” in *SWS_BSWGeneral.*

6 Requirements Traceability

Requirement	Description	Satisfied by
RS_HM_09235	Health Monitoring shall provide a Deadline Supervision	SWS_WdgM_00322, SWS_WdgM_00373, SWS_WdgM_00374, SWS_WdgM_00403, SWS_WdgM_00404
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_WdgM_NA_00345
SRS_BSW_00004	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	SWS_WdgM_NA_00345
SRS_BSW_00005	Modules of the \mu C Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_WdgM_NA_00345
SRS_BSW_00006	The source code of software modules above the \mu C Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_WdgM_NA_00345
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_WdgM_NA_00345
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_WdgM_NA_00345
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_WdgM_NA_00345
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_WdgM_00018, SWS_WdgM_00135, SWS_WdgM_00268, SWS_WdgM_00269, SWS_WdgM_00285, SWS_WdgM_00296, SWS_WdgM_00298, SWS_WdgM_00370
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_WdgM_NA_00345
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_WdgM_NA_00345
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction	SWS_WdgM_NA_00345

	layer which provides a standardized interface to higher software layers	
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_WdgM_NA_00345
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_WdgM_NA_00345
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_WdgM_NA_00345
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_WdgM_NA_00345
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_WdgM_NA_00345
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_WdgM_00104
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_WdgM_NA_00345
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_WdgM_NA_00345
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_WdgM_NA_00345
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_WdgM_NA_00345
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	SWS_WdgM_NA_00345

SRS_BSW_00305	Data types naming convention	SWS_WdgM_NA_00345
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_WdgM_NA_00345
SRS_BSW_00307	Global variables naming convention	SWS_WdgM_NA_00345
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_WdgM_NA_00345
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_WdgM_NA_00345
SRS_BSW_00310	API naming convention	SWS_WdgM_00151, SWS_WdgM_00153, SWS_WdgM_00154, SWS_WdgM_00159, SWS_WdgM_00168, SWS_WdgM_00169, SWS_WdgM_00175, SWS_WdgM_00261, SWS_WdgM_00263, SWS_WdgM_00264
SRS_BSW_00312	Shared code shall be reentrant	SWS_WdgM_NA_00345
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_WdgM_NA_00345
SRS_BSW_00318	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	SWS_WdgM_NA_00345
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_WdgM_NA_00345
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_WdgM_00010, SWS_WdgM_00020, SWS_WdgM_00021, SWS_WdgM_00030, SWS_WdgM_00031, SWS_WdgM_00039, SWS_WdgM_00172, SWS_WdgM_00173, SWS_WdgM_00176, SWS_WdgM_00253, SWS_WdgM_00254, SWS_WdgM_00256, SWS_WdgM_00257, SWS_WdgM_00258, SWS_WdgM_00270, SWS_WdgM_00278, SWS_WdgM_00279, SWS_WdgM_00284, SWS_WdgM_00288, SWS_WdgM_00388, SWS_WdgM_00389, SWS_WdgM_00390, SWS_WdgM_00392, SWS_WdgM_00394, SWS_WdgM_00395, SWS_WdgM_00396, SWS_WdgM_00397, SWS_WdgM_00401
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in	SWS_WdgM_NA_00345

	interrupt context shall be kept short	
SRS_BSW_00327	Error values naming convention	SWS_WdgM_00004, SWS_WdgM_00375, SWS_WdgM_00402
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_WdgM_NA_00345
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_WdgM_NA_00345
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_WdgM_NA_00345
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_WdgM_NA_00345
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_WdgM_NA_00345
SRS_BSW_00335	Status values naming convention	SWS_WdgM_NA_00345
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_WdgM_00261
SRS_BSW_00337	Classification of development errors	SWS_WdgM_00004, SWS_WdgM_00375, SWS_WdgM_00402
SRS_BSW_00339	Reporting of production relevant error status	SWS_WdgM_00129, SWS_WdgM_00142, SWS_WdgM_00408
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_WdgM_NA_00345
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_WdgM_NA_00345
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_WdgM_NA_00345
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_WdgM_NA_00345
SRS_BSW_00345	BSW Modules shall support pre-compile configuration	SWS_WdgM_00025, SWS_WdgM_00104
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set	SWS_WdgM_NA_00345

	of module files	
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_WdgM_NA_00345
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_WdgM_NA_00345
SRS_BSW_00350	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.	SWS_WdgM_00010, SWS_WdgM_00020, SWS_WdgM_00021, SWS_WdgM_00039, SWS_WdgM_00172, SWS_WdgM_00173, SWS_WdgM_00176, SWS_WdgM_00253, SWS_WdgM_00254, SWS_WdgM_00256, SWS_WdgM_00257, SWS_WdgM_00258, SWS_WdgM_00270, SWS_WdgM_00278, SWS_WdgM_00279, SWS_WdgM_00284, SWS_WdgM_00288, SWS_WdgM_00388, SWS_WdgM_00389, SWS_WdgM_00390, SWS_WdgM_00392, SWS_WdgM_00394, SWS_WdgM_00395, SWS_WdgM_00396, SWS_WdgM_00397, SWS_WdgM_00401
SRS_BSW_00351	Encapsulation of compiler specific methods to map objects	SWS_WdgM_NA_00345
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_WdgM_NA_00345
SRS_BSW_00357	For success/failure of an API call a standard return type shall be defined	SWS_WdgM_00011
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_WdgM_00151
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_WdgM_NA_00345
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_WdgM_NA_00345
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_WdgM_NA_00345
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_WdgM_00159

SRS_BSW_00374	All Basic Software Modules shall provide a readable module vendor identification	SWS_WdgM_NA_00345
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_WdgM_NA_00345
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_WdgM_NA_00345
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_WdgM_NA_00345
SRS_BSW_00379	All software modules shall provide a module identifier in the header file and in the module XML description file.	SWS_WdgM_NA_00345
SRS_BSW_00380	Configuration parameters being stored in memory shall be placed into separate c-files	SWS_WdgM_NA_00345
SRS_BSW_00383	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	SWS_WdgM_NA_00345
SRS_BSW_00384	The Basic Software Module specifications shall specify at least in the description which other modules they require	SWS_WdgM_NA_00345
SRS_BSW_00385	List possible error notifications	SWS_WdgM_00004, SWS_WdgM_00375, SWS_WdgM_00402
SRS_BSW_00386	The BSW shall specify the configuration and conditions for detecting an error	SWS_WdgM_NA_00345
SRS_BSW_00388	Containers shall be used to group configuration parameters that are defined for the same object	SWS_WdgM_NA_00345
SRS_BSW_00389	Containers shall have names	SWS_WdgM_NA_00345
SRS_BSW_00390	Parameter content shall be unique within the module	SWS_WdgM_NA_00345
SRS_BSW_00392	Parameters shall have a type	SWS_WdgM_NA_00345
SRS_BSW_00393	Parameters shall have a range	SWS_WdgM_NA_00345
SRS_BSW_00394	The Basic Software Module specifications shall specify the scope of the configuration parameters	SWS_WdgM_NA_00345

SRS_BSW_00395	The Basic Software Module specifications shall list all configuration parameter dependencies	SWS_WdgM_NA_00345
SRS_BSW_00396	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	SWS_WdgM_NA_00345
SRS_BSW_00397	The configuration parameters in pre-compile time are fixed before compilation starts	SWS_WdgM_NA_00345
SRS_BSW_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_WdgM_NA_00345
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_WdgM_NA_00345
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_WdgM_NA_00345
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_WdgM_NA_00345
SRS_BSW_00402	Each module shall provide version information	SWS_WdgM_NA_00345
SRS_BSW_00403	The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets	SWS_WdgM_NA_00345
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_WdgM_NA_00345
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_WdgM_NA_00345
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_WdgM_00021, SWS_WdgM_00039
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module	SWS_WdgM_NA_00345

	implementation	
SRS_BSW_00408	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	SWS_WdgM_NA_00345
SRS_BSW_00409	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	SWS_WdgM_NA_00345
SRS_BSW_00410	Compiler switches shall have defined values	SWS_WdgM_NA_00345
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_WdgM_NA_00345
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_WdgM_NA_00345
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_WdgM_NA_00345
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_WdgM_NA_00345
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_WdgM_NA_00345
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	SWS_WdgM_NA_00345
SRS_BSW_00419	If a pre-compile time configuration parameter is implemented as const it should be placed into a separate c-file	SWS_WdgM_NA_00345
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the Dem	SWS_WdgM_NA_00345
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_WdgM_NA_00345
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a	SWS_WdgM_NA_00345

	wait state	
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_WdgM_NA_00345
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_WdgM_NA_00345
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_WdgM_NA_00345
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_WdgM_NA_00345
SRS_BSW_00429	Access to OS is restricted	SWS_WdgM_NA_00345
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_WdgM_NA_00345
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_WdgM_NA_00345
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_WdgM_NA_00345
SRS_BSW_00438	Configuration data shall be defined in a structure	SWS_WdgM_NA_00345
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_WdgM_NA_00345
SRS_BSW_00440	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	SWS_WdgM_NA_00345
SRS_BSW_00441	Naming convention for type, macro and function	SWS_WdgM_NA_00345
SRS_BSW_00447	Standardizing Include file structure of BSW Modules Implementing Autosar Service	SWS_WdgM_NA_00345
SRS_BSW_00448	Module SWS shall not contain requirements from	SWS_WdgM_NA_00345

	other modules	
SRS_BSW_00449	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	SWS_WdgM_NA_00345
SRS_BSW_00450	A Main function of a un-initialized module shall return immediately	SWS_WdgM_00406, SWS_WdgM_00407
SRS_BSW_00451	Hardware registers shall be protected if concurrent access to these registers occur	SWS_WdgM_NA_00345
SRS_BSW_00452	Classification of runtime errors	SWS_WdgM_00030, SWS_WdgM_00031, SWS_WdgM_00142, SWS_WdgM_00319
SRS_BSW_00453	BSW Modules shall be harmonized	SWS_WdgM_NA_00345
SRS_BSW_00454	An alternative interface without a parameter of category DATA_REFERENCE shall be available.	SWS_WdgM_NA_00345
SRS_BSW_00456	A Header file shall be defined in order to harmonize BSW Modules	SWS_WdgM_NA_00345
SRS_BSW_00457	Callback functions of Application software components shall be invoked by the Basis SW	SWS_WdgM_NA_00345
SRS_BSW_00458	Classification of production errors	SWS_WdgM_00129, SWS_WdgM_00375, SWS_WdgM_00408
SRS_BSW_00459	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	SWS_WdgM_NA_00345
SRS_BSW_00460	Reentrancy Levels	SWS_WdgM_NA_00345
SRS_BSW_00461	Modules called by generic modules shall satisfy all interfaces requested by the generic module	SWS_WdgM_NA_00345
SRS_BSW_00462	All Standardized Autosar Interfaces shall have unique requirement Id / number	SWS_WdgM_NA_00345
SRS_BSW_00463	Naming convention of callout prototypes	SWS_WdgM_NA_00345
SRS_BSW_00464	File names shall be considered case sensitive regardless of the filesystem in which they are used	SWS_WdgM_NA_00345
SRS_BSW_00465	It shall not be allowed to name any two files so that	SWS_WdgM_NA_00345

	they only differ by the cases of their letters	
SRS_BSW_00466	Classification of extended production errors	SWS_WdgM_NA_00345
SRS_BSW_00467	The init / deinit services shall only be called by BswM or EcuM	SWS_WdgM_NA_00345
SRS_BSW_00469	Fault detection and healing of production errors and extended production errors	SWS_WdgM_00129, SWS_WdgM_00408
SRS_BSW_00470	Execution frequency of production error detection	SWS_WdgM_00129, SWS_WdgM_00408
SRS_BSW_00471	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	SWS_WdgM_00408
SRS_BSW_00472	Avoid detection of two production errors with the same root cause.	SWS_WdgM_NA_00345
SRS_BSW_00473	Classification of transient faults	SWS_WdgM_NA_00345
SRS_BSW_00477	The functional interfaces of AUTOSAR BSW modules shall be specified in C99	SWS_WdgM_NA_00345
SRS_BSW_00478	Timing limits of main functions	SWS_WdgM_NA_00345
SRS_BSW_00479	Interfaces for handling request from external devices	SWS_WdgM_NA_00345
SRS_BSW_00480	Null pointer errors shall follow a naming rule	SWS_WdgM_00004
SRS_BSW_00481	Invalid configuration set selection errors shall follow a naming rule	SWS_WdgM_00004
SRS_BSW_00482	Get version information function shall follow a naming rule	SWS_WdgM_NA_00345
SRS_BSW_00483	BSW Modules shall handle buffer alignments internally	SWS_WdgM_NA_00345
SRS_BSW_00484	Input parameters of scalar and enum types shall be passed as a value.	SWS_WdgM_NA_00345
SRS_BSW_00485	Input parameters of structure type shall be passed as a reference to a constant structure	SWS_WdgM_NA_00345
SRS_BSW_00486	Input parameters of array type shall be passed as a reference to the constant array base type	SWS_WdgM_NA_00345

SRS_BSW_00487	Errors for module initialization shall follow a naming rule	SWS_WdgM_00004
SRS_BSW_00488	Classification of security events	SWS_WdgM_NA_00345
SRS_BSW_00489	Reporting of security events	SWS_WdgM_NA_00345
SRS_BSW_00490	List possible security events	SWS_WdgM_NA_00345
SRS_BSW_00491	Specification of trigger conditions and context data	SWS_WdgM_NA_00345
SRS_BSW_00492	Reporting of security events during startup	SWS_WdgM_NA_00345
SRS_BSW_00493	Definition of security event ID symbols	SWS_WdgM_NA_00345
SRS_BSW_00494	ServiceInterface argument with a pointer datatype	SWS_WdgM_NA_00345
SRS_ModeMgm_09028	The Watchdog Manager shall support multiple watchdog instances	SWS_WdgM_00002
SRS_ModeMgm_09106	The list of entities supervised by the Watchdog Manager shall be configurable at pre-compile time	SWS_WdgM_00085
SRS_ModeMgm_09107	The Watchdog Manager shall provide an initialization service	SWS_WdgM_00018, SWS_WdgM_00135, SWS_WdgM_00151
SRS_ModeMgm_09109	It shall be possible to prohibit the disabling of watchdog	SWS_WdgM_00030, SWS_WdgM_00031
SRS_ModeMgm_09110	The watchdog Manager shall provide a service interface, to select a mode of the Watchdog Manager	SWS_WdgM_00139, SWS_WdgM_00154
SRS_ModeMgm_09112	The Watchdog Manager shall cyclically check the periodicity of the supervised entities	SWS_WdgM_00063, SWS_WdgM_00074, SWS_WdgM_00076, SWS_WdgM_00077, SWS_WdgM_00078, SWS_WdgM_00083, SWS_WdgM_00098, SWS_WdgM_00115, SWS_WdgM_00117, SWS_WdgM_00213, SWS_WdgM_00214, SWS_WdgM_00413
SRS_ModeMgm_09125	The Watchdog Manager shall provide a service allowing the Update temporal program flow monitoring	SWS_WdgM_00413, SWS_WdgM_00414
SRS_ModeMgm_09143	The Watchdog Manager shall set the triggering condition during inactive monitoring	SWS_WdgM_00083
SRS_ModeMgm_09158	The Watchdog Manager shall support Post build time and mode dependent	SWS_WdgM_00145

	selectable configuration sets for the Watchdog Manager	
SRS_ModeMgm_09159	The Watchdog Manager shall report failure of temporal or program flow monitoring to DEM	SWS_WdgM_00129, SWS_WdgM_00408
SRS_ModeMgm_09160	The Watchdog Manager shall provide the indication of failed temporal monitoring	SWS_WdgM_00148, SWS_WdgM_00150
SRS_ModeMgm_09161	The Watchdog Manager shall reset the triggering condition in the Watchdog Driver in Case of temporal failure	SWS_WdgM_00223
SRS_ModeMgm_09162	The Watchdog Manager shall be able to notify the software of an upcoming watchdog reset	SWS_WdgM_00150
SRS_ModeMgm_09163	It shall be possible to configure a delay before provoking a watchdog reset	SWS_WdgM_00077, SWS_WdgM_00215, SWS_WdgM_00219, SWS_WdgM_00220
SRS_ModeMgm_09169	The Watchdog Manager shall be able to immediately reset the MCU	SWS_WdgM_00133, SWS_WdgM_00134, SWS_WdgM_CONSTR_06500
SRS_ModeMgm_09221	The Watchdog Manager shall check the correct sequence of code execution in supervised entities	SWS_WdgM_00246, SWS_WdgM_00252, SWS_WdgM_00271, SWS_WdgM_00273, SWS_WdgM_00274, SWS_WdgM_00295, SWS_WdgM_00297, SWS_WdgM_00331
SRS_ModeMgm_09222	The Watchdog Manager shall provide a service allowing the Update logical program flow monitoring	SWS_WdgM_00246, SWS_WdgM_00252, SWS_WdgM_00271, SWS_WdgM_00273, SWS_WdgM_00274, SWS_WdgM_00295, SWS_WdgM_00297, SWS_WdgM_00331
SRS_ModeMgm_09225	The Watchdog Manager shall provide the indication of failed logical monitoring	SWS_WdgM_00148, SWS_WdgM_00150
SRS_ModeMgm_09226	The Watchdog Manager shall reset reset the triggering condition in the Watchdog Driver in Case of logical program flow violation	SWS_WdgM_00223
SRS_ModeMgm_09232	The Watchdog Manager shall provide a service to cause a watchdog reset	SWS_WdgM_00264

7 Functional Specification

This chapter presents the specification details of the internal functional behavior of the Watchdog Manager module.

7.1 Interaction of Supervision Functions

7.1.1 Overview

Supervised Entities are the units of supervision for the Watchdog Manager module. Each *Supervised Entity* can be supervised by a different *Supervision Function* or a combination of them.

The available *Supervision Functions* are:

- *Alive Supervision* (see Chapter 7.2.1)
- *Deadline Supervision* (see Chapter 7.2.2)
- *Logical Supervision* (see Chapter 7.2.3)

Each of three *Supervision Functions* results with a list of Results of *Supervision Function* for each *Supervised Entity* (highlighted in **Blue** on Figure 1), where each *Result* is either `correct` or `incorrect`. At Watchdog Manager initialization, all the Results are set to `correct`. This means that for every *Supervised Entity* there are three partial results (one from *Alive Supervision*, one from *Deadline Supervision* and one from *Logical Supervision*).

In a given *Mode*, each *Supervised Entity* may have zero, one or more *Alive Supervisions* (`WdgMAliveSupervision`), each having one `correct`/`incorrect` result.

In a given *Mode*, each *Supervised Entity* may have zero, one or more *Deadline Supervisions* (`WdgMDeadlineSupervision`), each having one `correct`/`incorrect` result.

Note: *Deadline Supervision* is the combination of the mechanisms for detection of:

- ***early arrivals:*** *End Checkpoint* reported before `WdgMDeadlineMin` since reporting of *Start Checkpoint*.
- ***delays***: *End Checkpoint* reported after `WdgMDeadlineMax` since reporting of Start Checkpoint.
- ***timeouts***: *End Checkpoint* not reported even after `WdgMDeadlineMax` since reporting of *Start Checkpoint*

In a given *Mode*, each *Supervised Entity* may have zero, one or more *Logical Supervision*s (i.e. *Graphs*) configured (`WdgMExternalLogicalSupervision` for one *External Graph*, a set of `WdgMInternalTransition`-s for one *Internal Graph*), each having one `correct`/`incorrect` result. Each *Logical Supervision* is for one *External* or *Internal Graph*.

In case there are zero active supervisions in a given *Mode*, then Main Function sees no EXPIRED local status, so `WdgIf_SetTriggerCondition` can be invoked.

Based on the results of *Supervision Functions* (`correct`/`incorrect`), the *Local Supervision Status* of each *Supervised Entity* (highlighted in **Green** on Figure 1) is determined by means of the *Local Supervision Status* state machine (see Chapter 7.1.2).

Based on *Local Supervision Status* of each *Supervised Entity*, the *Global Supervision Status* highlighted in **Red** on Figure 1) is determined by means of *Global Supervision Status* state machine (see Chapter 7.1.3)*.*

Based on the *Global Supervision Status*, the error handling (see Chapter 7.3) and watchdog handling (see Chapter 7.3) take place.

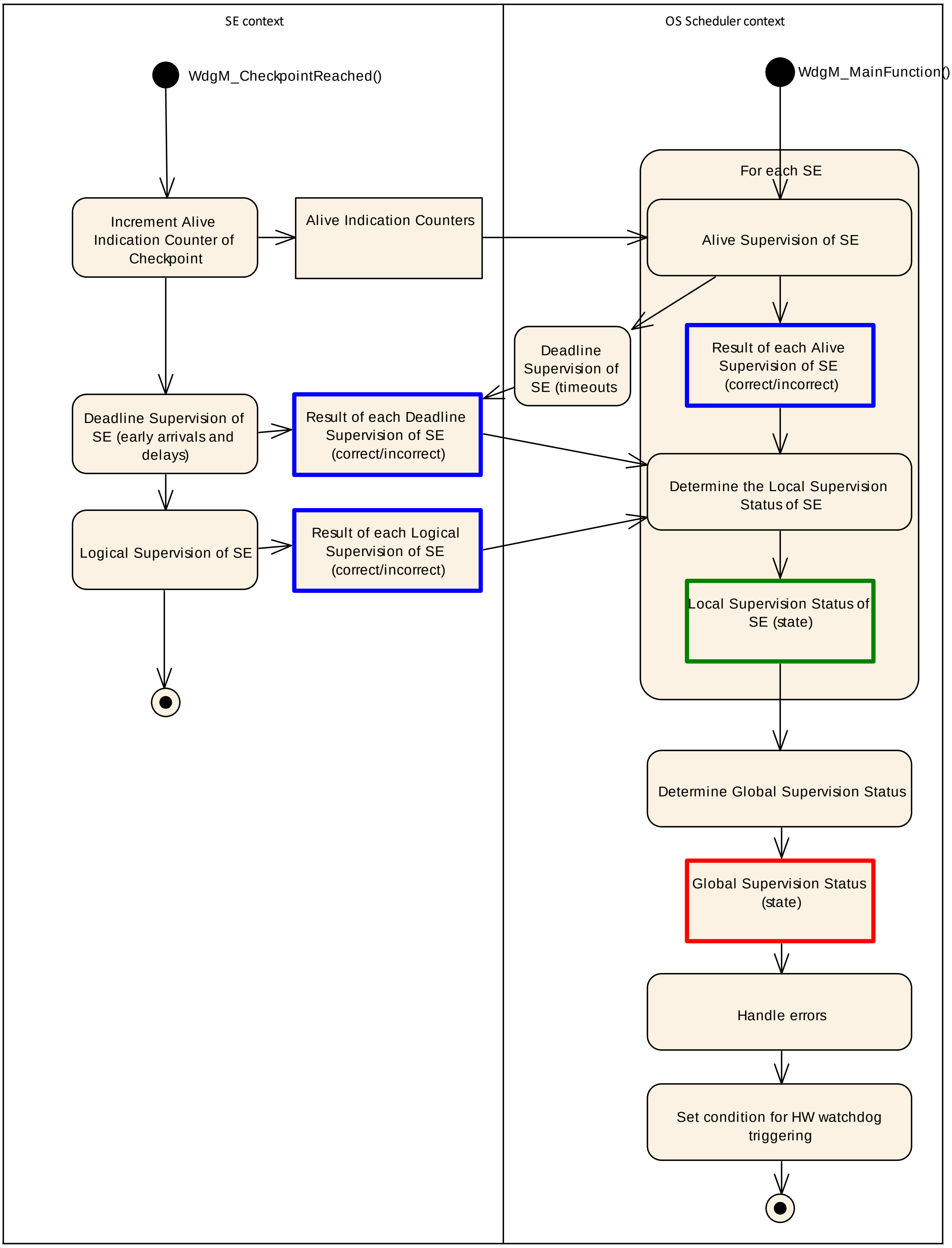
SE context
OS Scheduler context
WdgM_CheckpointReached()
WdgM_MainFunction()
For each SE
Increment Alive Indication Counter of Checkpoint
Alive Indication Counters
Alive Supervision of SE
Deadline Supervision of SE (timeouts
Result of each Alive Supervision of SE (correct/incorrect)
Deadline Supervision of SE (early arrivals and delays)
Result of each Deadline Supervision of SE (correct/incorrect)
Determine the Local Supervision Status of SE
Logical Supervision of SE
Result of each Logical Supervision of SE (correct/incorrect)
Local Supervision Status of SE (state)
Determine Global Supervision Status
Global Supervision Status (state)
Handle errors
Set condition for HW watchdog triggering

Figure 1: Overview of Watchdog Manager Supervision

The determination of supervision result for *Deadline Supervision* (detection of early arrivals and delays) and *Logical Supervision* is executed within the function `WdgM_CheckpointReached`. During one execution of this function, it updates the result for one particular *Supervised Entity* only.
The determination of supervision result for *Deadline Supervision* (timeout detection part) and *Alive Supervision* is executed within the function `WdgM_MainFunction`.

During one execution of this function, it updates the Results of *Deadline Supervision* (timeout detection part) and/or *Alive Supervision* for all *Supervised Entities*.

[SWS_WdgM_00406]⌈The WdgM module shall start both the *Supervision Functions* (for all *Supervision Algorithms*, including *Supervision Reference Cycles*) and the Watchdog Handling during the first invocation of the `WdgM_MainFunction` after the initialization of the module.⌋ (SRS_BSW_00450)

Note: If the WdgM module is not initialized, its Main Function will return immediately without performing any functionality and without raising any errors (see [SWS_BSW_00037]). Also, the module cannot use RTE APIs before first invocation of the Main Function (see [SWS_BSW_00218]). Therefore, the first call of the Main Function after initialization should be considered as the starting point of the *Supervision Functions* and the resulting handling of the hardware watchdog instances (using the WdgIf module), to have consistent behavior as a Safety-related Monitoring Mechanism.

[SWS_WdgM_00407]⌈The WdgM module shall stop the *Supervision Functions* (for all *Supervision Algorithms*) and Watchdog Handling in the `WdgM_DeInit`.⌋ (SRS_BSW_00450)

[SWS_WdgM_CONSTR_06510]⌈The following shall be available for the operation *Supervision Functions* of Watchdog Manager:

1. availability of initialized Wdg Interface,
2. availability of initialized OS,
3. initialized WdgM - by invocation of `WdgM_Init`() function, and
4. periodic invocation of `WdgM_MainFunction`() function. ⌋()

[SWS_WdgM_CONSTR_06511]⌈It shall be ensured by the callers of WdgM module, that the functions `WdgM_DeInit`, `WdgM_Init` and `WdgM_SetMode` are not invoked concurrently to the `WdgM_MainFunction`.⌋()

This can be achieved by the integrator by means of appropriate coordination of initialization and task scheduling.

{DRAFT} Note that, in the case of clustered software architecture (`WdgMSwClusterSupport` = ENABLE_SW_CLUSTER_SUPPORT), the `WdgM_MainFunction` instances in Application Software Clusters can be called at any time, regardless of the concurrent invocation of the functions `WdgM_DeInit`, `WdgM_Init` and `WdgM_SetMode` in the Host Software Cluster.

To be able to continue *Alive Supervision* and *Deadline Supervision* (timeout detection part) even if a *Supervised Entity* had a deadlock, each `WdgM_MainFunction` must be mapped to the tasks which don't contain *Supervised Entities* to be supervised by the `WdgM_MainFunction` instance.

[SWS_WdgM_CONSTR_00275] ⌈The OS task which is executing the main function `WdgM_MainFunction` shall be separated from the OS task(s) calling any function from a *Supervised Entity* under supervision.⌋()

7.1.2 Local Supervision Status

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

1. Previous value of the *Local Supervision Status*,
2. Current values of result of *Alive Supervision*, result of *Deadline Supervision*, result of *Logical Supervision*.

[SWS_WdgM_00409] {DRAFT} ⌈The *Local Supervision Status* state machine shall be calculated in every call of the function `WdgM_MainFunction` which the Supervised Entity is belonging to.⌋()

[SWS_WdgM_00410] {DRAFT} ⌈The state machine shall be initialized by the function `WdgM_Init`.⌋()

The Watchdog Manager module provides a feature to provide fault tolerance (corresponding to the local supervision status WDGM_LOCAL_STATUS_FAILED) for *Alive Supervision* for a configurable amount of (cumulative) time measured in multiples of the *Supervision Cycle* (*Supervision Cycle* is the period at which `WdgM_MainFunction` is called), named *Failed Supervision Reference Cycles* (see configuration parameter `WdgMFailedAliveSupervisionRefCycleTol`). If this parameter is set to 0, then there is no tolerance for *Alive Supervision* and then *Alive Supervision* behaves in the same way as *Deadline Supervision* and *Logical Supervision*, where the first incorrect result causes the transition to WDGM_LOCAL_STATUS_EXPIRED.

Note that, *Deadline* and *Logical Supervisions* will not be affected by `WdgMFailedAliveSupervisionRefCycleTol`.

[SWS_WdgM_00200] ⌈The Watchdog Manager module shall track the *Local Supervision Status* of each *Supervised Entity*.⌋()

Figure 2 shows the state machine for *Local Supervision Status* of a *Supervised Entity* with all possible states.

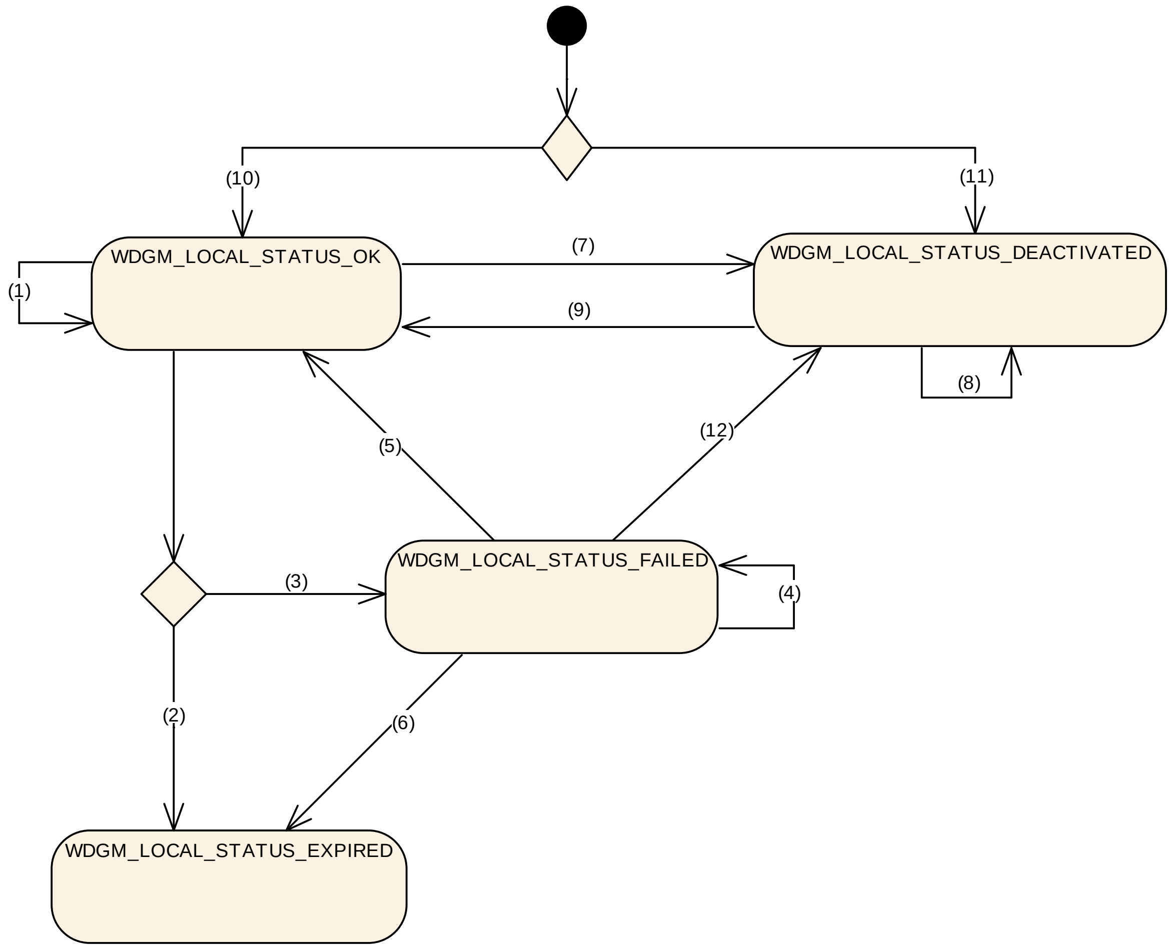

Figure 2: Local Supervision Status

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

[SWS_WdgM_00268] ⌈If the function `WdgM_Init` is successfully called, then for each *Supervised Entity* that is referenced from the *Initial Mode* (`WdgMInitialMode`) (i.e. each *Supervised Entity* that is activated in the *Initial Mode*), the function `WdgM_Init` shall set the *Local Supervision Status* for this *Supervised Entity* to WDGM_LOCAL_STATUS_OK. And the counter for *Failed Supervision Reference Cycles* shall be set to zero (0). (see Transition (10) in Figure 2).⌋(SRS_BSW_00101)

[SWS_WdgM_00269] ⌈If the function `WdgM_Init` is successfully called, then for each *Supervised Entity* that is not referenced from the *Initial Mode* (`WdgMInitialMode`), the function `WdgM_Init` shall set the *Local Supervision*

Status for this *Supervised Entity* to WDGM_LOCAL_STATUS_DEACTIVATED (see Transition (11) in Figure 2).

If the function `WdgM_Init` is successfully called and the parameter `WdgMInitialMode` [ECUC_WdgM_00336] of this *Supervised Entity* in `WdgMInitialMode` is **not** configured to `WDGM_LOCAL_STATUS_OK` then the Watchdog Manager module shall set the *Local Supervision Status* for this *Supervised Entity* to WDGM_LOCAL_STATUS_DEACTIVATED. (see Transition (11) in Figure 2)⌋(SRS_BSW_00101)

[SWS_WdgM_00201] ⌈If all values in three sets of results of *Supervision* (results of *Alive Supervision*, results of *Deadline Supervision*, results of *Logical Supervision*) for the *Supervised Entity* are `correct` and the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_OK, then the function `WdgM_MainFunction` shall keep the *Supervised Entity* in the *Local Supervision Status* WDGM_LOCAL_STATUS_OK (see Transition (1) in Figure 2).⌋()

[SWS_WdgM_00202] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_OK AND:

1. (At least one result of *Alive Supervision* of the *Supervised Entity* is `incorrect` and a *Failure Tolerance* of zero is configured (see configuration parameter `WdgMFailedAliveSupervisionRefCycleTol` [ECUC_WdgM_00327]) OR
2. If the result of at least one *Deadline Supervision* of the *Supervised Entity* or the result of at least one *Logical supervision* of the *Supervised Entity* is `incorrect`),

THEN the function `WdgM_MainFunction` shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_EXPIRED (see Transition (2) in Figure 2).⌋()

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

[SWS_WdgM_00203] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_OK AND:

1. (If the result of at least one *Alive Supervision* of the *Supervised Entity* is `incorrect` and a *Failure Tolerance* greater than zero is configured (see configuration parameter `WdgMFailedAliveSupervisionRefCycleTol` [ECUC_WdgM_00327]) AND
2. If all the results of *Deadline Supervision* of the *Supervised Entity* and all results of *Logical Supervision* of the *Supervised Entity* are `correct`),

THEN the function WdgM_MainFunction shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_FAILED and increment the counter for *Failed Supervision Reference Cycles* (see Transition (3) in Figure 2).⌋()

[SWS_WdgM_00204] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_FAILED AND:

1. (If the result of at least one *Alive Supervision* is incorrect and the counter for *failed supervision reference cycles* is less than the configured *Failure Tolerance* (see parameter WdgMFailedAliveSupervisionRefCycleTol [ECUC_WdgM_00327]) AND
2. If all the results of *Deadline Supervisions* of the *Supervised Entity* and all the result of *Logical Supervision* of the *Supervised Entity* are correct),

THEN the function WdgM_MainFunction shall keep the *Local Supervision Status* in WDGM_LOCAL_STATUS_FAILED and increment the counter for *Failed Supervision Reference Cycles* (see Transition (4) in Figure 2).⌋()

[SWS_WdgM_00300] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_FAILED AND:

1. (If all the results of *Alive Supervision* of the *Supervised Entity* are correct and the counter for *Failed Supervision Reference Cycles* is > 1) AND
2. If all the result of *Deadline Supervision* of the *Supervised Entity* and all the result of *Logical Supervision* of the *Supervised Entity* are correct),

THEN the function WdgM_MainFunction shall keep the *Local Supervision Status* in WDGM_LOCAL_STATUS_FAILED and decrement the counter for *Failed Supervision Reference Cycles* (see Transition (4) in Figure 2).⌋()

[SWS_WdgM_00205] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_FAILED AND:

1. (If all the results of *Alive Supervision* of the *Supervised Entity* are correct and the counter for *Failed Supervision Reference Cycles* equals 1) AND
2. If all the results of *Deadline Supervisions* of the *Supervised Entity* and all the results of *Logical Supervision* of the *Supervised Entity* are correct),

THEN the function WdgM_MainFunction shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_OK and decrement the counter for *Failed Supervision Reference Cycles* (see Transition (5) in Figure 2).⌋()

[SWS_WdgM_00206] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_FAILED AND:

1. (If at least one result of *Alive Supervision* is incorrect and the counter for *Failed Supervision Reference Cycles* is equal to the configured *Failure Tolerance* (see configuration parameter WdgMFailedAliveSupervisionRefCycleTol [ECUC_WdgM_00327]) OR
2. If at least one result of *Deadline Supervision* of the *Supervised Entity* or at least one the result of *Logical Supervision* of the *Supervised Entity* is incorrect),

THEN the function WdgM_MainFunction shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_EXPIRED (see Transition (6) in Figure 2).⌋()

[SWS_WdgM_00207] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_OK and if a call of WdgM_SetMode switches to a mode which deactivates the *Supervised Entity* (see [SWS_WdgM_00283]), then the Watchdog Manager module shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_DEACTIVATED (see Transition (7) in Figure 2).⌋()

[SWS_WdgM_00291] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_FAILED and if a call of WdgM_SetMode switches to a mode in which the *Supervised Entity* is Deactivated (see [SWS_WdgM_00283]), then the Watchdog Manager module shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_DEACTIVATED (see Transition (12) in Figure 2).⌋()

Note that the above requirement is only applicable for the WDGM_LOCAL_STATUS_FAILED status, but not for WDGM_LOCAL_STATUS_EXPIRED.

[SWS_WdgM_00208] ⌈If the *Supervised Entity* was in the *Local Supervision Status* WDGM_LOCAL_STATUS_DEACTIVATED, the functions WdgM_CheckpointReached and WdgM_MainFunction shall not perform any *Supervision Functions* for this *Supervised Entity* and keep the *Local Supervision Status* in the state WDGM_LOCAL_STATUS_DEACTIVATED. (see Transition (8) in Figure 2)⌋()

[SWS_WdgM_00209] ⌈If the *Supervised Entity* was in *Local Supervision Status* WDGM_LOCAL_STATUS_DEACTIVATED and if a call of WdgM_SetMode switches to a mode in which the *Supervised Entity* is active (see [SWS_WdgM_00282]), then the Watchdog Manager module shall change the *Local Supervision Status* to WDGM_LOCAL_STATUS_OK. And the counter for *Failed Supervision Reference Cycles* shall be set to zero (0). (see Transition (9) in Figure 2)⌋()

7.1.3 Global Supervision Status

Based on the *Local Supervision Status* of all *Supervised Entities*, the *Global Supervision Status* is computed.

The *Global Supervision Status* has similar values as the *Local Supervision Status*. The main differences are the addition of the WDGM_GLOBAL_STATUS_STOPPED value. Figure 3 shows the values and transitions between them.

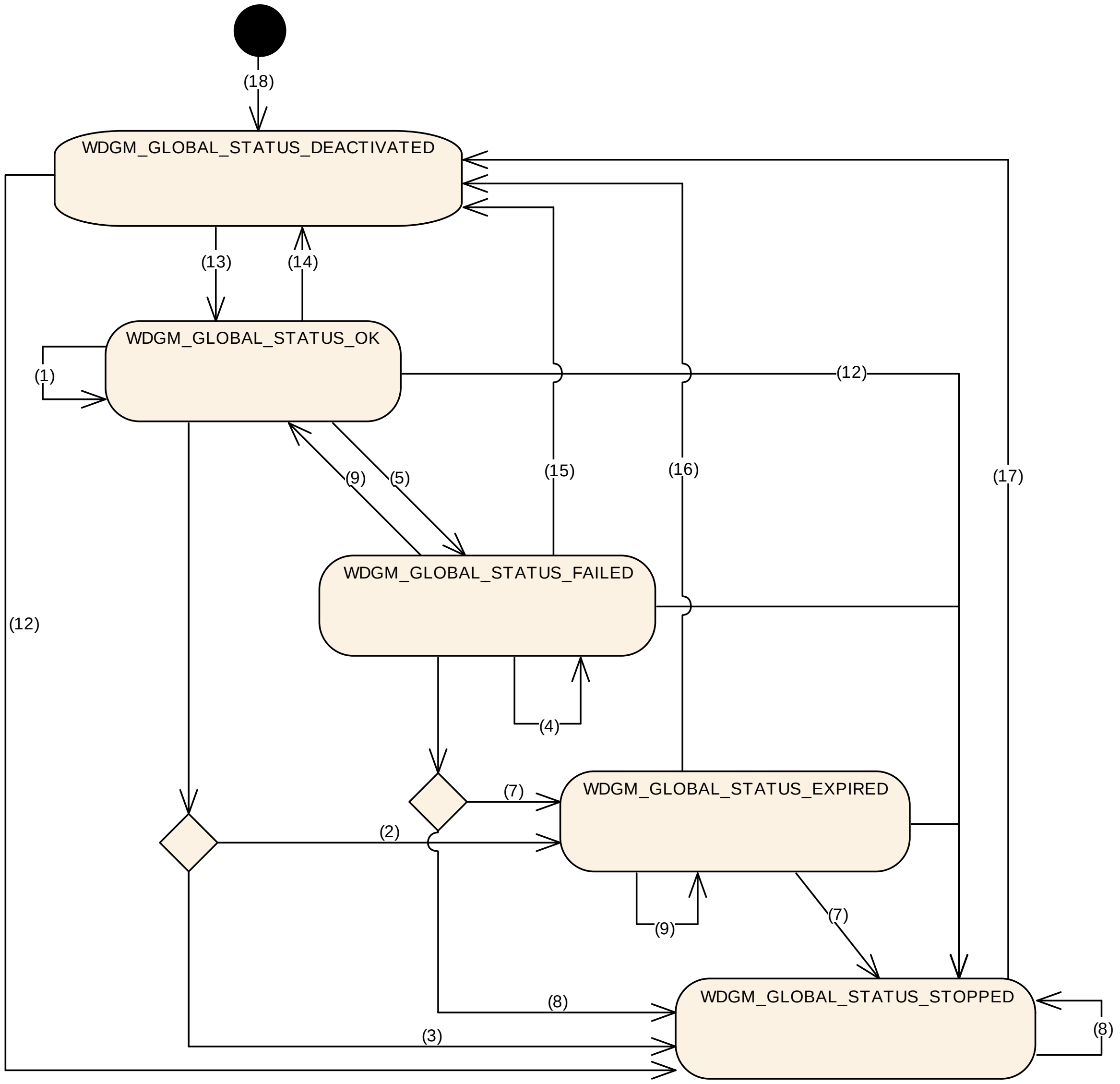
(18)
WDGM_GLOBAL_STATUS_DEACTIVATED
(13)
(14)
WDGM_GLOBAL_STATUS_OK
(1)
(12)
(9)
(5)
(15)
(16)
(17)
WDGM_GLOBAL_STATUS_FAILED
(12)
(4)
(7)
WDGM_GLOBAL_STATUS_EXPIRED
(2)
(9)
(7)
(8)
WDGM_GLOBAL_STATUS_STOPPED
(3)
(8)

Figure 3: Global Supervision Status

[SWS_WdgM_00213] ⌈The Watchdog Manager module shall have one *Global Supervision Status* for the whole monitored software.⌋(SRS_ModeMgm_09112)

[SWS_WdgM_00387] ⌈ *Global Supervision Status* shall be statically initialized with WDGM_GLOBAL_STATUS_DEACTIVATED (see Transition (18) in Figure 4). ⌋()

The Watchdog Manager module provides a feature to postpone the error reaction (the error reaction being not setting a correct trigger condition) for a configurable amount of time measured in multiples of the *Supervision Cycle*, named *Expired Supervision Tolerance* (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]).
The *Expired Supervision Tolerance* is implemented within the state machine of the *Global Supervision Status*. The defined state machine is in the state WDGM_GLOBAL_STATUS_EXPIRED while the blocking is postponed.

[SWS_WdgM_00214] {DRAFT} ⌈The function `WdgM_MainFunction` shall calculate the *Global Supervision Status* in every *Main Function Period*. The function shall compute the *Global Supervision Status* after computation of every *Local Supervision Status*.⌋(SRS_ModeMgm_09112)

The cyclic update of *Global Supervision Status* is necessary to trigger the timely transition from WDGM_GLOBAL_STATUS_EXPIRED to WDGM_GLOBAL_STATUS_STOPPED.

{DRAFT} Note that, in case of clustered software architecture, multiple `WdgM_MainFunction` instances may exist. In this case, Global Supervision Status can be updated every call of any `WdgM_MainFunction` instance.

Following rules shall be used to calculate the *Global Supervision Status*:

[SWS_WdgM_00285] ⌈If the function `WdgM_Init` [SWS_WdgM_00151] was successfully called then the function shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_OK. And the *Expired Cycle Counter* shall be set to zero (0). (see Transition (13) in Figure 4).⌋(SRS_BSW_00101)

[SWS_WdgM_00286] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_OK and the function `WdgM_DeInit` [SWS_WdgM_00261] is successfully called, then the function shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_DEACTIVATED (see Transitions (14), (15), (16) and (17) in Figure 4).⌋()

It has to be considered carefully that a deactivation of WdgM when it is in states WDGM_GLOBAL_STATUS_EXPIRED or WDGM_GLOBAL_STATUS_STOPPED can hinder error reporting or error reaction.

[SWS_WdgM_00078] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_OK and the *Local Supervision Status* of all *Supervised Entities* are either WDGM_LOCAL_STATUS_OK or WDGM_LOCAL_STATUS_DEACTIVATED then the function `WdgM_MainFunction` shall keep the *Global Supervision Status* WDGM_GLOBAL_STATUS_OK (see Transition (1) in Figure 3).⌋(SRS_ModeMgm_09112)

[SWS_WdgM_00076] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_OK, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_FAILED, and no *Supervised Entity* is in *Local Supervision Status* WDGM_LOCAL_STATUS_EXPIRED, then the function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_FAILED (see Transition (2) in Figure 3).⌋(SRS_ModeMgm_09112)

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

[SWS_WdgM_00215] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_OK, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_EXPIRED, and the *Expired Supervision Tolerance* is configured to a value larger than zero (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]), then function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_EXPIRED. And increment the *Expired Cycle Counter*. (see Transition (3) in Figure 3).⌋(SRS_ModeMgm_09163)

[SWS_WdgM_00216] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_OK, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_EXPIRED, and the *Expired Supervision Tolerance* is configured to zero (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]), then the function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_STOPPED (see Transition (4) in Figure 3).⌋()

[SWS_WdgM_00217] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_FAILED, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_FAILED, and no *Supervised Entity* is in *Local Supervision Status* WDGM_LOCAL_STATUS_EXPIRED, then function `WdgM_MainFunction` shall remain in *Global Supervision Status* WDGM_GLOBAL_STATUS_FAILED. (see Transition (5) in Figure 3)⌋()

[SWS_WdgM_00218] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_FAILED and the *Local Supervision Status* of all *Supervised Entities* is either WDGM_LOCAL_STATUS_OK or WDGM_LOCAL_STATUS_DEACTIVATED then function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_OK (see Transition (6) in Figure 3).⌋()

[SWS_WdgM_00077] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_FAILED, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_EXPIRED, and the *Expired Supervision Tolerance* is configured to a value larger than zero (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]), then function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_EXPIRED. And increment the *Expired Cycle Counter*. (see Transition (7) in Figure 3).⌋(SRS_ModeMgm_09112, SRS_ModeMgm_09163)

[SWS_WdgM_00117] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_FAILED, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_EXPIRED, and the *Expired Supervision Tolerance* is configured to zero (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]), then function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_STOPPED (see Transition (8) in Figure 3).⌋(SRS_ModeMgm_09112)

[SWS_WdgM_00219] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_EXPIRED, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_EXPIRED, and the *Expired Cycle Counter* is less than the configured *Expired Supervision Tolerance* (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]), then function `WdgM_MainFunction` shall keep *Global Supervision Status* WDGM_GLOBAL_STATUS_EXPIRED and increment the *Expired Cycle Counter* (see Transition (9) in Figure 3).⌋(SRS_ModeMgm_09163)

[SWS_WdgM_00220] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_EXPIRED, the *Local Supervision Status* of at least one *Supervised Entity* is WDGM_LOCAL_STATUS_EXPIRED, and the *Expired Cycle Counter* is equal to the configured *Expired Supervision Tolerance* (see configuration parameter `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]), then function `WdgM_MainFunction` shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_STOPPED (see Transition (10) in Figure 3).⌋(SRS_ModeMgm_09163)

[SWS_WdgM_00221] ⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_STOPPED, then function `WdgM_MainFunction` shall remain in *Global Supervision Status* WDGM_GLOBAL_STATUS_STOPPED (see Transition (11) in Figure 3).⌋()

[SWS_WdgM_00139] ⌈If a call to `WdgIf_SetMode` fails (see chapter 7.5.2), function shall assume a global supervision failure and set the *Global Supervision Status* to WDGM_GLOBAL_STATUS_STOPPED. (see Transition (12) in Figure 9) ⌋(SRS_ModeMgm_09110)

This is the final state and the failure recovery mechanisms will be started. Usually a watchdog reset will occur after the hardware watchdog has expired.

7.2 Supervision Functions

[SWS_WdgM_00413] {DRAFT} ⌈ *Alive Supervision* and *Deadline Supervision* (timeout detection part) for each *Supervised Entity* shall be executed within the corresponding Main Function instance, which is identified by `WdgMMainFunctionPartitionRef`. ⌋ (SRS_ModeMgm_09112, SRS_ModeMgm_09125)

[SWS_WdgM_00063] {DRAFT} ⌈If the *Global Supervision Status* is not in the state WDGM_GLOBAL_STATUS_DEACTIVATED, then the `WdgM_MainFunction()` shall execute *Alive Supervision* according to the configured *Supervision Cycle*.⌋(SRS_ModeMgm_09112)

[SWS_WdgM_00414] {DRAFT} ⌈If the *Global Supervision Status* is not in the state WDGM_GLOBAL_STATUS_DEACTIVATED, then the `WdgM_MainFunction()` shall execute *Deadline Supervision* (timeout detection part) according to the configured *Main Function Period*. ⌋(SRS_ModeMgm_09125)

7.2.1 Alive Supervision

Alive Supervision is one of the *Supervision Functions* of the Watchdog Manager module. The *Alive Supervision* offers a mechanism to periodically check the execution reliability of one or several *Supervised Entities*. This mechanism supports a check of cyclic timing constraints of independent *Supervised Entities*.

7.2.1.1 Alive Supervision Configuration

To provide *Alive Supervision*, the *Checkpoints* and their timing constraints need to be configured. The simplest configuration for *Alive Supervision* is one *Checkpoint* without any *Transitions*, as shown in Figure 4.

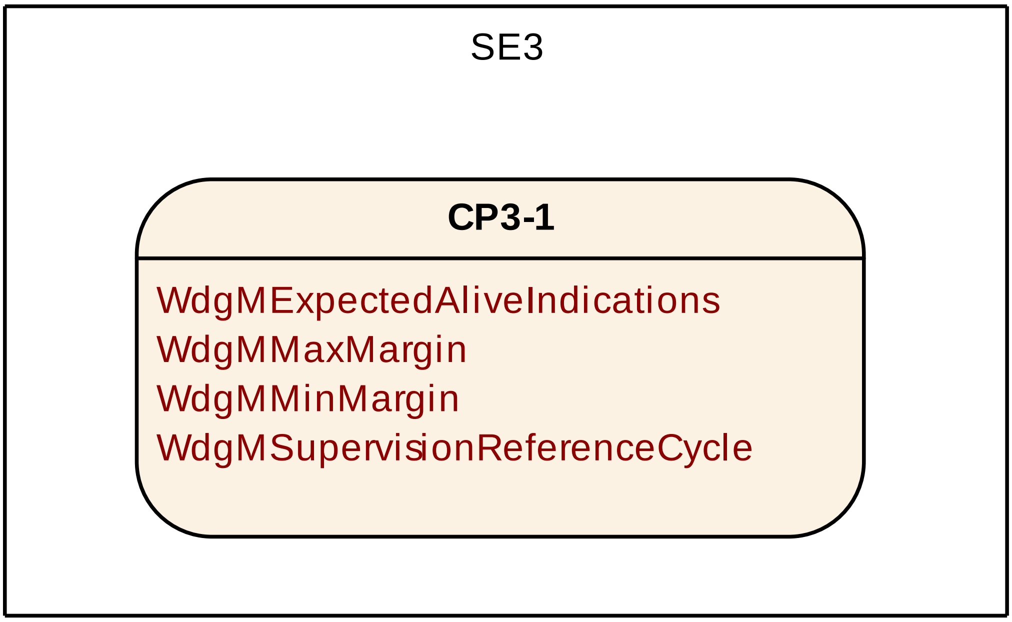

Figure 4: Simplest Alive Supervision Checkpoint Configuration

The above configuration provides backward compatibility to *Alive Supervision* as defined in versions before AUTOSAR Classic Platform R4.0.1, where each *Supervised Entity* could be supervised with one set of parameters only.

Moreover, it is also possible to have more than one *Checkpoint* as shown in Figure 5.

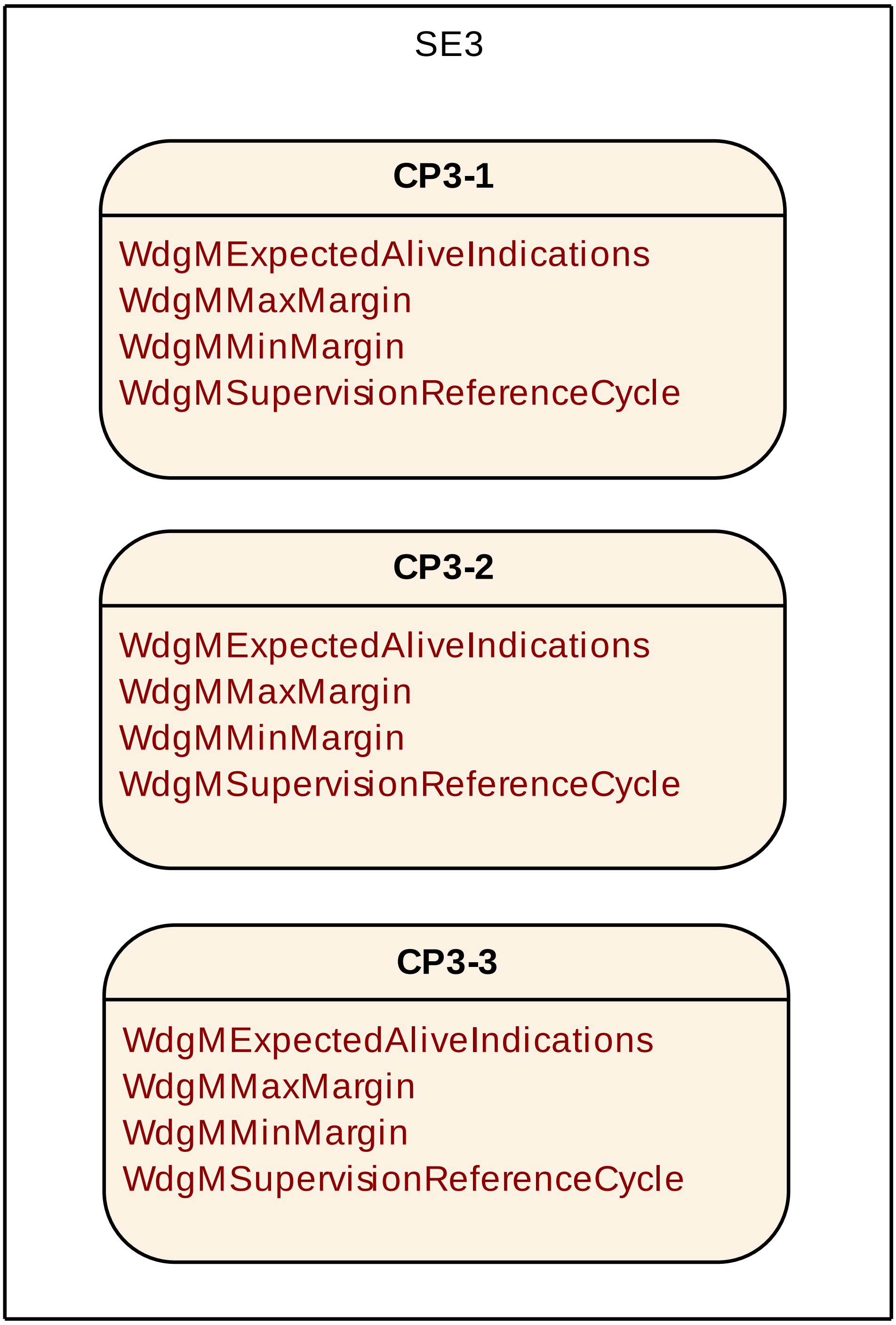
SE3
CP3-1
WdgMExpectedAliveIndications
WdgMMaxMargin
WdgMMinMargin
WdgMSupervisionReferenceCycle
CP3-2
WdgMExpectedAliveIndications
WdgMMaxMargin
WdgMMinMargin
WdgMSupervisionReferenceCycle
CP3-3
WdgMExpectedAliveIndications
WdgMMaxMargin
WdgMMinMargin
WdgMSupervisionReferenceCycle

Figure 5: Multiple Checkpoints for Alive Supervision in one Supervised Entity

Each *Checkpoint* has its own set of *Alive Supervision Parameters*. *Transitions* are not used by *Alive Supervision*. Although each *Checkpoint* has its own parameters, it is the *Supervised Entity* for which status is determined based on the frequency of *Checkpoints*.

The parameters of the *Alive Supervision* (see WdgMAliveSupervision) depend on the Watchdog Manager *Mode* and are defined for per *Checkpoint* (and not globally for the whole *Supervised Entity*).

None, some, or all of the *Checkpoints* of a *Supervised Entity* can be configured for *Alive Supervision* in a given *Mode*. Moreover, in each *Mode* the *Alive Supervision* options of *Checkpoints* can be different.

The WdgMExpectedAliveInidications [ECUC_WdgM_00311] (EAI) specifies the amount of expected alive indications from a given *Checkpoint*, within a fixed period of *Supervision Cycles*.

An acceptable negative variation (WdgMMinMargin [ECUC_WdgM_00312]) and acceptable positive variation (WdgMMaxMargin [ECUC_WdgM_00313]) can be configured.

The Watchdog Manager module has to support a configurable amount of independent *Supervised Entities*. As a consequence, the following general issue has to be considered.

[SWS_WdgM_00085] ⌈The Watchdog Manager module shall derive the required number of independent data resources to perform the *Alive Supervision* within the Watchdog Manager module from the number of *Supervised Entities*, number of WdgMModes and their WdgMAliveSupervisions.⌋(SRS_ModeMgm_09106)

Examples of independent data resources in context of the Watchdog Manager module are: *Alive Counters*, *Supervision Cycles* counters, *Failed Supervision Reference Cycles* counters, *Expired Supervision Cycles* counters, *Local Supervision Status*.

7.2.1.2 Alive Supervision Algorithm

To send an *Alive Indication*, a *Supervised Entity* invokes the function WdgM_CheckpointReached, which results with incrementation of an *Alive Counter* for the *Checkpoint*.

Alive Supervision is performed by counting the number of reports from *Supervised Entities* (by WdgM_CheckpointReached) during a configurable period.

This Supervision is executed by WdgM_MainFunctions with configurable cycle times. The cyclic examination of the Counter of each *Checkpoint* of a *Supervised Entity* by the Main Function happens at every *Supervision Reference Cycle* (which is a multiple of *Supervision Cycle*).

The *Supervision Cycle* and *Supervision Reference Cycle* (see WdgMSupervisionReferenceCycle) are the properties of an *Alive Supervision* of a *Checkpoint* in a given Watchdog Manager *Mode*.

[SWS_WdgM_00098] ⌈The function WdgM_MainFunction shall perform for each *Alive Supervision* (WdgMAliveSupervision) configured in the active *Mode*, the examination of the *Alive Counter* of each *Checkpoint* of the *Supervised Entity*. The examination shall be done at the period WdgMSupervisionReferenceCycle of the corresponding *Alive Supervision* (WdgMAliveSupervision).⌋(SRS_ModeMgm_09112)

Note: During the intermediate *Supervision Cycles of the Alive Supervision,* the function WdgM_MainFunction does not perform the examination of *Alive Counters*.

[SWS_WdgM_00074] ⌈The function WdgM_MainFunction shall examine an *Alive Counter* by checking if it is within the allowed tolerance (Expected – Min Margin; Expected + Max Margin) (see WdgMExpectedAliveIndications [ECUC_WdgM_00311], WdgMMinMargin, WdgMMaxMargin).⌋(SRS_ModeMgm_09112)

If any *Checkpoint* of a *Supervised Entity* fails the examination, then the result of *Alive Supervision* for the *Supervised Entity* is set to incorrect.

[SWS_WdgM_00115] ⌈If the function WdgM_MainFunction detects a deviation between the counted *Alive Indications* and the expected amount of *alive indications* [ECUC_WdgM_00311] (including tolerance margins [ECUC_WdgM_00312], [ECUC_WdgM_00313]) for any *Checkpoint* of a *Supervised Entity*, then *Alive Supervision* at this *Supervision Reference Cycle* for this *Supervised Entity* shall be defined as incorrect. Otherwise, it shall be defined as correct. ⌋(SRS_ModeMgm_09112)

If a *Checkpoint* is not Alive-Supervised in a mode, then it is ignored by Watchdog Manager.

[SWS_WdgM_00083] ⌈The function WdgM_MainFunction shall not perform the examination of the *Alive Counter of a Checkpoint* if no corresponding *Alive Supervision* (WdgMAliveSupervision) is defined in the active Watchdog Manager *Mode*.⌋(SRS_ModeMgm_09112, SRS_ModeMgm_09143)

7.2.2 Deadline Supervision

Deadline Supervision checks the timing constraints of non-cyclic *Supervised Entities*. In these *Supervised Entities*, a certain event happens and a following event happens within a given time span. This time span can have a maximum and minimum deadline (time window).

7.2.2.1 Deadline Supervision Configuration

For every *Deadline Supervision*, two *Checkpoints* connected by a *Transition* are configured. The *Deadline* is attached to the *Transition* from the start *Checkpoint* to the *End Checkpoint*. The simplest *Deadline Supervision* configuration contains two *Checkpoints* and one *Transition*, as shown in Figure 6.

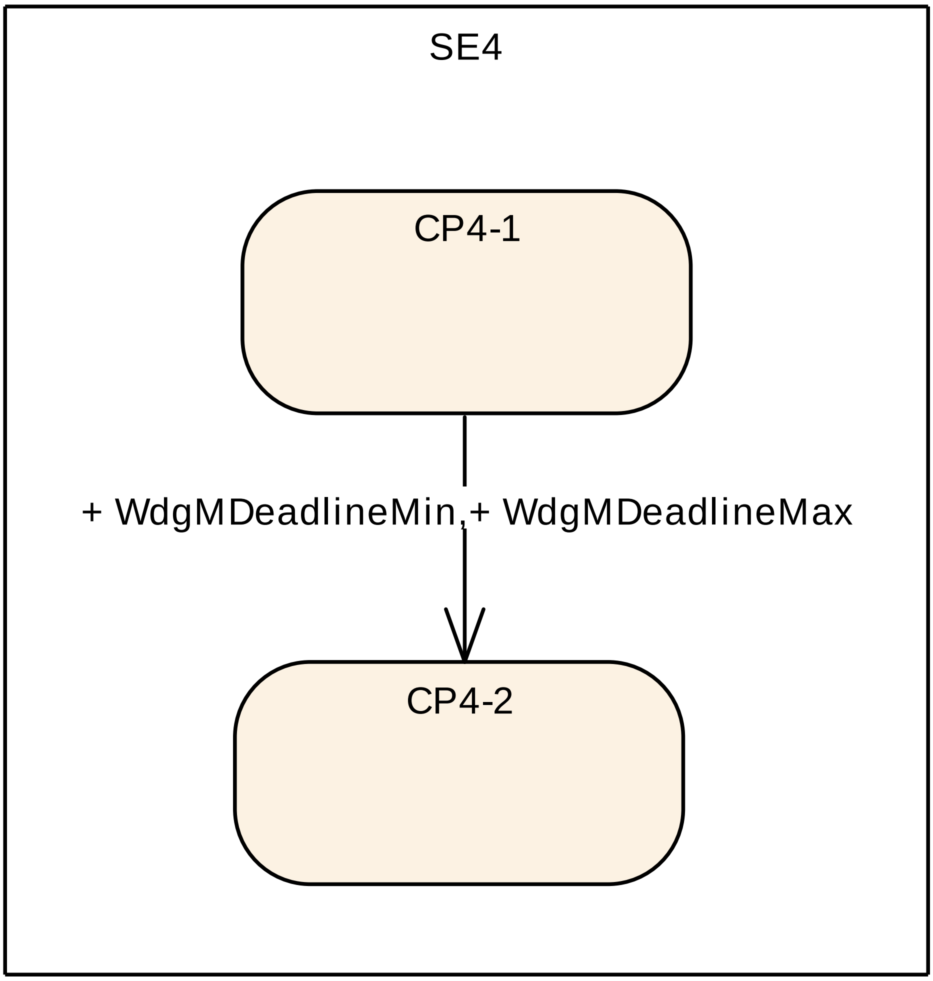

Figure 6: Simplest Deadline Supervision Configuration

More than one *Transition* can be defined in a *Supervised Entity*. The *Transitions* and *Checkpoints* do not have to form a closed graph. Since only the *Start* and *End Checkpoints* are considered by this *Supervision Function*, there can be independent graphs, as shown in Figure 7. Moreover, the *Checkpoints* can be chained.

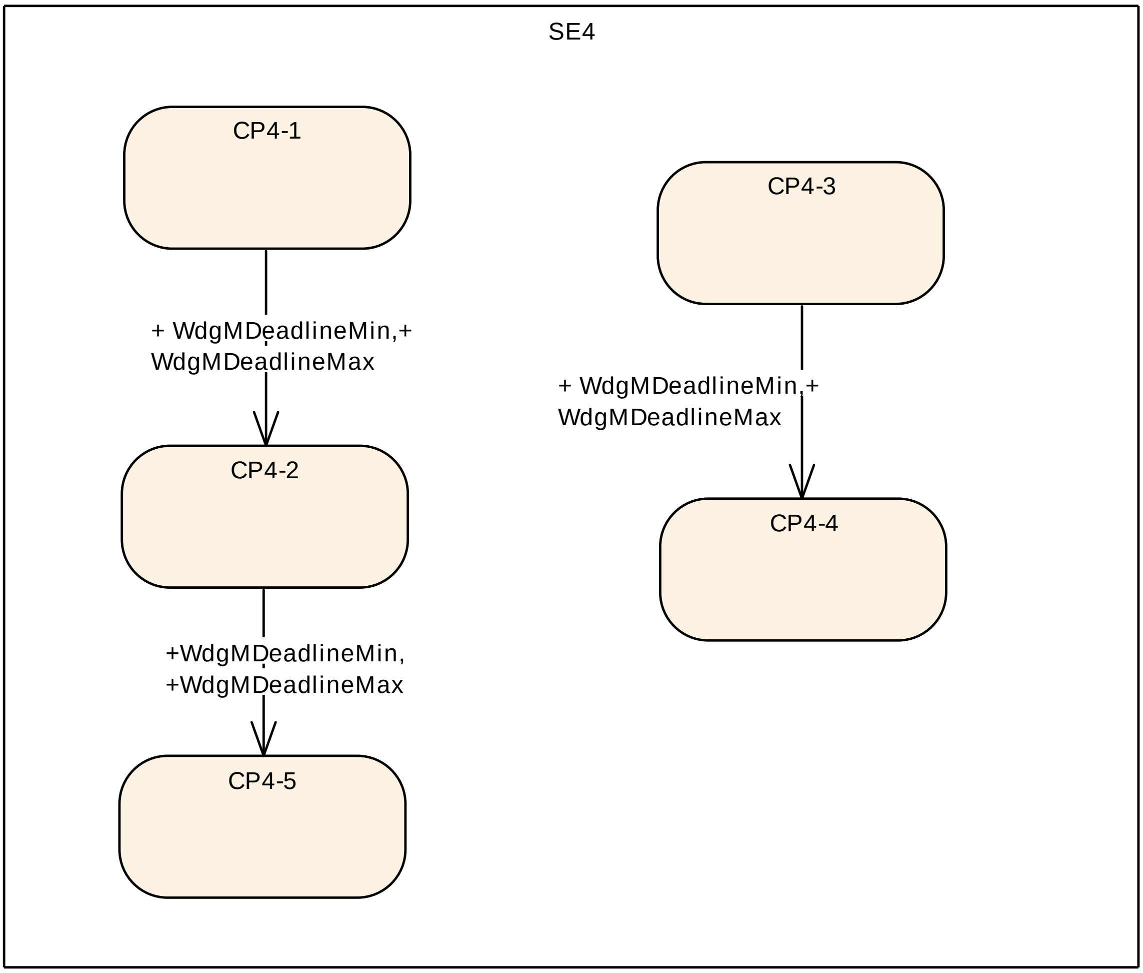

Figure 7: Multiple Transitions for Deadline Supervision in one Supervised Entity

The configuration of *Deadline Supervision* is similar to the one of *Alive Supervision*.

The parameters of the *Deadline Supervision* (see `WdgMDeadlineSupervision`) depend on the Watchdog Manager *Mode* (`WdgMMode`) and are defined for per a set of two *Checkpoints*. None, some, or all of the *Checkpoints* of a *Supervised Entity* can be configured for *Deadline Supervision* in a given *Mode*.

A *Deadline Supervision* is defined as a set of *Transitions* with time constraints. A *Transition* is defined as two references to two *Checkpoints*, called *Deadline Start Checkpoint* and *Deadline End Checkpoint* (`WdgMDeadlineStartRef` and `WdgMDeadlineEndRef`). A *Transition* has minimum and maximum time (`WdgMDeadlineMin` [ECUC_WdgM_00317], `WdgMDeadlineMax` [ECUC_WdgM_00318]).

[SWS_WdgM_00293] ⌈The Watchdog Manager module shall derive the required number of independent data resources to perform the Deadline Supervision within the Watchdog Manager module from the number of *Supervised Entities*, number of WdgMModes and their WdgMDeadlineSupervisions.⌋()

7.2.2.2 Deadline Supervision Algorithm

For each *Deadline Start Checkpoints* (i.e. *Checkpoint* referenced by WdgMDeadlineStartRef), Watchdog Manager has a timestamp variable storing the time when that *Checkpoint* has been reached.

A timestamp variable for *Deadline Supervision* is obtained by reading OS tick. For each *Supervised Entity,* an OS counter is configured.

An OS counter can be shared between *Supervised Entities*, or a separate OS counter can be used for each *Supervised Entity* (implementation-specific). In case OS-Applications/partitioning is used and a counter is shared across *Supervised Entities* belonging to different OS-applications, then the list of allowed OS-Applications to access the counter needs to be configured (OsCounterAccessingApplication).

[SWS_WdgM_CONSTR_06513] ⌈For each *Supervised Entity*, an OS counter shall be configured (see WdgMOSCounter, ECUC_WdgM_00361) if at least one *Deadline Supervision* is configured for the *Supervised Entity* in any of the Watchdog Manager *Modes*.⌋()

[SWS_WdgM_CONSTR_06514] ⌈The OS counters for each *Supervised Entity* shall be configured to be accessible from the OsApplication which contains the *Supervised Entity*.⌋()

[SWS_WdgM_CONSTR_06515] ⌈The OS counters for each *Supervised Entity* shall be configured to be also accessible from the OsApplication which calls WdgM_MainFunction, if WdgMEnableTimeoutDetection is set to true.⌋()

[SWS_WdgM_00373] ⌈To determine the timestamp and to compute the timestamp differences, the function WdgM_CheckpointReached shall use OS function GetElapsedValue, using as 1st parameter the CounterID that is configured for the *Supervised Entity*.

To determine the timestamp and to compute the timestamp differences, the function WdgM_CheckpointReached (for detection of both early arrivals and delays) and the function WdgM_MainFunction (for detection of timeouts) shall use OS function GetElapsedValue, using as 1st parameter the CounterID that is configured for the *Supervised Entity* (see WdgMOSCounter, ECUC_WdgM_00361) ⌋ (RS_HM_09235)

The timestamps are in ticks. However, the Watchdog deadline configuration is in seconds. The scaling between ticks and seconds is configured in OS.

[SWS_WdgM_00374] ⌈For scaling of timestamp difference to the limit values (`WdgMDeadlineMin` and `WdgMDeadlineMax`) (see SWS_WdgM_00294), the function `WdgM_CheckpointReached` (for detection of early arrivals and delays) and the function `WdgM_MainFunction` (for detection of timeouts) shall use `OsSecondsPerTick` configuration parameter.⌋(RS_HM_09235)

During the initialization, all the timestamps of *Deadline Start Checkpoints* (i.e. *Checkpoint* referenced by `WdgMDeadlineStartRef`) are cleared – set to 0.

[SWS_WdgM_00298] ⌈The function `WdgM_Init` shall for all Deadline Start *Checkpoints* set their timestamps to 0.⌋(SRS_BSW_00101)

When a *Deadline Start Checkpoint* (i.e. *Checkpoint* referenced by `WdgMDeadlineStartRef`) is reached, a *Supervised Entity* invokes the function `WdgM_CheckpointReached`, which results with the execution of *Deadline Supervision*.

[SWS_WdgM_00228] ⌈When the *Deadline Start Checkpoint* is reached and this *Checkpoint* is referenced in the active *Mode,* then the function `WdgM_CheckpointReached` shall record the current timestamp under the timestamp of the reached *Deadline Start Checkpoint*. The current timestamp shall be used as the reference for examining the time of the corresponding *Deadline End Checkpoint*.⌋()

The function `WdgM_CheckpointReached` shall determine the current timestamp by invoking the OS functions ()

SWS_WdgM_00228 means that the timestamp of the reached *Deadline Start Checkpoint* is overwritten by the current timestamp, regardless of the value (just before the overwriting) of the reached *Deadline Start Checkpoint.* Moreover, SWS_WdgM_00228 means that it is not considered as an error by *Deadline Supervision* if a given *Deadline Start Checkpoint* is reached several times without reaching the corresponding *Deadline End Checkpoint* (each time the timestamp is just updated).

[SWS_WdgM_00229] ⌈When the *Deadline End Checkpoint* is reached and this *Checkpoint* is referenced in the active *Mode*, and timestamp of the corresponding *Deadline Start Checkpoint* is <>0, then the function `WdgM_CheckpointReached` shall measure the time difference between current timestamp and the corresponding *Deadline Start Checkpoint* timestamp. Then, the function shall clear (i.e. set to 0) the timestamp of the corresponding *Deadline Start Checkpoint*.⌋()

[SWS_WdgM_00354] ⌈When the *Deadline End Checkpoint* is reached and this *Checkpoint* is referenced in the active *Mode*, and timestamp of the corresponding *Deadline Start Checkpoint* is =0, then the function `WdgM_CheckpointReached` shall exit with success (without measuring the time difference).⌋()

SWS_WdgM_00354 means that it is not considered as an error by *Deadline Supervision* if a given *Deadline End Checkpoint* is reached several times in a sequence.

[SWS_WdgM_00294] ⌈If the measured time difference (see SWS_WdgM_00229) is not within the minimum and the maximum limits (that is, the time difference is either less than `WdgMDeadlineMin` or greater than `WdgMDeadlineMax`), then the function `WdgM_CheckpointReached` shall define the result of *Deadline Supervision* for this *Supervised Entity* as `incorrect`. Otherwise, it shall be defined as `correct`. ⌋()

Note: If the maximum limit (`WdgMDeadlineMax`) is configured with value `'INF'`, it is not necessary to check whether time difference is greater than the limit.

[SWS_WdgM_00299] ⌈For any reported *Checkpoint* that is neither a *Deadline Start Checkpoint* nor a *Deadline End Checkpoint*, the function `WdgM_CheckpointReached` [SWS_WdgM_00263] shall ignore this *Checkpoint* and not update the result of the *Deadline Supervision* for the *Supervised Entity*.⌋()

[SWS_WdgM_00403] ⌈ If Deadline Timeout detection is enabled [i.e. `WdgMEnableTimeoutDetection` (ECUC_WdgM_00363) is set to 'true'] then, for all *Deadline Supervision*s configured in the active mode, if timestamp of the corresponding *Deadline Start Checkpoint* is <>0 (i.e. if the *Start Checkpoint* is reported but corresponding *End Checkpoint* is not yet reported), then the function `WdgM_MainFunction` shall measure the time difference between current timestamp and the corresponding *Deadline Start Checkpoint* timestamp. If the measured time difference exceeds (is greater than) maximum limit (`WdgMDeadlineMax`), then the function `WdgM_MainFunction` shall define the result of *Deadline Supervision* for the *Supervised Entity* as `incorrect`. ⌋(RS_HM_09235)

Note: With this, it is possible to detect error in case *Deadline End Checkpoint* is never reached (timeout detection part of *Deadline Supervision*).

7.2.3 Logical Supervision

Logical Supervision checks if the code of *Supervised Entities* is executed in the correct sequence.

7.2.3.1 Logical Supervision Configuration

For every *Logical Supervision*, there is a *Graph* of *Checkpoints* connected by *Transitions*. The *Graph* abstracts the behavior of the *Supervised Entity* for the Watchdog Manager module.

As an example for a *Supervised Entity*, let us consider the following code fragment, which contains the *Checkpoints* CP0-0 to CP0-6.

```
CP0-0   i = 0;
CP0-1   while(i < n) {
CP0-2       if (a[i] < b[i])
CP0-3           a[i] = b[i];
            else
CP0-4
                a[i] = 0;
CP0-5       i++;
CP0-6   }
```

This *Supervised Entity* can be represented by the *Graph* shown by Figure 8.

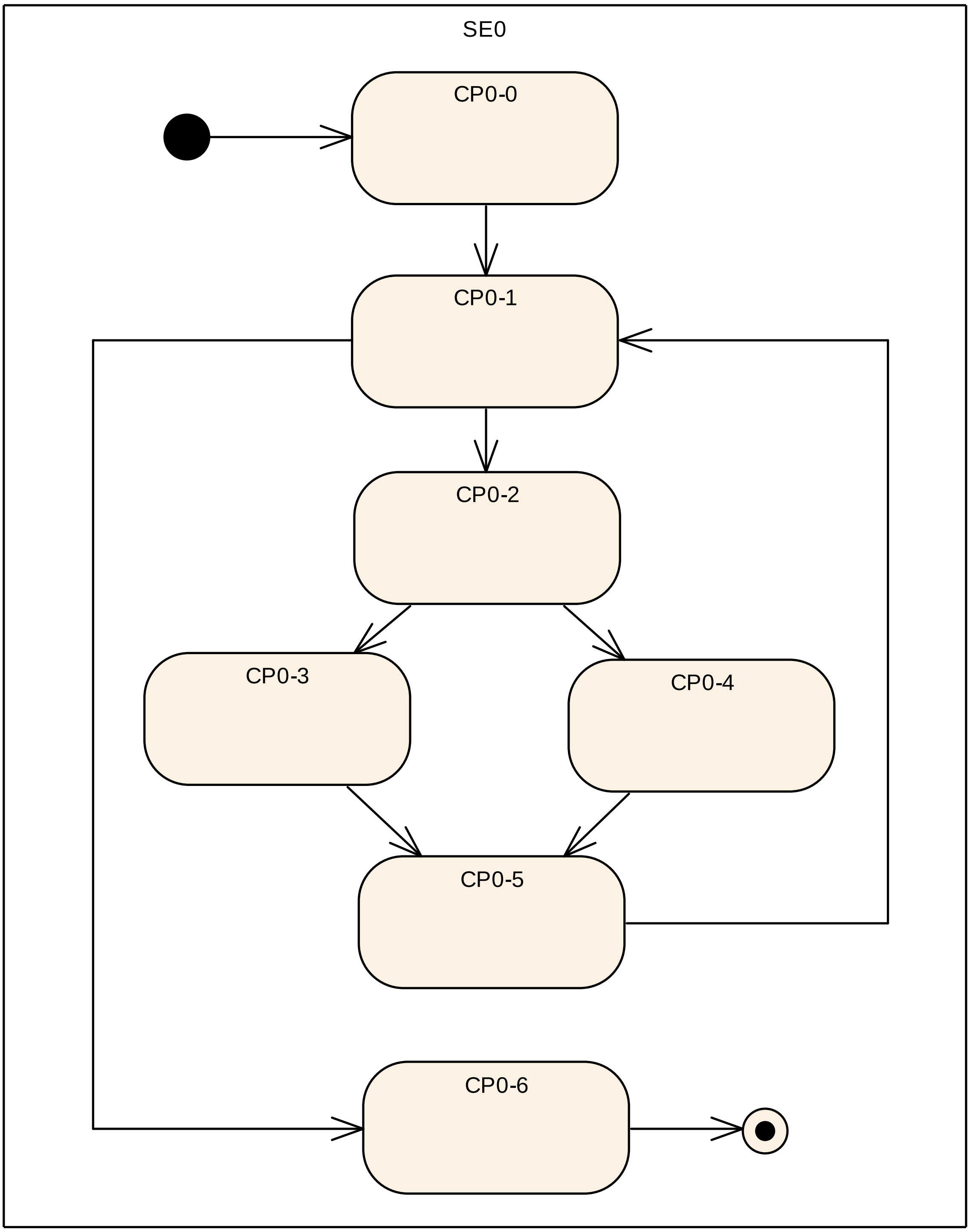

Figure 8: Example Control Flow Graph

A more abstract view of the *Supervised Entity* is given by the *Graph* shown in Figure 9, where the *Checkpoint* CP0-1 represents the complete while loop.

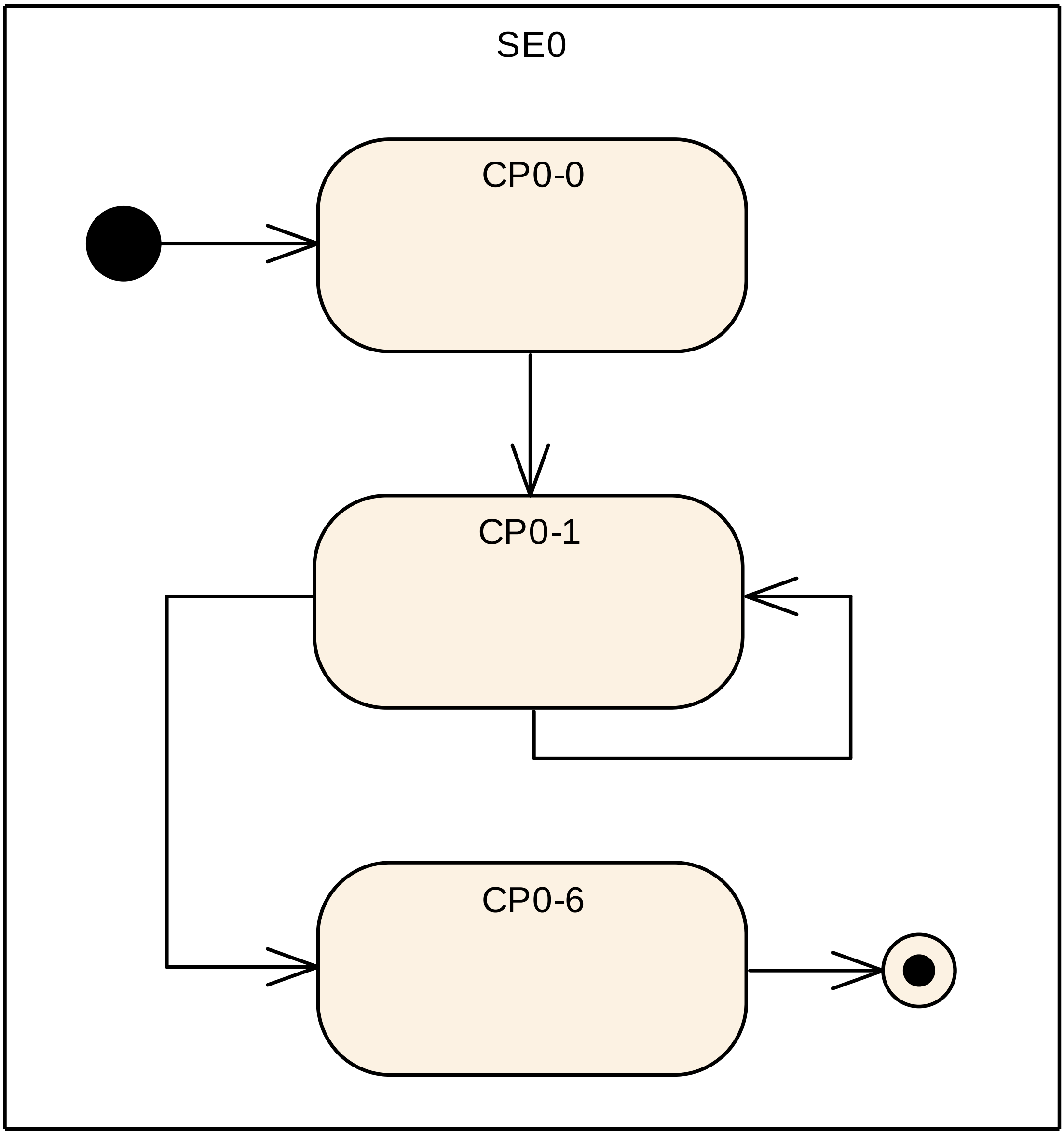

Figure 9: Abstracted Example Control Flow Graph

There are two types of *Graphs* for *Logical Supervision*. Firstly, there is an *Internal Graph*, in which all the *Checkpoints* belong to the same *Supervised Entity* and the *Checkpoints* are connected by *Internal Transitions*.
Second, there is an *External Graph*, in which at least two *Checkpoints* belong to different *Supervised Entities*. The *Checkpoints* are connected with *External Transitions*.

There are two types of *Graphs* for *Logical Supervision*. The main difference of the *Internal Graphs* and *External Graphs* is that an *Internal Graph* is a property of a *Supervised Entity* and is *Mode* independent (i.e. its structure does not change by switching *Watchdog Manager Mode*s, even though its supervision behavior can be disabled if the *Supervised Entity* is disabled in a *Mode*), whereas an *External Graph* is *Mode* dependent.

The parameters of the *Logical Supervision* for *Internal Graphs* are *Internal Transitions* (see `WdgMInternalTransition`), which are contained in a *Supervised Entity* (`WdgMSupervisedEntity`). Each *Internal Transition* connects two *Checkpoints*. This means that all the modes share the same *Internal Transitions*. It is only possible to deactivate a *Supervised Entity* in a *Mode*, which makes its L*ogical Supervision* of *Internal Transitions* inactive.

The parameters of the *External Graphs* (see `WdgMExternalLogicalSupervision`) are contained in a *Mode* (`WdgMMode`). Each *External Transition* connects two *Checkpoints*.

The *Checkpoints* exist irrespective if they are connected by any *Transitions*.

[SWS_WdgM_00366] ⌈The Watchdog Manager module shall derive the required number of independent data resources to perform the *Logical Supervision* within the Watchdog Manager module from the number of *Supervised Entities*, number of `WdgMMode`s and their `WdgMExternalLogicalSupervision`s and `WdgMInternalTransition`s.⌋()

7.2.3.2 Logical Supervision Algorithm

Immediately after initialization of the Watchdog Manager there has not yet been a *Checkpoint* reported, i.e. *Logical Supervision* for the *Supervised Entity* is inactive. This information is held in the *Activity Flag* (one flag per *Graph*).

Each *Internal Graph* represents one *Logical Supervision*. Assuming N *Internal Graphs*, this means that a *Supervised Entity* has N results from *Logical Supervision* for the *Supervised Entity* (Note: currently N is limited up to one per *Supervised Entity)*.

Each *External Graph* represents one *Logical Supervision*, but it spans across possibly several *Supervised Entities*. Assuming M *External Graphs* that cross a *Supervised Entity*, this results with M results from the *Logical Supervision* for the *Supervised Entity*.

[SWS_WdgM_00271] ⌈The Watchdog Manager module shall maintain an *Activity Flag* for each *Graph*.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

[SWS_WdgM_00296] ⌈The function `WdgM_Init` shall set the *Activity Flag* for each *Graph* to `false`.⌋(SRS_BSW_00101)

Each *Graph* may have one or more *Initial Checkpoints*. *Initial Checkpoints* are *Checkpoints* with which a *Graph* can start.

To notify reaching a *Checkpoint*, a *Supervised Entity* invokes the function `WdgM_CheckpointReached`, which results with execution of *Logical Supervision* algorithm.

To verify if transitions are valid, the algorithm needs to store the most recently reached *Checkpoint*. For every *External* and *Internal Graph*, the Watchdog Manger stores the most recently reached *Checkpoint*.

Because a *Checkpoint* can belong to multiple *Graph*s, the function `WdgM_CheckpointReached` has to be able to identify to which *Graph*(s) a *Checkpoint* belongs.

[SWS_WdgM_00295] {DRAFT} ⌈The Watchdog Manager module shall identify to which *Graph*(s) each *Checkpoint* belongs.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

[SWS_WdgM_00246] {DRAFT} ⌈The function `WdgM_CheckpointReached` shall store the *Checkpoint* that has been most recently reported by a *Supervised Entity*, for each *Graph* (see `WdgM_CheckpointReached` [SWS_WdgM_00263]).

If the *Activity Flag* for a *Graph* is true, the function `WdgM_CheckpointReached` checks for each new *Checkpoint* if the *Transition* between the stored *Checkpoint* and the newly reported *Checkpoint* is allowed.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

[SWS_WdgM_00274] ⌈The function `WdgM_CheckpointReached` [SWS_WdgM_00263] shall verify if the reported *Checkpoint* belonging to an *Internal Graph* is a correct one by the following checks:

1. If the *Activity Flag* for the *Graph* of the reported *Checkpoint* is `false`, then:
   a. If the *Checkpoint* is an *Initial Checkpoint* (`WdgMInternalCheckpointInitialRef`) the result of *Logical Supervision* for the *Supervised Entity* is `correct`, otherwise `incorrect`.
2. Else if *Activity Flag* is `true` and all previously called *Checkpoints* of this *Graph* were called in the right sequence, then:
   a. If the reported *Checkpoint* is a successor of the stored *Checkpoint* within the *Graph* of the reported *Checkpoint* (this means there is an `WdgMInternalTransition` with `WdgMInternalTransitionSourceRef` and `WdgMInternalTransitionDestRef`), then the result of this *Logical Supervision* of the *Supervised Entity* is `correct`, otherwise `incorrect`.
3. Else (i.e. *Activity Flag* is `true`, but at least one *Checkpoint* in this *Graph* was previously called in a wrong sequence):

a. The result of this *Logical Supervision* of the *Supervised Entity* keeps `incorrect`.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

A similar check takes place for *Checkpoints* belonging to *External Graphs*.

[SWS_WdgM_00252] ⌈The function `WdgM_CheckpointReached` [SWS_WdgM_00263] shall verify if the reported *Checkpoint* belonging to an *External Graph* is a correct one by the following checks:

1. If the *Activity Flag* for the *Graph* of the reported *Checkpoint* is `false`, then:
   a. If the *Checkpoint* is an *Initial Checkpoint* (`WdgMExternalCheckpointInitialRef`), then the result of this *Logical Supervision* within the *Supervised Entity* of the reported *Checkpoint* is `correct`, otherwise `incorrect`.
2. Else if *Activity Flag* is `true` and all previously called *Checkpoints* of this *Graph* were called in the right sequence, then:
   a. If the reported *Checkpoint* is a successor of the stored *Checkpoint* within the *Graph* of the reported *Checkpoint* (this means there is an `WdgMExternalTransition` with `WdgMExternalTransitionSourceRef` and `WdgMExternalTransitionDestRef`), then the result of this *Logical Supervision* for *Supervised Entity* of the reported *Checkpoint* is `correct`, otherwise `incorrect`.
3. Else (i.e. *Activity Flag* is `true`, but at least one *Checkpoint* in this *Graph* was previously called in a wrong sequence):
   a. The result of this *Logical Supervision* of the *Supervised Entity* keeps `incorrect`.

The above requirement means that in case of an incorrect *External Transition*, the *Supervised Entity* that is considered as erroneous is the one that reported the incorrect *Checkpoint*.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

If a Checkpoint is one of the initial *Checkpoints* of a *Graph*, then the *Graph* is set as active.

[SWS_WdgM_00273] ⌈If the function `WdgM_CheckpointReached` determines that the result of the *Logical Supervision* for the given *Checkpoint* is `correct`, and the *Checkpoint* is defined as an initial one, then the function `WdgM_CheckpointReached` shall set the *Activity Flag* of the corresponding *Graph* to `true`.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

The reverse applies for the *Final Checkpoint*.

[SWS_WdgM_00331] ⌈If the function `WdgM_CheckpointReached` determines that the result of the *Logical Supervision* for the given *Checkpoint* is `correct`, and the *Checkpoint* is defined as a final one, then the function `WdgM_CheckpointReached` shall set the *Activity Flag* of the corresponding *Graph* to false.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

As a result, after the report from a *Final Checkpoint*, the correct reports within the same *Graph* are only from *Initial Checkpoints* (Note: for an evaluation of the *Graph*, any reports from the Checkpoints not belonging to the *Graph* are ignored, see **SWS_WdgM_00297**).

A *Checkpoint* can belong to multiple *Graphs* (can be a combination of *Internal* and *External Graphs*). This means that both the check defined in **SWS_WdgM_00274** and the one in **SWS_WdgM_00252** can be executed simultaneously, and also means that, in any execution of WdgM_CheckpointReached and if the reported checkpoint belongs to any *Internal* or *External Graphs*, the function can set the result of *Logical Supervision* for each corresponding *Supervised Entity* to correct (for all belonging *Graphs*) or incorrect (for all or a part of belonging *Graphs*).

If the reported *Checkpoint* does not belong to any *Graph*, then the result of *Logical Supervision* is not be updated. This is because the *Checkpoint* may be used by other *Supervision Functions* (*Alive* or *Deadline*).

[SWS_WdgM_00297] ⌈For any reported *Checkpoint* that does not belong to any *Graph*, the function WdgM_CheckpointReached [SWS_WdgM_00263] shall ignore it and not update the result of the *Logical Supervision* for the *Supervised Entity*.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

7.3 Error Handling / Failure Recovery

The Watchdog Manager module initiates a number of mechanisms to recover from supervision failures. These range from local error recovery within the *Supervised Entity* to a global reset of the ECU.

7.3.1 RTE Mode Mechanism Notifications

The Watchdog Manager module informs SW-Cs and CDDs about supervision failures via the RTE Mode mechanism. The SW-C and CDDs can then take its actions to recover from that failure. (see [SWS_WdgM_00197], [SWS_WdgM_00198]).

7.3.2 Report to DEM in WDGM_GLOBAL_STATUS_STOPPED

The Watchdog Manager module registers an entry with the Diagnostic Event Manager (DEM) when Watchdog Manages reaches the state

WDGM_GLOBAL_STATUS_STOPPED. An SW-C or a CDD can take recovery actions based on that error entry.

[SWS_WdgM_00129] ⌈Within the first call of `WdgM_MainFunction` after `WdgM_Init` and when the reset-cause was that in the previous operation cycle the *Global Supervision Status* had reached WDGM_GLOBAL_STATUS_STOPPED and if the parameter `WDGM_E_SUPERVISION` is configured, the Watchdog Manager module shall report an error status FAILED for `WDGM_E_SUPERVISION` to the DEM.⌋ (SRS_BSW_00339, SRS_BSW_00458, SRS_BSW_00469, SRS_BSW_00470, SRS_ModeMgm_09159)

7.3.3 Not Setting the Watchdog Trigger Condition

In the state WDGM_GLOBAL_STATUS_STOPPED, the Watchdog Manager module stops setting the trigger condition to Watchdog Interface. As a result, after the timeout of the hardware watchdog, it will cause a reset of the ECU.

See chapter 7.4.2 for the corresponding requirements.

7.3.4 MCU Reset

For applications which need a microcontroller reset as soon as an unrecoverable supervision failure is detected, or to have the independent shutdown path from the Hardware Watchdog, the Watchdog Manager module can perform an immediate reset of the MCU.

[SWS_WdgM_00133] ⌈If the configuration parameter `WdgMImmediateReset` [ECUC_WdgM_00339] is set to TRUE and the *Global Supervision Status* has reached the state WDGM_GLOBAL_STATUS_STOPPED, the Watchdog Manager module shall call the MCU service `Mcu_PerformReset` on the MCU Driver module.⌋(SRS_ModeMgm_09169)

[SWS_WdgM_CONSTR_06500] Interface provision in MCU driver ⌈ The parameter `WdgMImmediateReset` [ECUC_WdgM_00339] may only be set to TRUE if the `McuPerformResetApi` (defined in SWS_Mcu_Driver) is set to TRUE.⌋ (SRS_ModeMgm_09169)

[SWS_WdgM_00134] ⌈In case of an immediate MCU reset, the Watchdog Manager module shall not provide a notification to the application via the RTE mode mechanism.⌋(SRS_ModeMgm_09169)

7.4 Watchdog Handling

The handling of watchdogs is an important feature of the Watchdog Manager module. It prevents the ECU from resets by expired hardware watchdog instances while program execution is running properly.

Usually hardware watchdogs have their own timing constraints and the trigger for each watchdog instance must be performed cyclically within a maximum time span or within a defined time window according to the timing constraints of the corresponding watchdog instance. If the trigger does not occur, the corresponding hardware watchdog instance will cause a reset.

The actual timing of watchdog triggering is encapsulated in the Watchdog Driver. The Watchdog Manager only sets via the Watchdog Interface a triggering condition that instructs the Watchdog Driver to continue triggering.

7.4.1 Support for Multiple Watchdog Instances

Some hardware platforms can be designed to have multiple watchdog instances (i.e. an internal and an external watchdog in parallel).

[SWS_WdgM_00002] ⌈The Watchdog Manager module shall support the parallel usage of multiple watchdogs.⌋(SRS_ModeMgm_09028)

7.4.2 Setting the Trigger Conditions

The Watchdog Manager module uses the service `WdgIf_SetTriggerCondition` of the Watchdog Interface modules to set (update) the trigger condition of the watchdogs. This service requires the watchdog device index and the timeout/counter as a parameter (see configuration parameter `WdgMTrigger` [ECUC_WdgM_00331]).

[SWS_WdgM_00223] ⌈The Watchdog Manager module shall update the trigger condition every time the *Global Supervision Status* has been recomputed. The following rules shall be used to derive the decision, how to set the triggering condition:

1. For the states WDGM_GLOBAL_STATUS_OK, WDGM_GLOBAL_STATUS_FAILED and WDGM_GLOBAL_STATUS_EXPIRED, the function `WdgM_MainFunction` shall correctly set the trigger conditions.
2. For the state WDGM_GLOBAL_STATUS_STOPPED, the function `WdgM_MainFunction` shall set the trigger condition to 0, which results in a reset through HW watchdog(s).
3. For the state WDGM_GLOBAL_STATUS_DEACTIVATED, the function `WdgM_MainFunction` shall not perform setting of the trigger condition (because this state means that the Watchdog Manager module is not properly initialized).

⌋(SRS_ModeMgm_09161, SRS_ModeMgm_09226)

[SWS_WdgM_00119] ⌈If the *Global Supervision Status* has recomputed as WDGM_GLOBAL_STATUS_OK, then the Watchdog Manager module shall call WdgIf_SetTriggerCondition for all watchdogs not configured as WDGIF_OFF_MODE [ECUC_WdgM_00332] with <parameter for id> set to WdgMWatchdogDeviceRef [ECUC_WdgM_00348] and <parameter for trigger condition> set to WdgMTriggerCondition [ECUC_WdgM_00333].⌋()

[SWS_WdgM_00120] ⌈If the *Global Supervision Status* has recomputed as WDGM_GLOBAL_STATUS_FAILED, then the Watchdog Manager module shall call WdgIf_SetTriggerCondition for all watchdogs not configured as WDGIF_OFF_MODE [ECUC_WdgM_00332] with <parameter for id> set to WdgMWatchdogDeviceRef [ECUC_WdgM_00348] and <parameter for trigger condition> set to WdgMTriggerCondition [ECUC_WdgM_00333].⌋()

[SWS_WdgM_00121] ⌈If the *Global Supervision Status* has recomputed as WDGM_GLOBAL_STATUS_EXPIRED, then the Watchdog Manager module shall call WdgIf_SetTriggerCondition for all watchdogs not configured as WDGIF_OFF_MODE [ECUC_WdgM_00332] with <parameter for id> set to WdgMWatchdogDeviceRef [ECUC_WdgM_00348] and <parameter for trigger condition> set to WdgMTriggerCondition [ECUC_WdgM_00333].⌋()

[SWS_WdgM_00122] ⌈If the *Global Supervision Status* has recomputed as WDGM_GLOBAL_STATUS_STOPPED, then the Watchdog Manager module shall call WdgIf_SetTriggerCondition for all watchdogs not configured as WDGIF_OFF_MODE [ECUC_WdgM_00332] with <parameter for id> set to WdgMWatchdogDeviceRef [ECUC_WdgM_00348] and <parameter for trigger condition> set to zero.⌋()

Setting the trigger condition to zero will immediately prevent the Watchdog Driver module from triggering the hardware watchdog.

7.5 Switching Modes

7.5.1 Effect on Supervision Status

The function WdgM_SetMode (see [SWS_WdgM_00154]) is used to switch between different modes. The modes are statically configured and contained in the Watchdog Manager module configuration set.

A *Mode* switch changes the supervision parameters of the *Supervised Entities*.

[SWS_WdgM_00182] ⌈If the current global status is WDGM_GLOBAL_STATUS_OK or WDGM_GLOBAL_STATUS_FAILED then for each *Supervised Entity* that is activated in the new mode (passed to function `WdgM_SetMode` as parameter), the function `WdgM_SetMode` shall retain the current state of the *Supervised Entity*.

Switching to the mode where a *Supervised Entity* is deactivated clears also errors that had resulted with the WDGM_GLOBAL_STATUS_FAILED status.⌋()

[SWS_WdgM_00315] ⌈If the current global status is WDGM_GLOBAL_STATUS_OK or WDGM_GLOBAL_STATUS_FAILED then for each *Supervised Entity* that is deactivated in the new mode (passed to function `WdgM_SetMode` as parameter), the function `WdgM_SetMode` shall change the state of the *Supervised Entity* to WDGM_LOCAL_STATUS_DEACTIVATED; It shall set its Results of *Active*, *Deadline* and *Logical Supervision* to `correct`; It shall also clear its failed reference cycle counter to 0.⌋()

Executing a mode switch is possible when the Watchdog Manager module is in the state WDGM_GLOBAL_STATUS_OK or WDGM_GLOBAL_STATUS_FAILED. In other modes the function `WdgM_SetMode` has no effect (see [SWS_WdgM_00145]).

[SWS_WdgM_00316] ⌈If the current global status is not WDGM_GLOBAL_STATUS_OK nor WDGM_GLOBAL_STATUS_FAILED then the function `WdgM_SetMode` shall return without doing any actions.⌋()

7.5.2 Effect on Watchdogs

A mode switch also changes the parameters for watchdog triggering.

[SWS_WdgM_00186] ⌈If function `WdgM_SetMode` (see [SWS_WdgM_00154]) is called, the Watchdog Manager module shall apply the configured watchdog mode parameters (see `WdgMWatchdogMode` [ECUC_WdgM_00332]) to each watchdog by calling the `WdgIf_SetMode` service.⌋()

Note: If a call to `WdgIf_SetMode` service fails, the Watchdog Manager module assumes a global supervision failure and set the *Global Supervision Status* to WDGM_GLOBAL_STATUS_STOPPED (see [SWS_WdgM_00139]). This will cause a reset, either when the first watchdog expires or immediately, if an immediate reset of the Watchdog Manager module is configured.

There is also the possibility to forbid switching off the watchdogs (see [SWS_WdgM_00031]).

7.5.3 Watchdog Handling during Sleep

When the ECU State Manager enters SLEEP state it activates the sleep mode and calls the service `WdgM_DeInit`.

The `WdgM_DeInit` (see [SWS_WdgM_00261]) updates the trigger conditions via a Watchdog Manager *Mode* switch to a sleep mode defined by the integrator and deinitializes the Watchdog Manager module. The mode switch is needed to update the watchdogs trigger conditions of all running watchdogs to a timeout that allows the rest of the shutdown to be executed without a watchdog reset. This is needed as a consequence of the concept “Windowed Watchdogs”.

While the ECU is in SLEEP state, the normal execution of code and therefore also of the Watchdog Manager module is suspended. If the hardware watchdogs cannot or shall not be deactivated during SLEEP, this would inevitably lead to a watchdog reset.

Thus, the watchdogs have to be triggered at some time during SLEEP. BSW components which are still in-service (like the BswM or the EcuM) have to care about the triggering of the hardware watchdogs while the Watchdog Manager module is deactivated. The Integrator has to configure the needed modes accordingly.

7.6 Watchdog Manager Configuration

7.6.1 Mode-independent Supervision Settings

7.6.1.1 Supervised Entity

To support portability of SW-Cs across platforms, the Watchdog Manager module needs to be adapted to the amount of *Supervised Entities* located on the respective ECU.

[SWS_WdgM_CONSTR_06502] {DRAFT} ⌈A unique *Supervised Entity* identifier for each *Supervised Entity* is provided in configuration parameter WdgMSupervisedEntityID (see [ECUC_WdgM_00304]). The Identifier shall be unique in the scope of a Watchdog Manager configuration.⌋()

The *Supervised Entities* and *Checkpoints* exist irrespective of *Modes*. On the other side, the *Supervision Functions* exist partially irrespective of *Modes*, and partially dependent on *Modes*.

[SWS_WdgM_00282] ⌈In order to have a *Supervised Entity* with supervision activated in a given mode (in short: Activated *Supervised Entity*), the following shall be fulfilled:

1. The *Supervised Entity* shall be referenced from the *Mode* (see WdgMMode → WdgMLocalStatusParams → WdgMLocalStatusSupervisedEntityRef → WdgMSupervisedEntity AND
2. At least one of mode-dependent settings of *Supervision Functions* shall be set for the given *Mode* (*Alive*, *Deadline*, *Logical* for *External Graphs*)⌋()

[SWS_WdgM_00283] ⌈In order to have a *Supervised Entity* with supervision deactivated in a given mode (in short: Deactivated *Supervised Entity*), the following shall be fulfilled:

1. The *Supervised Entity* shall not be referenced from the *Mode* (see WdgMMode → WdgMLocalStatusParams → WdgMLocalStatusSupervisedEntityRef → WdgMSupervisedEntity AND
2. No mode-dependent settings of *Supervision Functions* shall be set for the given *Mode* (*Alive*, *Deadline*, *Logical* for *External Graphs*)

As the *Logical supervision* for *Internal Graphs* is a property of a *Supervised Entity*, the configurations of *Logical Supervision* for *Internal Graphs* do not impact the deactivation/activation status of *Supervised Entity*.⌋()

7.6.1.2 Logical Supervision of Internal Graphs

Each *Supervised Entity* can have a configured control flow that is supervised by Watchdog Manager. This control flow is abstracted by its *Checkpoints* and *Transitions* (see [ECUC_WdgM_00303]). At least one of the *Checkpoints* per *Graph* is marked as the initial one (see [ECUC_WdgM_00343]).

[SWS_WdgM_CONSTR_06506] ⌈*Internal Transitions* (see WdgMInternalTransition) in a *Supervised Entity* shall not connect *Checkpoints* that do not both belong to the same *Supervised Entity*.⌋()

To switch on and off the *Logical Supervision* of an *Internal Graph* depending on the mode, it is needed to reference (or respectively do not reference) the *Supervised Entity* from each mode (see WdgMLocalStatusParams).

It is possible to have zero or one *Internal Graphs* per *Supervised Entity*. Not all *Checkpoints* of a *Supervised Entity* need to be a part of its *Internal Graph*.

The *Internal Transitions* and *Internal Graphs* are a property of *Supervised Entity*. These *Internal Transitions* depend only on the control flow within the *Supervised Entity*. Thus, the developer of an SW-C or BSW module that contains the *Supervised Entity* can deliver this configuration of *Checkpoints* and *Internal Transitions* independently of other *Supervised Entities*. Figure 10 shows a configuration of two independently *Supervised Entities*, with independently configured *Internal Graphs*.

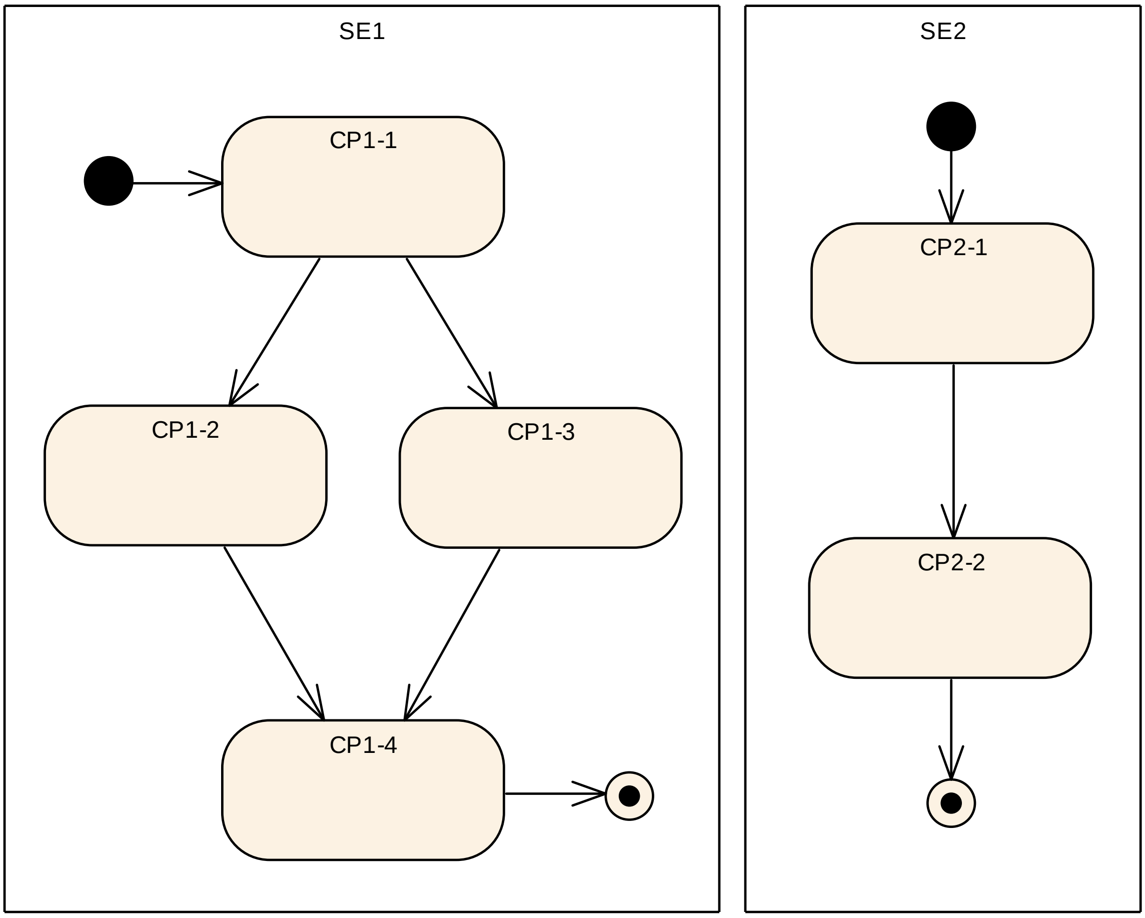

Figure 10: Two Supervised Entities with their Checkpoints and Internal Transitions

7.6.2 Mode-Dependent Parameters

7.6.2.1 Mode

Changing the mode of the Watchdog Manager module (Watchdog Manager *Mode*) also leads to changed conditions for handling the watchdogs, such as different watchdog modes. Therefore the Watchdog Manager module provides for each configured mode and for each watchdog a number of statically configured watchdog parameters (see `WdgMTrigger` [ECUC_WdgM_00331]).

[SWS_WdgM_00181] ⌈For each watchdog instance, the watchdog mode shall be statically configured and represented by the parameter `WdgMWatchdogMode.`⌋()

The corresponding watchdog can be disabled by configuring the watchdog mode to WDG_OFF_MODE.

The Watchdog Manager module has a set of statically configured supervision parameters for each configured mode (WdgMMode [ECUC_WdgM_00335]) and for each *Supervised Entity* that is expected to be supervised in the given mode.

7.6.2.2 Logical Supervision of External Graphs

There are also *Transitions* that cross the boundaries of *Supervised Entities*. These *External Transitions* appear when the Watchdog Manager module should also supervise the execution sequence of multiple *Supervised Entities*. The *External Transitions* form *External Graphs*.

Thus, *External Transitions* have to be configured independently from the *Internal Transitions* and only in the context of *Logical Supervision*. (see WdgMExternalLogicalSupervision [ECUC_WdgM_00319])

When we integrate the two *Supervised Entities* from Figure 10, we can for example decide that *Supervised Entity* SE1 must always be executed to *Checkpoint* CP1-4 and then *Supervised Entity* SE2 has to start execution at *Checkpoint* CP2-1. Then it is necessary to configure a *Transition* from CP1-4 to CP2-1. This *Transition* does neither belong to SE1 nor to SE2. Figure 6 shows the *External Transition*.

There is a significant difference in configuring *Internal* and *External Transitions*. An *Internal Transition* belongs to one *Supervised Entity* and it does not depend on the Watchdog Manager *Modes*. One can configure to activate/deactivate an SE in a given mode by referencing it from the mode. However, it is not possible to have different *Transitions* or *Checkpoints* within the same SE depending on the mode. In contrary, *External Transitions* are contained in a particular Watchdog Manager *Mode*. There can be several *External Transition Graphs* per mode. In case two different *Modes* have same global *Graphs* of global *Transitions*, then they need to be duplicated.

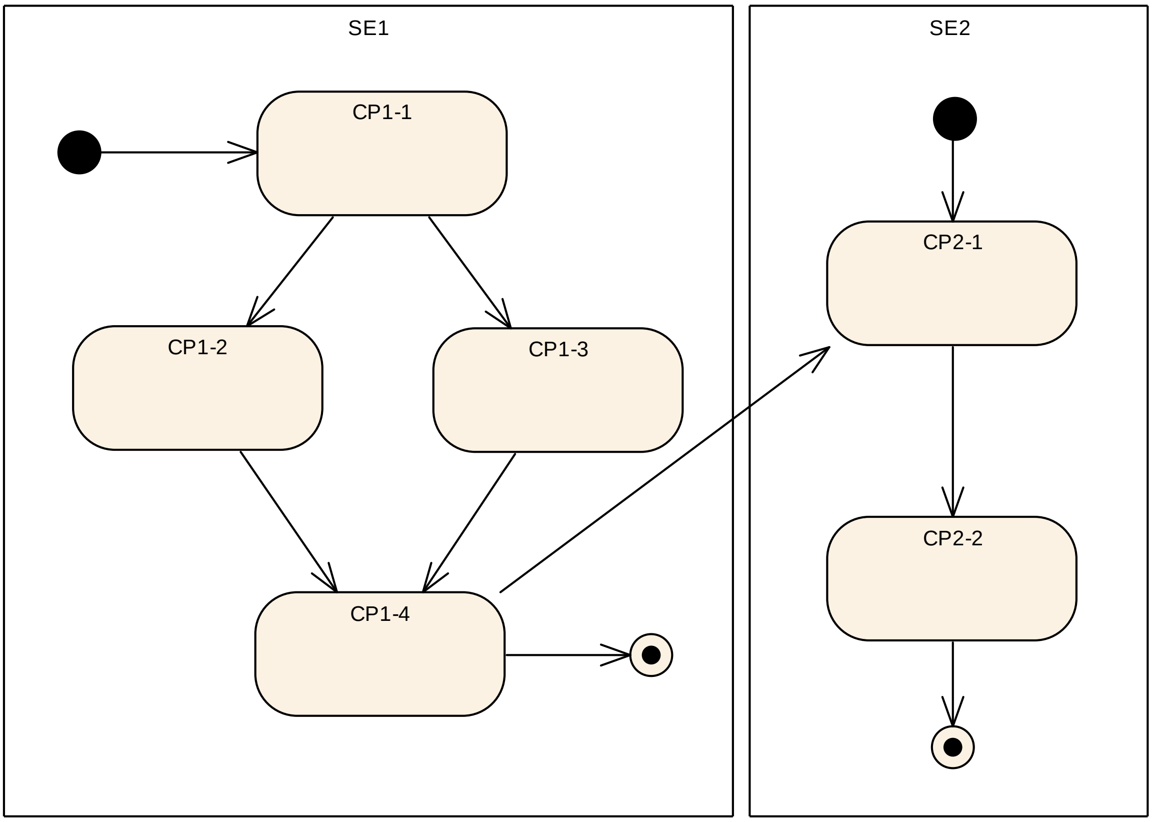

Figure 11: Two Supervised Entities with an External Transition

The start points (see [ECUC_WdgM_00324]), endpoints (see [ECUC_WdgM_00323]) and the *External Transitions* are configured for each Watchdog Manager *Mode* (see [ECUC_WdgM_00319]).

The Watchdog Manager module supports a number of different modes (see `WdgMConfigSet` [ECUC_WdgM_00337]) of operation. Each mode (see `WdgMMode` [ECUC_WdgM_00335]) is defined by:

- the set of *Activated Supervised Entities* (see [SWS_WdgM_00282]) and their parameters (see `WdgMLocalStatusParams` [ECUC_WdgM_00325]),
- the *Supervision Functions* (see `WdgMAliveSupervision` [ECUC_WdgM_00308], `WdgMDeadlineSupervision` [ECUC_WdgM_00314], `WdgMProgramFlow- Supervision` [ECUC_WdgM_00319]),
- the set of watchdogs to have their trigger condition updated (see `WdgMTrigger` [ECUC_WdgM_00331])

Different modes are needed for different phases in the ECU life cycle. E.g. one mode is active during startup and shutdown, another during normal operation and yet another during sleep. Even during normal operation, multiple modes could be needed: when multiple applications run on the same ECU, one application could be shutdown already and require no supervision, while another application still runs and needs to be supervised.

[SWS_WdgM_00178] ⌈Each mode of the Watchdog Manager module has an identifier (see `WdgMModeId` [ECUC_WdgM_00307]) which shall be unique.⌋()

[SWS_WdgM_00179] ⌈The Watchdog Manager module has one initial mode `WdgMInitialMode` [ECUC_WdgM_00336] which shall be activated when it is initialized.⌋()

7.6.2.3 Alive Supervision

The timing constraints of each *Checkpoint* are represented by configurable parameters of the Watchdog Manager module (see `WdgMAliveSupervision` [ECUC_WdgM_00308]). Although the timing constraints are defined for a *Checkpoint*, the Watchdog Manager determines the result of the *Alive Supervision* for the whole *Supervised Entity*.

The acceptable amount of *Failed Supervision Reference Cycles* is based on application context of each *Supervised Entity*. Therefore the individual thresholds to check if *Alive Supervision* of the corresponding *Supervised Entity* has failed finally, needs to be a configurable parameter (see `WdgMFailedAliveSupervisionRefCycleTol` [ECUC_WdgM_00327]).

When the *Alive Supervision* has reached expired conditions by any *Local Supervision Status*, this will make recovery obsolete. As a consequence the watchdog triggering will be stopped, but to ensure a certain time-period for any further reactions on this condition, the blocking of watchdog triggering could be postponed for an amount of consecutive *Supervision Cycles* (see `WdgMExpiredSupervisionCycleTol` [ECUC_WdgM_00329]).

[SWS_WdgM_CONSTR_00320] ⌈No two `WdgMAliveSupervisions` aggregated by the same `WdgMMode` shall refer to the identical `WdgMCheckpoint`.⌋()

7.6.2.4 Deadline Supervision

[SWS_WdgM_CONSTR_06505] ⌈*Deadline Supervision* (`WdgMDeadlineSupervision`) of a *Supervised Entity* shall refer to *Checkpoints* (`WdgMDeadlineStartRef`, `WdgMDeadlineEndRef`) that both belong to that *Supervised Entity*. In other words, any of the referred *Checkpoints* shall not belong to other *Supervised Entities*.⌋()

[SWS_WdgM_CONSTR_06512]⌈Any ordered set of two *Checkpoints* shall not have more than one *Deadline Supervision* (`WdgMDeadlineSupervision`) defined.⌋()

7.7 Support for Clustered Software Architecture using Software Cluster Connector (SwCluC)

This section is applicable to clustered software architecture (`WdgMSwClusterSupport` = ENABLE_SW_CLUSTER_SUPPORT) only, i.e. not applicable to non-clustered software architecture.

7.7.1 Software Architectural Assumptions and Constraints

For an ECU Software which supports clustered software architecture (with or without a multi-partition configuration), it is assumed that the Watchdog Manager will be allocated to each Software Cluster with the fashion below (also illustrated in Figure 12):

- Within the Host Software Cluster, the WdgM shall provide complete sets of APIs (WdgM_MainFunction, WdgM_CheckpointReached etc.). At least one WdgM_MainFunction will be available per EcucPartition. These API sets perform:
  - Alive, Deadline and Logical Supervision within the Host Software Cluster, per EcucPartition (i.e. in the master and in every satellites)
  - Logical Supervision over Software Clusters, based on Cross-Cluster Graph (only in the EcucPartition which contains master side of WdgM)
  - Determination of Local Supervision Status per Supervised Entity
  - Determination of Global Supervision Status (only in the master)
  - Recovery Actions based on Local Supervision Status
  - Recovery Actions based on Global Supervision Status (only in the master)
  - Watchdog Handling (incl. Watchdog Trigger via WdgIf and Wdg modules) (only in the master)
- Within the Host Software Cluster, WdgM shall provide satellites (WdgM_MainFunctions) on all EcucPartitions, that can be connected to WdgM masters within every Application Software Cluster. This ensures that each WdgM (master) in an Application Software Cluster can get access to the WdgM in the Host Software Cluster on the same partition.
- Within each Application Software Cluster, WdgM shall provide subsets of APIs. At least one WdgM_MainFunction will be available per EcucPartition.
  - Alive, Deadline and Logical Supervision within the Host Software Cluster, per EcucPartition (i.e. in the master and in every satellites)
  - Determination of Local Supervision Status per Supervised Entity
  - Recovery Actions based on Local Supervision Status

Note that, if there're multiple Main Functions in the master side within Host Software Cluster, following design decision will be required, but not standardized in this specification (because realization of master-satellite pattern is implementation specific).

- Mapping of Recovery Action etc. to Main Functions
- Availability of Init / DeInit APis etc.

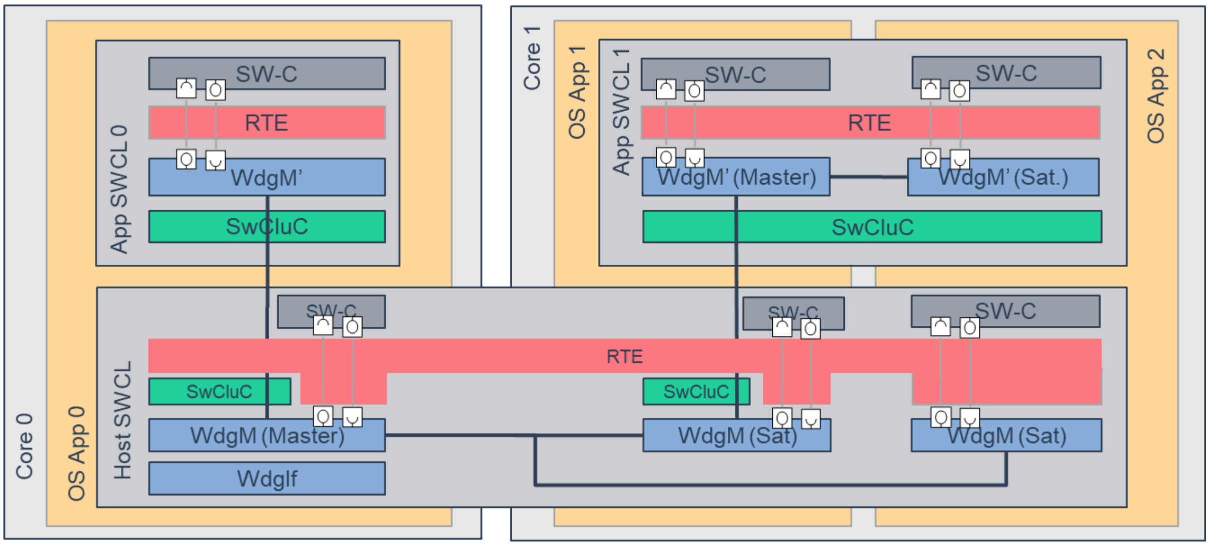

Figure 12: Overview of Watchdog Manager with Software Clustering

7.7.2 Configuration Aspects

[SWS_WdgM_CONSTR_06516] {DRAFT} Software Cluster related configurations cannot be used with disabled Software Cluster Support ⌈In case of non-clustered software architecture (WdgMSwClusterSupport is not set or set to DISABLE_SW_CLUSTER_SUPPORT), the parameters and containers WdgMCrossClusterTransition, WdgMTransitionProxy and WdgMBaseSocket shall not exist.⌋()

[SWS_WdgM_CONSTR_06517] {DRAFT} Valid cross cluster transition ⌈A WdgMCrossClusterTransition is only valid in following configurations:

- from a WdgMCheckpoint to a WdgMTransitionProxy
- from a WdgMTransitionProxy to a WdgMCheckpoint
- from a WdgMTransitionProxy to another WdgMTransitionProxy (in Host Software Cluster only)
- from a WdgMTransitionProxy to the identical WdgMTransitionProxy (in Application Software Cluster only for the case that no WdgMCheckpoint has to be reached in the Application Software Cluster), or
- from a WdgMCheckpoint to a WdgMCheckpoint (in case the cross cluster transition graph is entirely described with WdgMCrossClusterTransition containers).

Hereby the "from" is configured with the WdgMCrossClusterTransitionSourceRef, and the "to" is given by the WdgMCrossClusterTransitionDestRef.⌋()

[SWS_WdgM_CONSTR_06518] {DRAFT} WdgMBaseSocket relates only to a CpSoftwareClusterServiceResource of category SWCLUSTER_RES_WDGM_BASES_SOCKET ⌈The WdgMBaseSocket.WdgMResourceRef shall only reference a CpSoftwareClusterServiceResource of category SWCLUSTER_RES_WDGM_BASES_SOCKET.⌋()

[SWS_WdgM_CONSTR_06519] {DRAFT} WdgMTransitionProxy relates only to a CpSoftwareClusterServiceResource of category SWCLUSTER_RES_WDGM_TRANSITION ⌈The WdgMTransitionProxy.WdgMResourceRef shall only reference a CpSoftwareClusterServiceResource of category SWCLUSTER_RES_WDGM_TRANSITION.⌋()

ECU Configuration will be made per Software Cluster. Therefore,

- A *Supervised Entity ID* can be reused in different Software Clusters (see also [SWS_WdgM_CONSTR_06502])
- WdgMMode and WdgMInitialMode configuration must be consistent over Software Clusters (Host Software Clusters and Application Software Clusters)

Note that, type of Software Cluster can be identified by SwCluCGeneral. SwCluCDefinitionSelection.

7.7.2.1 Configuration for Cross-Cluster External Graphs

Cross-Cluster External Graph is an extension of *External Graph* to model *Graphs* that spans over multiple Software Clusters for clustered software architecture.
To model *Graphs* with inter-Cluster *Transitions*, following configuration elements can be used:

- WdgMCrossClusterTransition (instead of WdgMExternalTransition) which represents a *Transition* to other Software Cluster (contains reference to destination *Checkpoint* in other Software Cluster) or a *Transition* from other Software Cluster (contains reference to source *Checkpoint* in other Software Cluster)
- WdgMTransitionProxy (instead of WdgMCheckpoint) which represents a *Checkpoint* in other Software Cluster

7.8 Error classification

7.8.1 Development Errors

[SWS_WdgM_00004]⌈

Type of error	*Related error code*	*Error value*
API service used in wrong context (without module initialization)	WDGM_E_ UNINIT	0x10
API service Wdg_Init was called with an erroneous configuration set	WDGM_E_ PARAM_CONFIG	0x11
API service called with wrong "mode" parameter	WDGM_E_ PARAM_MODE	0x12
API service called with wrong "supervised entity identifier" parameter	WDGM_E_ PARAM_SEID	0x13
API service called with invalid pointer	WDGM_E_INV_ POINTER	0x14
API service used with an invalid CheckpointId.	WDGM_E_CPID	0x16
API service used in wrong context - WdgM_Init called when module is not deinitialized (global status is not WDGM_GLOBAL_STATUS_DEACTIVATED)	WDGM_E_NO_ DEINIT	0x1A
Initialization failed, e.g. selected configuration set doesn't exist	WDGM_E_INIT_ FAILED	0x1B
API service called with a null pointer parameter	WDGM_E_ PARAM_ POINTER	0x1C

⌋(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00480, SRS_BSW_00481, SRS_BSW_00487)

7.8.2 Runtime Errors

[SWS_WdgM_00402]⌈

Type of error	*Related error code*	*Error value*
Disabling of watchdog not allowed (e.g. in safety-related systems)	WDGM_E_DISABLE_ NOT_ALLOWED	0x15
API service used with a checkpoint of a Supervised Entity that is deactivated in the current Watchdog Manager mode.	WDGM_E_ SEDEACTIVATED	0x19

Watchdog drivers' mode switch has failed	WDGM_E_SET_MODE	0x1D

⌋(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385)

7.8.3 Transient Faults

There are no transient faults.

7.8.4 Production Errors

The Watchdog Manager module detects the following production errors:

[SWS_WdgM_00375]⌈

Error Name:	WDGM_E_SUPERVISION	
Short Description:	Supervision has failed and a watchdog reset will occur	
Long Description:	Supervision has failed (Global Supervision Status has reached WDGM_GLOBAL_STATUS_STOPPED) and a watchdog reset will occur.	
Detection Criteria:	Fail	WDGM_GLOBAL_STATUS_STOPPED has been reached, the reset will occur.
	Pass	After a start up.
Secondary Parameters:	-	
Time Required:	depending on configuration of WdgM	
Monitor Frequency	periodic supervision within WdgM	

⌋(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00458)

Note: The stored DTC will never show up as "confirmed", because it will be reset at each start up (see [SWS_Dem_00391]).

Note: The stored DTC may not show up "test failed (event active)" even if DemStatusBitStorageTestFailed were set to true, because storage of the DTC cannot be always ensured after reaching *Global Supervision Status* = WDGM_GLOBAL_STATUS_STOPPED (see [SWS_Dem_00388] and [SWS_Dem_00525]).

[SWS_WdgM_00408] ⌈Within the first call of WdgM_MainFunction after WdgM_Init, but after [SWS_WdgM_00129] is executed and if the parameter WDGM_E_SUPERVISION is configured, the Watchdog Manager module shall report an error status PASSED for WDGM_E_SUPERVISION to the DEM. ⌋ (SRS_BSW_00339, SRS_BSW_00458, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_ModeMgm_09159)

7.8.5 Extended Production Errors

There are no extended production errors.

8 API Specification

8.1 Imported Types

The following data types are used by Watchdog Manager module.

[SWS_WdgM_00011]⌈

Module	*Header File*	*Imported Type*
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Os	Os.h	StatusType
	Os.h	TickRefType
	Os.h	TickType
	Rte_Os_Type.h	CounterType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType
SwCluC	SwCluC_BManif.h	SwCluC_BManif_HandleIndexType (draft)
	SwCluC_BManif.h	SwCluC_BManif_SwClusterIdType (draft)
	SwCluC_BManif.h	SwCluC_BManif_TableIndexType (draft)
WdgIf	WdgIf.h	WdgIf_ModeType

⌋(SRS_BSW_00357)

8.2 Type Definitions

The following Data Types are used for the functions defined in this specification.

8.2.1 WdgM_ConfigType

[SWS_WdgM_00355]⌈

Name	WdgM_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	--
	Comment	The contents of this structure depends on the configuration variant.

Description	This structure contains all post-build configurable parameters of the Watchdog Manager. A pointer to this structure is passed to the Watchdog Manager initialization function for configuration.
Available via	WdgM.h

⌋()

8.3 Function Definitions

[SWS_WdgM_00411] {DRAFT} ⌈For clustered software architecture (one Host Software Cluster and zero or more Application Software Cluster), Host Software Cluster shall provide all APIs which are permanently available or enabled by configuration.⌋()

[SWS_WdgM_00412] {DRAFT} ⌈For clustered software architecture, Application Software Cluster shall provide following APIs which are permanently available or enabled by configuration.

- `WdgM_GetVersionInfo`
- `WdgM_CheckpointReached`
- `WdgM_GetMode`
- `WdgM_GetLocalStatus`

⌋()

8.3.1 WdgM_Init

[SWS_WdgM_00151]⌈

Service Name	WdgM_Init	

Syntax	`void WdgM_Init (`
	`  const WdgM_ConfigType* ConfigPtr`
	`)`	

Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to post-build configuration data
Parameters (inout)	None	

Parameters (out)	None
Return value	None
Description	Initializes the Watchdog Manager.
Available via	WdgM.h

⌋(SRS_BSW_00310, SRS_BSW_00358, SRS_ModeMgm_09107)

This function initializes the Watchdog Manager. After execution of this function, supervision is activated according to the list of *Supervised Entities* defined in the initial *Mode*.

To perform a module reinitialization (e.g. after error), the caller can invoke WdgM_DeInit() and then WdgM_Init().

[SWS_WdgM_00018] ⌈The function WdgM_Init shall initialize all module variables (global and static) of the Watchdog Manager module.⌋(SRS_BSW_00101, SRS_ModeMgm_09107)

[SWS_WdgM_00135] ⌈The function WdgM_Init shall establish the initial mode of the Watchdog Manager module.⌋(SRS_BSW_00101, SRS_ModeMgm_09107)

Note: If a call to WdgIf_SetMode service fails during WdgM_Init, then the MCU Reset API is called directly (only if configured, see [SWS_WDGM_00133]) and the Watchdog Manager module will be in state initialized afterwards with *Global Supervision Status* = WDGM_GLOBAL_STATUS_STOPPED (see [SWS_WdgM_00139]). This will cause a reset, either when the first watchdog expires (if an immediate reset of the Watchdog Manager module is not configured) or immediately (if an immediate reset is configured).

[SWS_WdgM_00030] ⌈If the WdgMOffModeEnabled [ECUC_WdgM_00340] switch is not enabled, and the initial mode provided by the configuration (ConfigPtr) will disable the watchdog (WDGIF_OFF_MODE) then the function WdgM_Init shall return with E_NOT_OK without any action, and the function WdgM_Init shall report runtime error code WDGM_E_DISABLE_NOT_ALLOWED to the Default Error Tracer. ⌋(SRS_BSW_00323, SRS_BSW_00452, SRS_ModeMgm_09109)

There are optional checks that are executed if and only if WdgMDevErrorDetect is enabled.

[SWS_WdgM_00389] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled: The function WdgM_Init shall report the error to default error tracer with error code WDGM_E_UNINIT, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00390] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is disabled: The function WdgM_Init shall return without any effect if the Watchdog Manager is not in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00010] ⌈If the WdgMDevErrorDetect [ECUC_WdgM_00301] switch is enabled and the configuration variant is VARIANT-POST-BUILD, the function WdgM_Init shall check the contents of the given configuration set for being within the allowed boundaries. If the function WdgM_Init detects an error, then it shall not execute the initialization of the Watchdog Manager module and it shall report the error code WDGM_E_PARAM_CONFIG to the Det_ReportError service of the Default Error Tracer.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00370] ⌈The function WdgM_Init shall clear from the non-initialized RAM the double-inverse value storing the SEID that first reached the EXIRED state. See *8.3.10* for more information.⌋(SRS_BSW_00101)

8.3.2 WdgM_DeInit

[SWS_WdgM_00261]⌈

Service Name	WdgM_DeInit

Syntax	`void WdgM_DeInit (`
	`  void`
	`)`

Service ID [hex]	0x01
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	De-initializes the Watchdog Manager.

Available via	WdgM.h

⌋(SRS_BSW_00310, SRS_BSW_00336)

This function deinitializes the Watchdog Manager module and updates the trigger conditions of all Watchdog Drivers via a mode switch (see [SWS_WdgM_00154]).

Note this service is needed as a consequence of the concept “Windowed Watchdogs”. Before the Watchdog Manager module stops working, it has to set the trigger conditions of all running watchdogs to a timeout that allows the rest of the shutdown to be executed without a watchdog reset.

There are optional checks that are executed if and only if `WdgMDevErrorDetect` is enabled.

[SWS_WdgM_00288] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled: The function `WdgM_DeInit` shall report the error to default error tracer with error code `WDGM_E_UNINIT`, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00388] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is disabled: The function `WdgM_DeInit` shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

8.3.3 WdgM_GetVersionInfo

[SWS_WdgM_00153]⌈

Service Name	WdgM_GetVersionInfo

Syntax	`void WdgM_GetVersionInfo (`
	`  Std_VersionInfoType* VersionInfo`
	`)`

Service ID [hex]	0x02
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters	None

(inout)		
Parameters (out)	Version Info	Pointer to where to store the version information of the module WdgM.
Return value	None	
Description	Returns the version information of this module.	
Available via	WdgM.h	

⌋(SRS_BSW_00310)

[SWS_WdgM_00256] ⌈If the `WdgMDevErrorDetect` [ECUC_WdgM_00301] switch is enabled, the function `WdgM_GetVersionInfo` shall check if a NULL pointer is passed for the `VersionInfo` parameter. In case of an error the remaining function `WdgM_GetVersionInfo` shall not be executed and the function `WdgM_GetVersionInfo` shall report development error code `WDGM_E_INV_POINTER` to the `Det_ReportError` service of the Default Error Tracer.⌋(SRS_BSW_00323, SRS_BSW_00350)

8.3.4 WdgM_SetMode

[SWS_WdgM_00154]⌈

Service Name	WdgM_SetMode	

Syntax	`Std_ReturnType WdgM_SetMode (`
	`  WdgM_ModeType Mode`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Mode	One of the configured Watchdog Manager modes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Successfully changed to the new mode E_NOT_OK: Changing to the new mode failed
Description	Sets the current mode of Watchdog Manager.	
Available via	WdgM.h	

⌋(SRS_BSW_00310, SRS_ModeMgm_09110)

The behavior of this service and the corresponding functional requirements are described in chapter 7.5.

[SWS_WdgM_00145] ⌈The Watchdog Manager module shall only execute the service WdgM_SetMode if the *Global Supervision Status* is equal to [WDGM_GLOBAL_STATUS_OK or WDGM_GLOBAL_STATUS_FAILED.⌋ (SRS_ModeMgm_09158)

[SWS_WdgM_00142] ⌈If the function WdgM_SetMode [SWS_WdgM_00154] fails because a call to WdgIf_SetMode service fails [SWS_WdgM_00139], the Watchdog Manager shall report to the Default Error Tracer a runtime error with the value WDGM_E_SET_MODE.⌋(SRS_BSW_00339, SRS_BSW_00452)

[SWS_WdgM_00031] ⌈If disabling the watchdog is not allowed by setting the parameter WdgMOffModeEnabled [ECUC_WdgM_00340] to FALSE, the routine shall check if the requested mode would disable the watchdog (WDGIF_OFF_MODE). In this case (i.e. it would disable while it is not allowed),

1. The mode switch shall not be executed.
2. The error shall be reported to the Default Error Tracer with the runtime error code WDGM_E_DISABLE_NOT_ALLOWED.
3. The routine shall return the value E_NOT_OK.

⌋(SRS_BSW_00323, SRS_BSW_00452, SRS_ModeMgm_09109)

There are optional checks that are executed if and only if WdgMDevErrorDetect is enabled.

[SWS_WdgM_00020] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled, the parameter Mode shall be checked for being in the allowed range. In case of an error, the mode switch shall not be executed and the error shall be reported to the Default Error Tracer with the value WDGM_E_PARAM_MODE.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00021] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled: The function WdgM_SetMode shall report the error to default error tracer with error code WDGM_E_UNINIT, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋(SRS_BSW_00323, SRS_BSW_00350, SRS_BSW_00406)

[SWS_WdgM_00392] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is disabled: The function WdgM_SetMode shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

8.3.5 WdgM_GetMode

[SWS_WdgM_00168]⌈

Service Name	WdgM_GetMode	

Syntax	`Std_ReturnType WdgM_GetMode (`
	`  WdgM_ModeType* Mode`
	`)`	

Service ID [hex]	0x0b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	Mode	Current mode of the Watchdog Manager.
Return value	Std_ReturnType	E_OK: Current mode successfully returned E_NOT_OK: Returning current mode failed
Description	Returns the current mode of the Watchdog Manager.	
Available via	WdgM.h	

⌋(SRS_BSW_00310)

[SWS_WdgM_00170] ⌈The WdgM_GetMode service shall return the currently active mode of the Watchdog Manager. If the WdgM_SetMode service is active while this service is called, WdgM_GetMode shall return the previously active mode as long as the new mode has not been completely activated.⌋()

There are optional checks that are executed if and only if WdgMDevErrorDetect is enabled.

[SWS_WdgM_00253] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled: The function WdgM_GetMode shall report the error to default error tracer with error code WDGM_E_UNINIT, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00395] ⌈ If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is disabled: The function `WdgM_GetMode` shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00254] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled, the routine shall check if NULL pointers are passed for OUT parameters. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code `WDGM_E_INV_POINTER`.⌋(SRS_BSW_00323, SRS_BSW_00350)

8.3.6 WdgM_CheckpointReached

[SWS_WdgM_00263]⌈

Service Name	WdgM_CheckpointReached	

Syntax	`Std_ReturnType WdgM_CheckpointReached (`
	`  WdgM_SupervisedEntityIdType SEID,`
	`  WdgM_CheckpointIdType CheckpointID`
	`)`	

Service ID [hex]	0x0e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	SEID	Identifier of the Supervised Entity that reports a Checkpoint.
	CheckpointID	Identifier of the Checkpoint within a Supervised Entity that has been reached.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: Successfully updated alive counter E_NOT_OK: Update failed
Description	Indicates to the Watchdog Manager that a Checkpoint within a Supervised Entity has been reached.	
Available via	WdgM.h	

⌋(SRS_BSW_00310)

[SWS_WdgM_00321] ⌈The function WdgM_CheckpointReached() shall increment the *Alive Counter* of reported *Checkpoint*.⌋()

[SWS_WdgM_00322] ⌈The function WdgM_CheckpointReached() shall perform the *Deadline Supervision* (detection of early arrivals and delays) for the reported *Supervised Entity* using the reported *Checkpoint*. The output shall be an updated result of *Deadline Supervision* for the *Supervised Entity*.⌋(RS_HM_09235)

[SWS_WdgM_00323] ⌈The function WdgM_CheckpointReached() shall perform the *Logical Supervision* for the reported Supervised Entity using the reported *Checkpoint*. The output shall be an updated result of *Logical Supervision* for the *Supervised Entity*.⌋()

[SWS_WdgM_00319] ⌈The routine shall check if *Supervised Entity* to which the parameter CheckpointID belongs, is activated in the current mode. In case of an error (i.e. the *Supervised Entity* is deactivated in the current mode), the service shall return with E_NOT_OK without any action, and the error shall be reported to the Default Error Tracer with the runtime error code WDGM_E_SEDEACTIVATED.⌋(SRS_BSW_00452)

There are optional checks that are executed if and only if `WdgMDevErrorDetect` is enabled.

[SWS_WdgM_00394] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is disabled: The function `WdgM_CheckpointReached` shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00278] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled, the parameter `SEID` shall be checked for being in the list of the entities under control of the Watchdog Manager. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code `WDGM_E_PARAM_SEID`.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00279] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled: The function `WdgM_CheckpointReached` shall report the error to default error tracer with error code `WDGM_E_UNINIT`, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00396] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is disabled: The function `WdgM_CheckpointReached` shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00284] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled, the routine shall check if the parameter CheckpointID is within the set of *Checkpoints* (see [ECUC_WdgM_00303]) associated with the *Supervised Entity* given by the parameter `SEID`. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code `WDGM_E_CPID`.⌋(SRS_BSW_00323, SRS_BSW_00350)

8.3.7 WdgM_GetLocalStatus

[SWS_WdgM_00169]⌈

Service Name	WdgM_GetLocalStatus	

Syntax	`Std_ReturnType WdgM_GetLocalStatus (`
	`  WdgM_SupervisedEntityIdType SEID,`
	`  WdgM_LocalStatusType* Status`
	`)`	

Service ID [hex]	0x0c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	SEID	Identifier of the supervised entity whose supervision status shall be returned.
Parameters (inout)	None	
Parameters (out)	Status	Supervision status of the given supervised entity.
Return value	Std_Return-Type	E_OK: Current supervision status successfully returned E_NOT_OK: Returning current supervision status failed
Description	Returns the supervision status of an individual Supervised Entity.	
Available via	WdgM.h	

⌋(SRS_BSW_00310)

[SWS_WdgM_00171] ⌈The `WdgM_GetLocalStatus` service shall return the individual supervision status of the given *Supervised Entity*.⌋()

There are optional checks that are executed if and only if `WdgMDevErrorDetect` is enabled.

[SWS_WdgM_00172] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled, the parameter `SEID` shall be checked for being in the list of entities under control of the Watchdog Manager. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code `WDGM_E_PARAM_SEID`.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00257] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled, the routine shall check if NULL pointers are passed for OUT parameters. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code `WDGM_E_INV_POINTER`.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00173] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled: The function WdgM_GetLocalStatus shall report the error to default error tracer with error code WDGM_E_UNINIT, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00397] ⌈ If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is disabled: The function WdgM_GetLocalStatus shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

8.3.8 WdgM_GetGlobalStatus

[SWS_WdgM_00175]⌈

Service Name	WdgM_GetGlobalStatus	

Syntax	`Std_ReturnType WdgM_GetGlobalStatus (`
	`  WdgM_GlobalStatusType* Status`
	`)`	

Service ID [hex]	0x0d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	Status	Global supervision status of the Watchdog Manager.
Return value	Std_ReturnType	E_OK: Current supervision status successfully returned E_NOT_OK: Returning current supervision status failed
Description	Returns the global supervision status of the Watchdog Manager.	
Available via	WdgM.h	

⌋(SRS_BSW_00310)

[SWS_WdgM_00344] ⌈If development error detection for the Watchdog Manager module is enabled, then the function WdgM_GetGlobalStatus shall check whether the parameter Status is a NULL pointer (NULL_PTR). If Status is a NULL pointer, then the function shall raise the development error WDGM_E_INV_POINTER (i.e. invalid pointer), without any further effect.⌋()

There are optional checks that are executed if and only if WdgMDevErrorDetect is enabled.

[SWS_WdgM_00258] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled, the routine shall check if NULL pointers are passed for OUT parameters. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code WDGM_E_INV_POINTER.⌋(SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00176] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled, the routine shall check if the Watchdog Manager is initialized. In case of an error, the service shall not be executed and the error shall be reported to the Default Error Tracer with the error code WDGM_E_UNINIT.⌋(SRS_BSW_00323, SRS_BSW_00350)

8.3.9 WdgM_PerformReset

[SWS_WdgM_00264]⌈

Service Name	WdgM_PerformReset

Syntax	`void WdgM_PerformReset (`
	`  void`
	`)`

Service ID [hex]	0x0f
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Instructs the Watchdog Manager to cause a watchdog reset.
Available via	WdgM.h

⌋(SRS_BSW_00310, SRS_ModeMgm_09232)

[SWS_WdgM_00232] ⌈When this service is called, the Watchdog Manager shall set the trigger condition for all configured Watchdog Drivers to 0 (zero).⌋()

Thereby, the hardware watchdogs will cause an external hardware reset.

[SWS_WdgM_00233] ⌈After this service has been called, the Watchdog Manager shall not update the trigger condition anymore.⌋()

When this API has been called, *Global Supervision Status* is not considered anymore.

There are optional checks that are executed if and only if `WdgMDevErrorDetect` is enabled.

[SWS_WdgM_00270] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled: The function `WdgM_PerformReset` shall report the error to default error tracer with error code `WDGM_E_UNINIT`, without any further effect, if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED.⌋ (SRS_BSW_00323, SRS_BSW_00350)

[SWS_WdgM_00401] ⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is disabled: The function `WdgM_PerformReset` shall return without any effect if the Watchdog Manager is in WDGM_GLOBAL_STATUS_DEACTIVATED. ⌋ (SRS_BSW_00323, SRS_BSW_00350)

8.3.10 WdgM_GetFirstExpiredSEID

[SWS_WdgM_00346]⌈

Service Name	WdgM_GetFirstExpiredSEID

Syntax	`Std_ReturnType WdgM_GetFirstExpiredSEID (`
	`  WdgM_SupervisedEntityIdType* SEID`
	`)`

Service ID [hex]	0x10
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None

Parameters (inout)	None	
Parameters (out)	SEID	Identifier of the supervised entity that first reached the state WDGM_LOCAL_STATUS_EXPIRED.
Return value	Std_ReturnType	E_OK: SEID successfully returned E_NOT_OK: Error when returning the SEID
Description	Returns SEID that first reached the state WDGM_LOCAL_STATUS_EXPIRED.	
Available via	WdgM.h	

⌋()

[SWS_WdgM_00347] ⌈If development error detection for the Watchdog Manager module is enabled, then the function `WdgM_GetFirstExpiredSEID()` shall check whether the parameter `SEID` is a NULL pointer (NULL_PTR). If `Status` is a `NULL` pointer, then the function shall raise the development error `WDGM_E_INV_POINTER` (i.e. invalid pointer), without any further effect.⌋()

[SWS_WdgM_00348] ⌈The function `WdgM_GetFirstExpiredSEID()` shall be available before `WdgM_Init`.⌋()

[SWS_WdgM_00349] ⌈The function `WdgM_GetFirstExpiredSEID()` shall read the `SEID` from non-initialized RAM location, stored as a double-inverse value. In case the value and the inverse value do not correspond to each other, then the function shall return `E_NOT_OK` and shall write 0 to `*SEID`. In case the value and the inverse value correspond, the function shall return `E_OK` and set write the read value to `*SEID`.⌋()

8.4 Call-back Notifications

Not Applicable

8.5 Scheduled Functions

These functions are directly called by Basic Software Scheduler.

8.5.1 WdgM_MainFunction

[SWS_WdgM_00159]⌈

Service Name	WdgM_MainFunction

Syntax	`void WdgM_MainFunction (`
	`  void`
	`)`

Service ID [hex]	0x08
Description	Performs the processing of the cyclic Watchdog Manager jobs.
Available via	SchM_WdgM.h

⌋(SRS_BSW_00310, SRS_BSW_00373)

[SWS_WdgM_00324] ⌈The function `WdgM_MainFunction()` shall perform the *Alive Supervision* for the reported *Supervised Entity* using the reported *Checkpoint*. The input of this function shall be the *Alive Counters* of the *Checkpoint*. The output of this function shall be the *Results* of *Alive Supervision* for the *Supervised Entity*.⌋()

[SWS_WdgM_00404] ⌈The function `WdgM_MainFunction`() shall perform the *Deadline Supervision* (detection of timeouts) for the all *Supervised Entities* with active *Deadline Supervisions* (e.g. reached a *Deadline Start Checkpoints* and before reaching the corresponding *Deadline End Checkpoint*). The output shall be an updated result of *Deadline Supervision* for the *Supervised Entity*.⌋(RS_HM_09235)

[SWS_WdgM_00325] ⌈Based on the results from *Alive*, *Deadline* and *Logical Supervision*, for each activated *Supervised Entity* the function `WdgM_MainFunction()` shall determine the *Local Supervision Status*.⌋()

[SWS_WdgM_00351] ⌈For the <u>first</u> *Supervised Entity* that switched to the state WDGM_LOCAL_STATUS_EXPIRED since the last time `WdgM_Init()` was called, the function `WdgM_MainFunction()` shall store the `SEID` of that *Supervised Entity* in a non-initialized RAM, as a double-inverted value (i.e. `SEID` and `~SEID`).⌋()

[SWS_WdgM_00326] ⌈Based on the *Local Supervision Status* of each activated *Supervised Entity*, the function `WdgM_MainFunction()` shall determine the *Global Supervision Status*.⌋()

[SWS_WdgM_00415] {DRAFT} ⌈If multiple Main Functions were configured (see `WdgMMainFunction`), each Main Function shall have function name

WdgM_MainFunction_<shortName>. The suffix <shortName> shall be derived from the short name of the WdgMMainFunction configuration container in the ECU configuration.⌋()

[SWS_WdgM_00039] ⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled, the routine shall check if the Watchdog Manager is initialized. In case of an error, the main function shall not be executed and the development error shall be reported to the Default Error Tracer with the error code WDGM_E_UNINIT.⌋(SRS_BSW_00323, SRS_BSW_00350, SRS_BSW_00406)

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

Note: BswM_WdgM_RequestPartitionReset has been set to obsolete since R21-11.

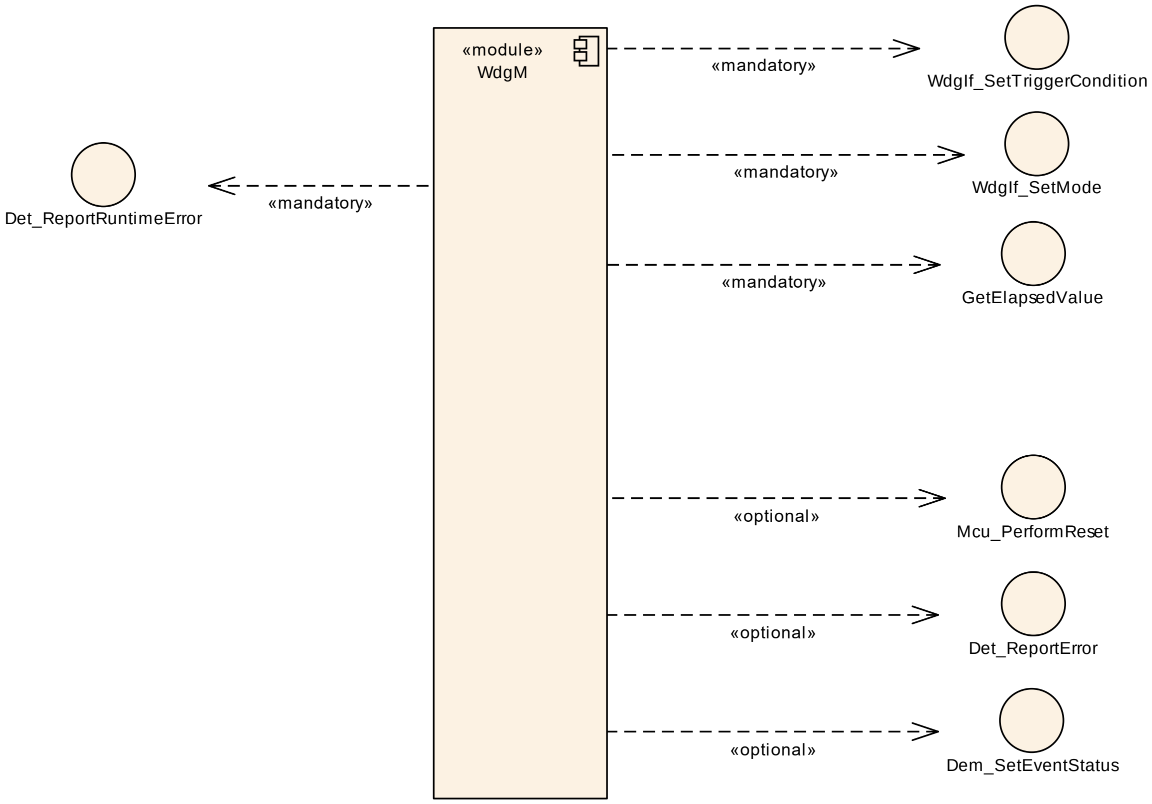

Figure 13: Expected Interfaces

8.6.1 Mandatory Interfaces

This chapter defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_WdgM_00161]⌈

API Function	***Header File***	***Description***
Det_Report-RuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
GetElapsedValue	Os.h	This service gets the number of ticks between the current tick value and a previously read tick value.
WdgIf_SetMode	WdgIf.h	Map the service WdgIf_SetMode to the service Wdg_SetMode of the corresponding Watchdog Driver.
WdgIf_Set-TriggerCondition	WdgIf.h	Map the service WdgIf_SetTriggerCondition to the service Wdg_Set TriggerCondition of the corresponding Watchdog Driver.

⌋()

8.6.2 Optional Interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

[SWS_WdgM_00162]⌈

API Function	***Header File***	***Description***
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEvent Status can safely ignore the return value. This API will be available only if ({Dem/DemConfigSet/DemEventParameter/ DemEventReportingType} == STANDARD_REPORTING)
Det_ReportError	Det.h	Service to report development errors.
Mcu_PerformReset	Mcu.h	The service performs a microcontroller reset.
SwCluC_BManif_Get-ConSwClusterId_<-ResourceEntryGroup>-_<Handle> (draft)	SwCluC_ BManif.h	Returns the Id of the connected Software Cluster for a Notifier Handle of a Provide Resource Entry or a Provide Handle of a Require Resource Entry **Tags:** atp.Status=draft
SwCluC_BManif_Get-ConSwClusterId_<-ResourceEntryGroup>-_<ResourceEntry>_<-Handle> (draft)	SwCluC_ BManif.h	Returns the Id of the connected Software Cluster for a Notifier Handle of a Provide Resource Entry or a Provide Handle of a Require Resource Entry **Tags:** atp.Status=draft
SwCluC_BManif_Get-Handle_<Resource-	SwCluC_ BManif.h	Returns a handle of a Resource Entry in a Resource Entry Group

EntryGroup>_<Handle> (draft)		**Tags:** atp.Status=draft
SwCluC_BManif_Get-Handle_<Resource-EntryGroup>_<-ResourceEntry>_<-Handle> (draft)	SwCluC_ BManif.h	Returns a handle of a Resource Entry in a Resource Entry Group **Tags:** atp.Status=draft
SwCluC_BManif_Get-NoOfHandleSets_<-Resource Entry Group> (draft)	SwCluC_ BManif.h	Returns the number of actually used - and thereby connected - handle sets. **Tags:** atp.Status=draft
SwCluC_BManif_Get-NoOfHandleSets_<-Resource Entry Group>-_<Resource Entry> (draft)	SwCluC_ BManif.h	Returns the number of actually used - and thereby connected - handle sets **Tags:** atp.Status=draft

⌋()

8.6.3 Configurable Interfaces

Not Applicable

8.6.4 Job End Notification

Not Applicable

8.7 Service Interfaces

This chapter specifies the AUTOSAR Interfaces which are provided by the Watchdog Manager module. The SW-C description of the Watchdog Manager Service will define the Watchdog Manager ports available to SW-Cs and CDDs. Each AUTOSAR SW-C or CDD that uses the service must contain service ports in its own description. These ports are typed with the same interfaces and have to be connected to the ports of the Watchdog Manager module, so that the RTE can generate the appropriate IDs and the required symbols.

The *Local Supervision Status* and the *Global Supervision Status* of the Watchdog Manager module are reported to SW-Cs and CDDs through mode ports. An SW-C and CDD can define its own mode port with the same interface as the mode ports of the Watchdog Manager module. Afterwards the SW-C or CDD can query the status and will be informed of status changes via the mode port. In addition, the SW-C can define Runnables that are started or stopped by the RTE because of status changes.

BSW modules can call the WdgM API functions directly and taking into account the mapping by RTE, or call them via Service Ports using RTE.

[SWS_WdgM_00416] {DRAFT} ⌈For clustered software architecture (one Host Software Cluster and zero or more Application Software Cluster), Host Software Cluster shall provide all Ports and corresponding Port Interfaces with all Operations and ModeGroups which are permanently available or enabled by configuration.⌋()

[SWS_WdgM_00417] {DRAFT} ⌈For clustered software architecture, Application Software Cluster shall provide following Ports and corresponding Port Interfaces with listed Operations and ModeGroups which are permanently available or enabled by configuration.

- Port: `localSupervision_{SupervisedEntityCheckpointName}` [SWS_WdgM_00147] (Port Interface: `WdgM_LocalSupervision` [SWS_WdgM_00333] with the Operation: `CheckpointReached`)
- Port: `globalSupervision` [SWS_WdgM_91002] (Port Interface: `WdgM_GlobalSupervision` [SWS_WdgM_91001] with the Operation: `GetMode`)
- Port: `mode_{SupervisedEntityName}` [SWS_WdgM_00149] (Port Interface: `WdgM_LocalMode` [SWS_WdgM_00335] with the ModeGroup: `currentMode`)

⌋()

8.7.1 Ports and Port Interface for Supervision

8.7.1.1 General Approach

To reduce the number of ports provided by the Watchdog Manager module all interfaces between SW-Cs / CDD and the service are modeled as Client/Server communication. To report *Checkpoints* the sender-receiver paradigm may seem more appropriate, but this kind of modeling would double the number of ports. Therefore, also for this functionality, the Client/Server paradigm has been chosen.

The unique *Supervised Entity* IDs are used to identify the *Supervised Entities* within an ECU. In order to keep the application code independent of the configuration of ECU-dependent *Supervised Entity* IDs, the IDs used by SW-Cs and CDDs are not modeled explicitly as data elements to be passed between SW-C and service. These IDs are modeled as “port defined argument values” of the Provide Ports of the Watchdog Manager module. As a consequence, the *Supervised Entity* IDs will not show up as arguments in the operations of the client-server interface. As a further consequence for this approach, there will be separate ports for each *Supervised Entity*.

8.7.1.2 Data Types

The information passed between the application and the service are:

1. ID to identify a *Supervised Entity* (as port defined argument value) and
2. ID to identify a *Checkpoint*.

The type for this *Supervised Entity Identifier* shall be based on the type *WdgM_SupervisedEntityIdType*. This type is defined as `uint16`. Therefore, the following type description is required:

[SWS_WdgM_00356]⌈

Name	WdgM_SupervisedEntityIdType		
Kind	Type		
Derived from	uint16		
Range	0-<Number of Supervised Entities>	--	The range of valid IDs depends on the number of configured Supervised Entities.
Description	This type identifies an individual Supervised Entity for the Watchdog Manager.		
Variation	--		
Available via	Rte_WdgM_Type.h		

⌋()

The type for this *Checkpoint Identifier* shall be based on the type *WdgM_CheckpointIdType*. This type is defined as `uint16`. Therefore, the following type description is required:

[SWS_WdgM_00357]⌈

Name	WdgM_CheckpointIdType		
Kind	Type		
Derived from	uint16		
Range	0-<Maximum number of Checkpoints>	--	The range of valid IDs depends on the maximum number of configured Checkpoints within all configured Supervised Entities.
Description	This type identifies a Checkpoint in the context of a Supervised Entity for the Watchdog Manager. Note that an individual Checkpoint can only be identified by the pair of Supervised Entity ID and Checkpoint ID.		
Variation	--		
Available via	Rte_WdgM_Type.h		

⌋()

Beware, that the *Checkpoint* ID by itself is not unique. Only the pair of *Supervised Entity* ID and *Checkpoint* ID uniquely identifies a *Checkpoint*.

8.7.1.3 Port Interfaces

All operations are put into two interfaces (one with operations specific for an individual *Supervised Entity*, and one for global WdgM operations).

[SWS_WdgM_00333]⌈

Name	WdgM_LocalSupervision		
Comment	--		
IsService	true		
Variation	--		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	CheckpointReached
Comment	Indicates to the Watchdog Manager that a Checkpoint within a Supervised Entity has been reached.
Mapped to API	WdgM_CheckpointReached
Variation	--
Possible Errors	E_OK E_NOT_OK

⌋()

[SWS_WdgM_91004]⌈

Name	WdgM_LocalSupervisionStatus		
Comment	--		
IsService	true		
Variation	--		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	GetLocalStatus	
Comment	Returns the supervision status of an individual Supervised Entity.	
Mapped to API	WdgM_GetLocalStatus	
Variation	--	
Parameters	Status	
	Type	WdgM_LocalStatusType
	Direction	OUT
	Comment	Supervision status of the given supervised entity.
	Variation	--
Possible Errors	E_OK E_NOT_OK	

⌋()

[SWS_WdgM_91001]⌈

Name	WdgM_GlobalSupervision		
Comment	--		
IsService	true		
Variation	--		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	GetFirstExpiredSEID	
Comment	Returns SEID that first reached the state WDGM_LOCAL_STATUS_EXPIRED.	
Mapped to API	WdgM_GetFirstExpiredSEID	
Variation	--	
Parameters	SEID	
	Type	WdgM_SupervisedEntityIdType
	Direction	OUT
	Comment	Identifier of the supervised entity that first reached the state WDGM_LOCAL_STATUS_EXPIRED.
	Variation	--
Possible	E_OK	

Errors	E_NOT_OK

Operation	GetGlobalStatus	
Comment	Returns the global supervision status of the Watchdog Manager.	
Mapped to API	WdgM_GetGlobalStatus	
Variation	--	
Parameters	Status	
	Type	WdgM_GlobalStatusType
	Direction	OUT
	Comment	Global supervision status of the Watchdog Manager.
	Variation	--
Possible Errors	E_OK E_NOT_OK	

Operation	GetMode	
Comment	Returns the current mode of the Watchdog Manager.	
Mapped to API	WdgM_GetMode	
Variation	--	
Parameters	Mode	
	Type	WdgM_ModeType
	Direction	OUT
	Comment	Current mode of the Watchdog Manager.
	Variation	--
Possible Errors	E_OK E_NOT_OK	

Operation	PerformReset
Comment	Instructs the Watchdog Manager to cause a watchdog reset.
Mapped to API	WdgM_PerformReset
Variation	--
Possible Errors	--

Operation	SetMode

Comment	Sets the current mode of Watchdog Manager.	
Mapped to API	WdgM_SetMode	
Variation	--	
Parameters	Mode	
	Type	WdgM_ModeType
	Direction	IN
	Comment	One of the configured Watchdog Manager modes.
	Variation	--
Possible Errors	E_OK E_NOT_OK	

⌋()

Compared to the API, the "`WdgM_`" prefix in the names is not required, because the names given here will show up in the XML not globally but as part of an interface description.

8.7.1.4 Service Ports

Figure 14 shows how AUTOSAR Software components (single or multiple instances) are connected via service ports to the Watchdog Manager module. On the left side, there are two instances (`swc1` and `swc2`) of component `SWC Type A` and one instance (`swc3`) of component `SWC Type B`.

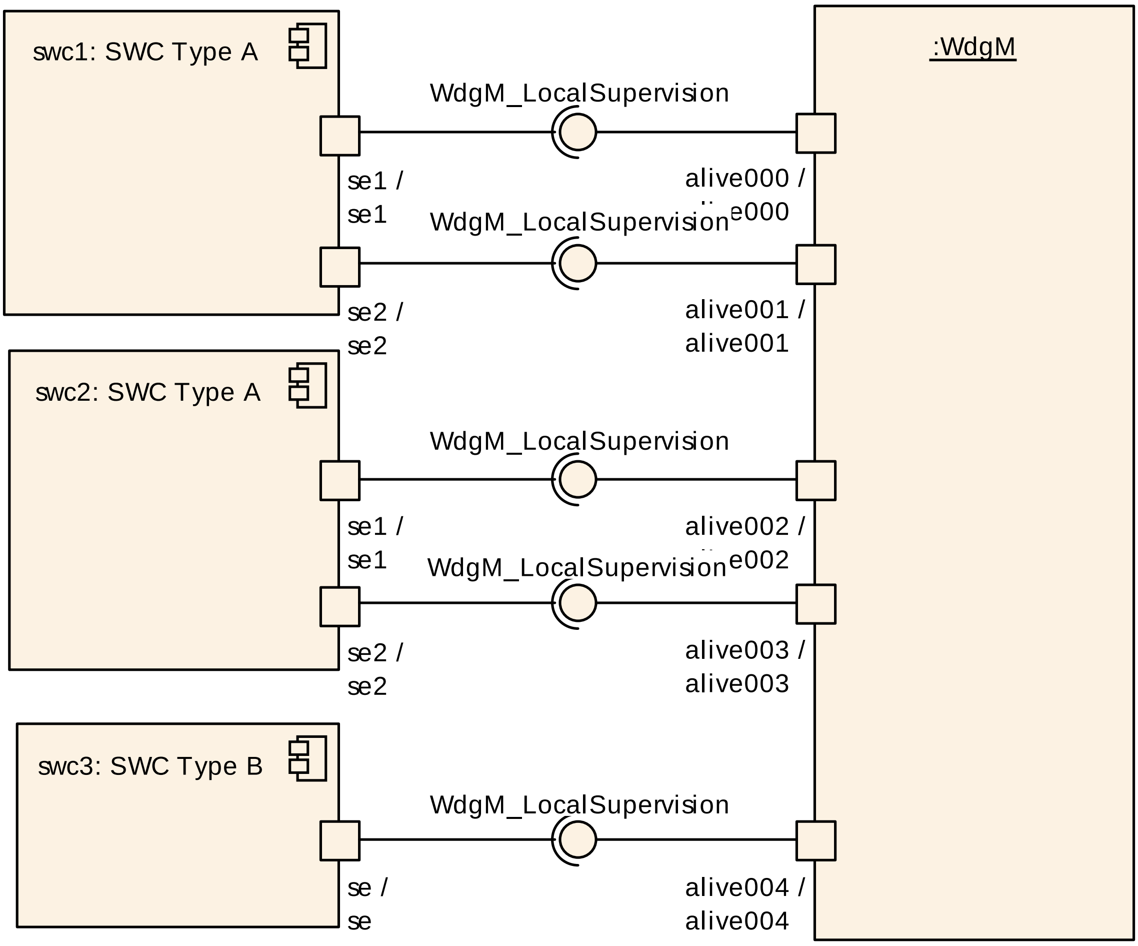

Figure 14: Example of SW-Cs connected to the Watchdog Manager via service ports

On the Watchdog Manager side, there is one port per *Supervised Entity* providing all the services of the interface `WdgM_AliveSupervision` described above. Each *Supervised Entity* has one port for requiring those services for each *Supervised Entity* associated with that application.

[SWS_WdgM_00146] ⌈The Watchdog Manager module shall provide a single service port for *Supervision* for each *Supervised Entity* that is configured.

To be able to match a *Supervision* port with its corresponding mode port for Status Reporting, a naming convention is necessary.⌋()

The *Local Supervision* ports of the Watchdog Manager module is named as follows:

[SWS_WdgM_00147]⌈

Name	localSupervision_{SupervisedEntityCheckpointName}		
Kind	ProvidedPort	***Interface***	WdgM_LocalSupervision
Description	This port provides the Supervision interface of one Supervised Entity Checkpoint		

	to a SWC.	
Port Defined Argument Value(s)	***Type***	WdgM_SupervisedEntityIdType
	Value	{ecuc(WdgM/WdgMGeneral/WdgMSupervisedEntity/WdgMSupervisedEntityId.value)}
	Type	WdgM_CheckpointIdType
	Value	ecuc{WdgM/WdgMGeneral/WdgMSupervisedEntity/WdgMCheckpoint/WdgMCheckpointId}
Variation	SupervisedEntityCheckpointName = {ecuc(WdgM/WdgMGeneral/WdgMSupervisedEntity.SHORT-NAME)}_{ecuc(WdgM/WdgMGeneral/WdgMSupervisedEntity/WdgMCheckpoint.SHORT-NAME)}	

⌋()

[SWS_WdgM_91003]⌈

Name	localSupervisonStatus_{SupervisedEntityName}		
Kind	ProvidedPort	***Interface***	WdgM_LocalSupervisionStatus
Description	This port provides the Supervision status interface of one Supervised Entity to a SWC.		
Port Defined Argument Value(s)	***Type***	WdgM_SupervisedEntityIdType	
	Value	{ecuc(WdgM/WdgMGeneral/WdgMSupervisedEntity/WdgMSupervisedEntityId.value)}	
Variation	SupervisedEntityName = {ecuc(WdgM/WdgMGeneral/WdgMSupervisedEntity.SHORT-NAME)}		

⌋()

The *Global Supervision* ports of the Watchdog Manager module is named as follows:

[SWS_WdgM_91002]⌈

Name	globalSupervision		
Kind	ProvidedPort	***Interface***	WdgM_GlobalSupervision
Description	This port provides the Global Supervision interface of the WdgM.		
Variation	--		

⌋()

8.7.1.5 Error Codes

The *Supervision* service does not return any service specific error codes.

8.7.2 Ports and Port Interface for Status Reporting

8.7.2.1 General Approach

To control the state-dependent behavior of SW-Cs and CDDs, the RTE provides the mechanism of mode ports. A mode manager can switch between different modes that are defined in the mode port. The SW-C / CDD that connects to the mode port can use the mode information in two ways:

- The SW-C / CDD can query the current mode via the mode port.
- The SW-C / CDD can declare Runnables that are started or stopped by the RTE because of mode changes.

According to RTE Specification [5] a mode port has a `ModeSwitchInterface`. The mode manager, here the Watchdog Manager module, is the sender and the SW-Cs are the receivers.

The Watchdog Manager module uses mode ports to provide two kinds of information:

- First, it provides the *Local Supervision Status* of each *Supervised Entity*. Therefore, the Watchdog Manager module has a mode port for each *Supervised Entity*.
- Second, the Watchdog Manager module provides the *Global Supervision Status* which reflects the combined *Supervision Status* of all *Supervised Entities*. Therefore, it has one additional mode port.

8.7.2.2 Data Types

The mode declaration group `WdgM_Mode` represents the modes of the Watchdog Manager module that will be notified to the SW-Cs / CDDs and the RTE.

[SWS_WdgM_00334]⌈

Name	WdgM_Mode	
Kind	ModeDeclarationGroup	
Category	EXPLICIT_ORDER	
Initial mode	SUPERVISION_OK	
On transition value	255	
Modes	SUPERVISION_OK	0
	SUPERVISION_FAILED	1
	SUPERVISION_EXPIRED	2
	SUPERVISION_STOPPED	3
	SUPERVISION_DEACTIVATED	4
Description	The category of ModeDeclarationGroup WdgM_Mode is EXPLICIT_ORDER, The	

	attribute value for the ModeDeclaration are set as following: "SUPERVISION_OK" = 0 "SUPERVISION_FAILED" = 1 "SUPERVISION_EXPIRED" = 2 "SUPERVISION_STOPPED" = 3 "SUPERVISION_DEACTIVATED" = 4 The onTransitionValue is defined as 255

⌋()

[SWS_WdgM_00359]⌈

Name	WdgM_LocalStatusType		
Kind	Type		
Derived from	uint8		
Range	WDGM_LOCAL_STATUS_OK	0	The supervision of this Supervised Entity has not shown any failures.
	WDGM_LOCAL_STATUS_FAILED	1	The supervision of this Supervised Entity has failed but can still be "healed". I.e., if the Supervised Entity returns to a normal behavior, its supervision state will also return to WDGM_LOCAL_STATUS_OK. Furthermore, the number of times that the supervision has failed has not yet exceeded a configurable limit. When this limit has been exceeded the state will change to WDGM_LOCAL_STATUS_EXPIRED.
	WDGM_LOCAL_STATUS_EXPIRED	2	The supervision of this Supervised Entity has failed permanently. This state cannot be left.
	WDGM_LOCAL_STATUS_DEACTIVATED	4	The supervision of this Supervised Entity is temporarily disabled.
Description	This type shall be used for variables that represent the current status of supervision for individual Supervised Entities.		
Variation	--		
Available via	Rte_WdgM_Type.h		

⌋()

[SWS_WdgM_00360]⌈

Name	WdgM_GlobalStatusType
Kind	Type
Derived from	uint8

Range	WDGM_GLOBAL_STATUS_OK	0	Supervision did not show any failures.
	WDGM_GLOBAL_STATUS_FAILED	1	Supervision has failed but is still within the limit of allowed failures.
	WDGM_GLOBAL_STATUS_EXPIRED	2	Supervision has failed, the allowed limit of failures has been exceeded, but the Watchdog Driver has not yet been instructed to stop triggering.
	WDGM_GLOBAL_STATUS_STOPPED	3	Supervision has failed, the allowed limit of failures has been exceeded, and the Watchdog Driver has been instructed to stop triggering. A watchdog reset is about to happen.
	WDGM_GLOBAL_STATUS_DEACTIVATED	4	WdgM is not initialized and therefore will not manage the watchdogs.
Description	This type shall be used for variables that represent the global supervision status of the Watchdog Manager module.		
Variation	--		
Available via	Rte_WdgM_Type.h		

⌋()

[SWS_WdgM_00358]⌈

Name	WdgM_ModeType		
Kind	Type		
Derived from	uint8		
Range	0-<Number of Modes>	--	The actual upper limit depends on the number of configured modes for Watchdog Manager.
Description	This type distinguishes the different modes that were configured for the Watchdog Manager.		
Variation	--		
Available via	Rte_WdgM_Type.h		

⌋()

8.7.2.3 Port Interfaces

There are two different interfaces to indicate changes in the *Supervision Status* to interested SW-Cs / CDDs and the RTE.

The interface `WdgM_LocalMode` is used to signal the *Local Supervision Status* of a single *Supervised Entity*.

[SWS_WdgM_00335]⌈

Name	WdgM_LocalMode	
Comment	--	
IsService	true	
Variation	--	
ModeGroup	currentMode	WdgM_Mode

⌋()

The interface `WdgM_GlobalMode` is used to signal the *Global Supervision Status* that is combined from all individual *Supervised Entities*.

[SWS_WdgM_00336]⌈

Name	WdgM_GlobalMode	
Comment	--	
IsService	true	
Variation	--	
ModeGroup	currentMode	WdgM_Mode

⌋()

The reason for defining two different interfaces is the way these interfaces are used. For the `WdgM_GlobalMode` interfaces the Watchdog Manager module provides only one single port with that interface. By contrast, for the `WdgM_LocalMode` interface the Watchdog Manager module provides as many ports as there are *Supervised Entities*. In order to access these ports efficiently, the Indirect Port API of the RTE can be used. This API provides a list of all ports that have the same interface, e.g.:

```
/**
  * Called within WdgM. Reports the status/mode of the SE
  * to SW-Cs / CDDs through Rte
  */
void WdgM_NotifyOKToSE(WdgM_SupervisedEntityIdType se)
{
 Rte_PortHandle_WdgM_LocalMode_P ph =
                         Rte_Ports_WdgM_LocalMode_P();
 ph[se].Switch_currentMode(RTE_MODE_WdgM_Mode_SUPERVISION_OK);
}
```

To avoid that the mode port for the *Global Supervision Status* shows up in this list, this port uses a different interface, i.e. `WdgM_GlobalMode` instead of `WdgM_LocalMode`.

8.7.2.4 Mode Ports

Figure 15 shows how AUTOSAR Software components (single or multiple instances) are connected via mode and service ports to the Watchdog Manager module. On the left side, there are two instances (`swc1` and `swc2`) of component `SWC Type A` and one instance (`swc3`) of component `SWC Type B`. Each component is connected to the mode ports that correspond to its own *Supervised Entities*. In addition, `swc3` is connected to the global mode port and can therefore react to changes in the combined *Supervision Status* of all *Supervised Entities*.

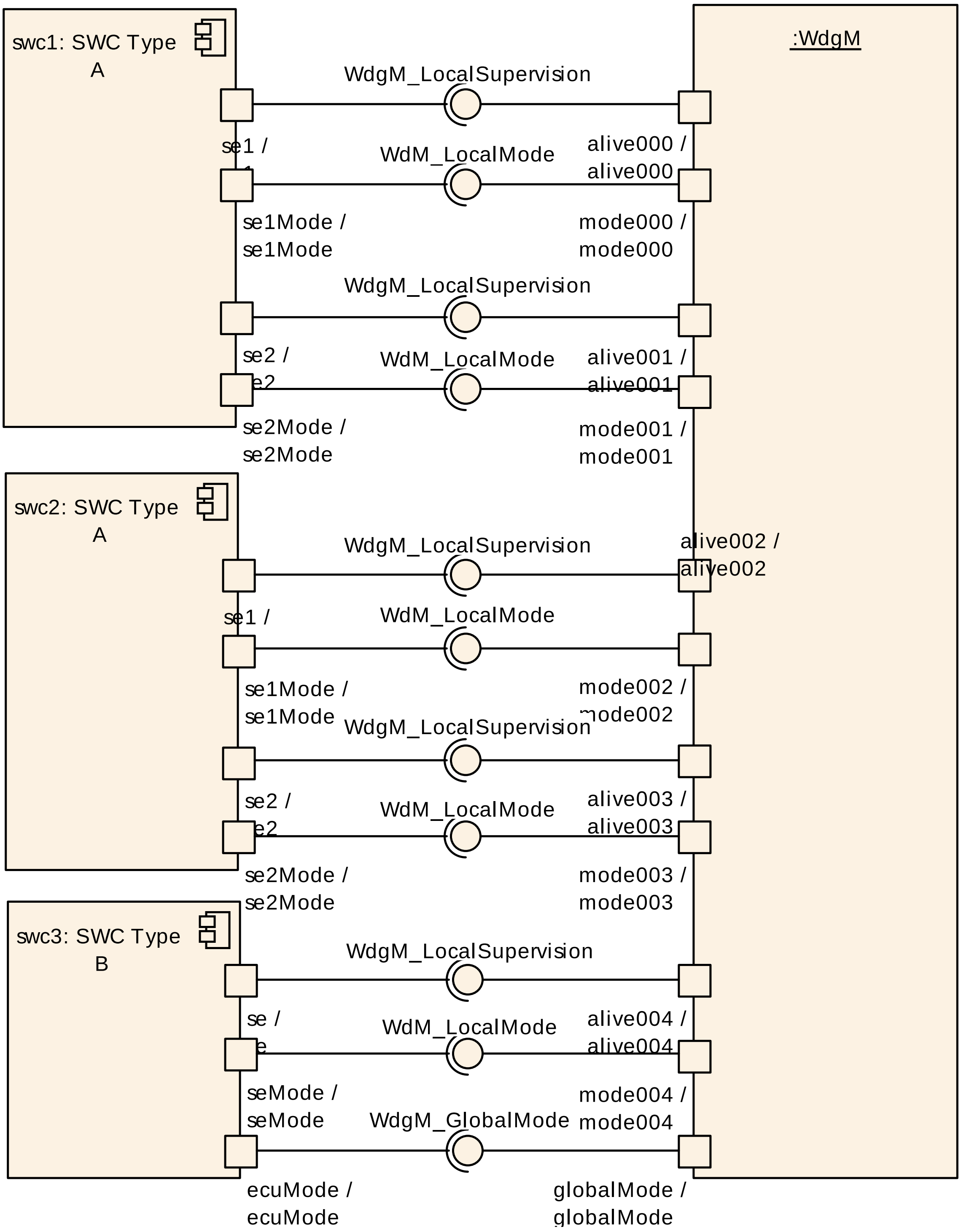

Figure 15: Example of SW-Cs connected to the Watchdog Manager via service ports and mode ports

This results in one mode port per *Supervised Entity*.

[SWS_WdgM_00148] ⌈The Watchdog Manager module shall provide a single mode port for reporting the *Local Supervision Status* of each *Supervised Entity* that is configured.

To be able to match a Supervision port with its corresponding mode port for Status Reporting, a naming convention is necessary.⌋(SRS_ModeMgm_09160, SRS_ModeMgm_09225)

The Watchdog Manager provides mode ports for reporting the *Supervision Status* of each *Supervised Entity*:

[SWS_WdgM_00149]⌈

Name	mode_{SupervisedEntityName}		
Kind	ProvidedPort	***Interface***	WdgM_LocalMode
Description	--		
Variation	SupervisedEntityName = {ecuc(WdgM/WdgMGeneral/WdgMSupervisedEntity/WdgMSupervisedEntityId.SHORT-NAME)}		

⌋()

[SWS_WdgM_00197] ⌈When the *Local Supervision Status* of a single *Supervised Entity* changes, the Watchdog Manager module shall report that change via the mode port for that *Supervised Entity* immediately after it has been recognized.⌋()

The Watchdog Manager module provides one mode port for reporting the *Global Supervision Status*:

[SWS_WdgM_00150]⌈

Name	globalmode		
Kind	ProvidedPort	***Interface***	WdgM_GlobalMode
Description	--		
Variation	--		

⌋(SRS_ModeMgm_09160, SRS_ModeMgm_09225, SRS_ModeMgm_09162)

[SWS_WdgM_00198] ⌈When the *Global Supervision Status* changes, the Watchdog Manager module shall report that change via the global mode port.⌋()

[SWS_WdgM_00199] ⌈After computing the *Global Supervision Status* from all *Local Supervision Status*, the Watchdog Manager module shall report any change in the resulting *Global Supervision Status* only once.⌋()

The resulting behavior is that first all changes in *Local Supervision Status* are reported. Afterwards the *Global Supervision Status* is reported only once and only if it changed due to the individual changes.

For instance, if in one *Supervision Cycle* SE1 goes from WDGM_LOCAL_STATUS_OK to WDGM_LOCAL_STATUS_FAILED, WDGM_LOCAL_STATUS_FAILED is reported on the local mode port for SE1. In the same *Supervision Cycle* SE2 goes from WDGM_LOCAL_STATUS_OK to WDGM_LOCAL_STATUS_EXPIRED directly, WDGM_LOCAL_STATUS_EXPIRED is reported on the local mode port for SE2. The resulting *Global Supervision Status* in this *Supervision Cycle* changes from WDGM_GLOBAL_STATUS_OK to WDGM_GLOBAL_STATUS_EXPIRED and only WDGM_GLOBAL_STATUS_EXPIRED is reported on the global mode port. In that example WDGM_GLOBAL_STATUS_FAILED is not reported on the global mode port, because it was only an intermediate state while evaluating a subset of *Supervised Entities*.

9 Sequence Diagrams

This chapter shows the interactions between the Watchdog Manager and other BSW modules as well as *Supervised Entities*.

9.1 Initialization

The diagram shows the initialization of the Watchdog Manager module. The initialization should be done at a late phase of ECU initialization after the initialization of the OS.

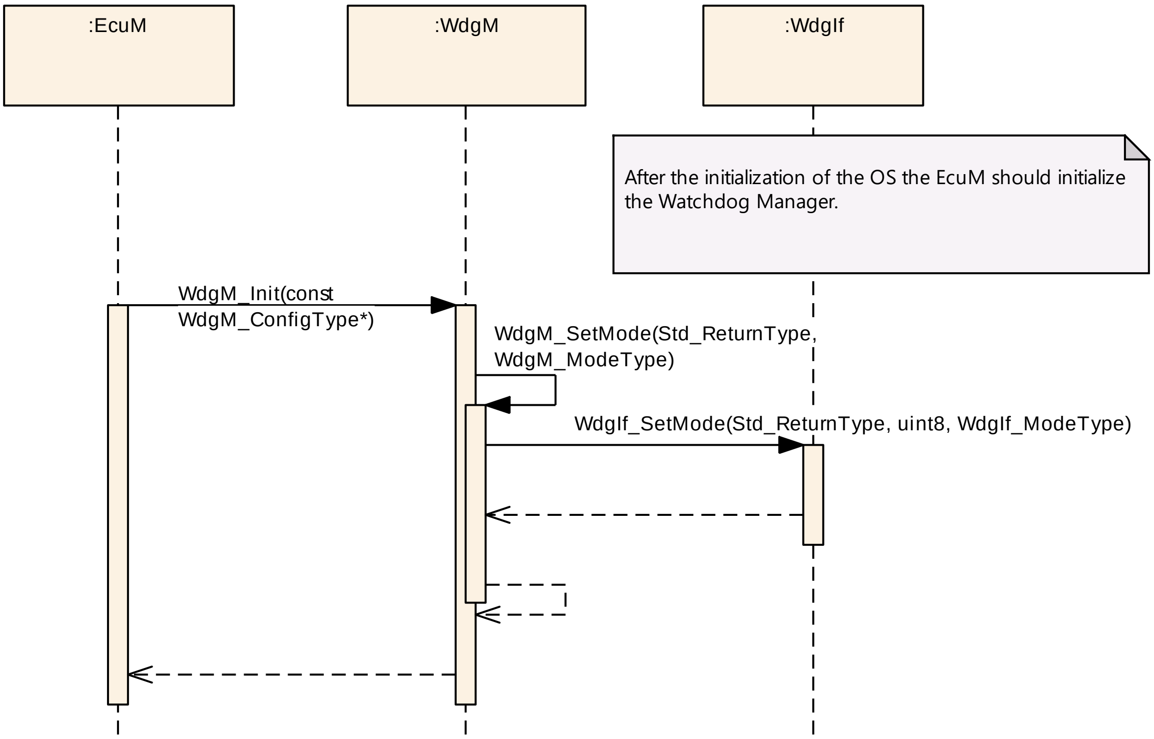
:EcuM
:WdgM
:WdgIf
After the initialization of the OS the EcuM should initialize the Watchdog Manager.
WdgM_Init(const WdgM_ConfigType*)
WdgM_SetMode(Std_ReturnType, WdgM_ModeType)
WdgIf_SetMode(Std_ReturnType, uint8, WdgIf_ModeType)

Figure 16: Initialization of the Watchdog Manager module

10 Configuration Specification

10.1 Parameter Differentiation

Within this chapter, you find a brief introduction of terms, which are used to differentiate type of configuration parameters. In the subchapter you find concrete specification issue for parameters in Watchdog Manager context.

For details refer to the chapter 10.1 “Introduction to configuration specification” in *SWS_BSWGeneral.*

10.1.1 Static Configuration Parameters

[SWS_WdgM_00025] ⌈The parameters of the Watchdog Manager module that shall minimally be configurable at system generation and / or system compile time (pre-compile).⌋(SRS_BSW_00345)

10.1.2 Runtime Configuration Parameters

[SWS_WdgM_00029] ⌈The parameters of the Watchdog Manager module that shall be configurable at post-build time.⌋()

10.1.3 Precompile Options

[SWS_WdgM_00104] ⌈The precompile options shall be used for code implementations that are not directly generated out of code generators. Therefore, the precompile options support the optimization of re-used source code-file of the Watchdog Manager module according to settings of static configuration.⌋(SRS_BSW_00345, SRS_BSW_00171)

10.2 Containers and Configuration Parameters

The following variants are supported by Watchdog Manager module:

10.2.1 Variants

For details refer to the chapter 10.1.2 “Variants” in *SWS_BSWGeneral.*

10.2.2 WdgM

SWS Item	[ECUC_WdgM_00001]
Module Name	WdgM
Description	Configuration of the WdgM (Watchdog Manager) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WdgMConfig-Set	1	This container describes one of multiple configuration sets of WdgM.
WdgMGeneral	1	Container defines all general configuration parameters of the Watchdog Manager.

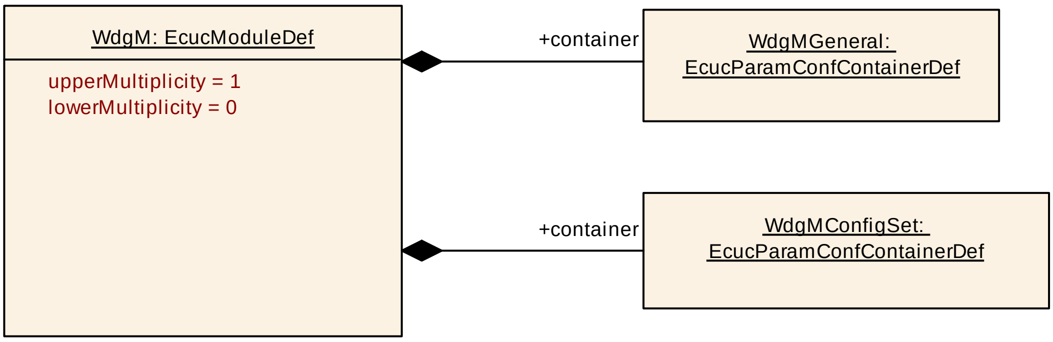

Figure 17: Configuration Module WdgM

10.2.3 WdgMGeneral

SWS Item	[ECUC_WdgM_00300]
Container Name	WdgMGeneral
Parent Container	WdgM
Description	Container defines all general configuration parameters of the Watchdog Manager.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00301]		
Parameter Name	WdgMDevErrorDetect		
Parent Container	WdgMGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00363]		
Parameter Name	WdgMEnableTimeoutDetection		
Parent Container	WdgMGeneral		
Description	This parameter enables the timeout detection part of the Deadline Supervision (needed to detect deadline supervision violation when end checkpoint is never reached). true : Timeout detection is enabled. false : Timeout detection is disabled. Note: By default this option is disabled for backward compatibility reasons.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00339]		
Parameter Name	WdgMImmediateReset		
Parent Container	WdgMGeneral		
Description	This parameter enables/disablse the immediate reset feature in case of alive-supervision failure. true: Immediate reset is enabled false: Immediate reset is disabled		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00340]		
Parameter Name	WdgMOffModeEnabled		
Parent Container	WdgMGeneral		
Description	This parameter enables/disables the selection of the "OffMode" of the watchdog driver. true: "OffMode" selection is allowed false: "OffMode" selection is disallowed		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00365]
Parameter Name	WdgMSwClusterSupport

Parent Container	WdgMGeneral		
Description	This parameter selects the support for SW Architecture with Software Clusters. If the parameter is not set the default behavior DISABLE_SW_CLUSTER_SUPPORT applies. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	DISABLE_SW_CLUSTER_SUPPORT	Additional functionality to support the Watchdog Manager integration into a SW Architecture with Software Clusters is disabled. **Tags:** atp.Status=draft	
	ENABLE_SW_CLUSTER_SUPPORT	Additional functionality to support the Watchdog Manger integration into a SW Architecture with Software Clusters is enabled. **Tags:** atp.Status=draft	
Default value	DISABLE_SW_CLUSTER_SUPPORT		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00302]		
Parameter Name	WdgMVersionInfoApi		
Parent Container	WdgMGeneral		
Description	Preprocessor switch to enable/disable the existence of the API WdgM_GetVersionInfo. Shall be used to remove unneeded code segments. true: API is enabled false: API is disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value	**Pre-compile time**	X	All Variants

Configuration Class	**Link time**	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WdgMBase-Socket	0..*	This container configures how many EcucPartitions specific infrastructure links are required for the WdgM instances in Application Software Clusters provided by the Host Software Cluster. Such infrastructure links serve for: the initialization of Application Software Cluster WdgM instances by Host WdgM instance the transmission of supervision results from Application Software Cluster WdgM instances to Host WdgM instance any other implementation specific purpose which is need for the interaction of Application Software Cluster WdgM instances and Host WdgM instance If the infrastructure connection is specific to one or several Ecuc Partition(s) the WdgMSocketEcucPartitionRef(s) denotes the applicable EcucPartition. **Tags:** atp.Status=draft
WdgMMain-Function	0..*	Reference to the WdgMInstanceMainFunction which this Supervised Entity belongs to. Relevant to Alive Supervision and Deadline Supervision **Tags:** atp.Status=draft
WdgM-Supervised-Entity	0..65535	This container collects all common (mode-independent) parameters of a Supervised Entity to be supervised by the Watchdog Manager.
WdgM-Watchdog	0..255	This container collects all common (mode-independent) parameters of a Watchdog to be triggered by the Watchdog Manager.

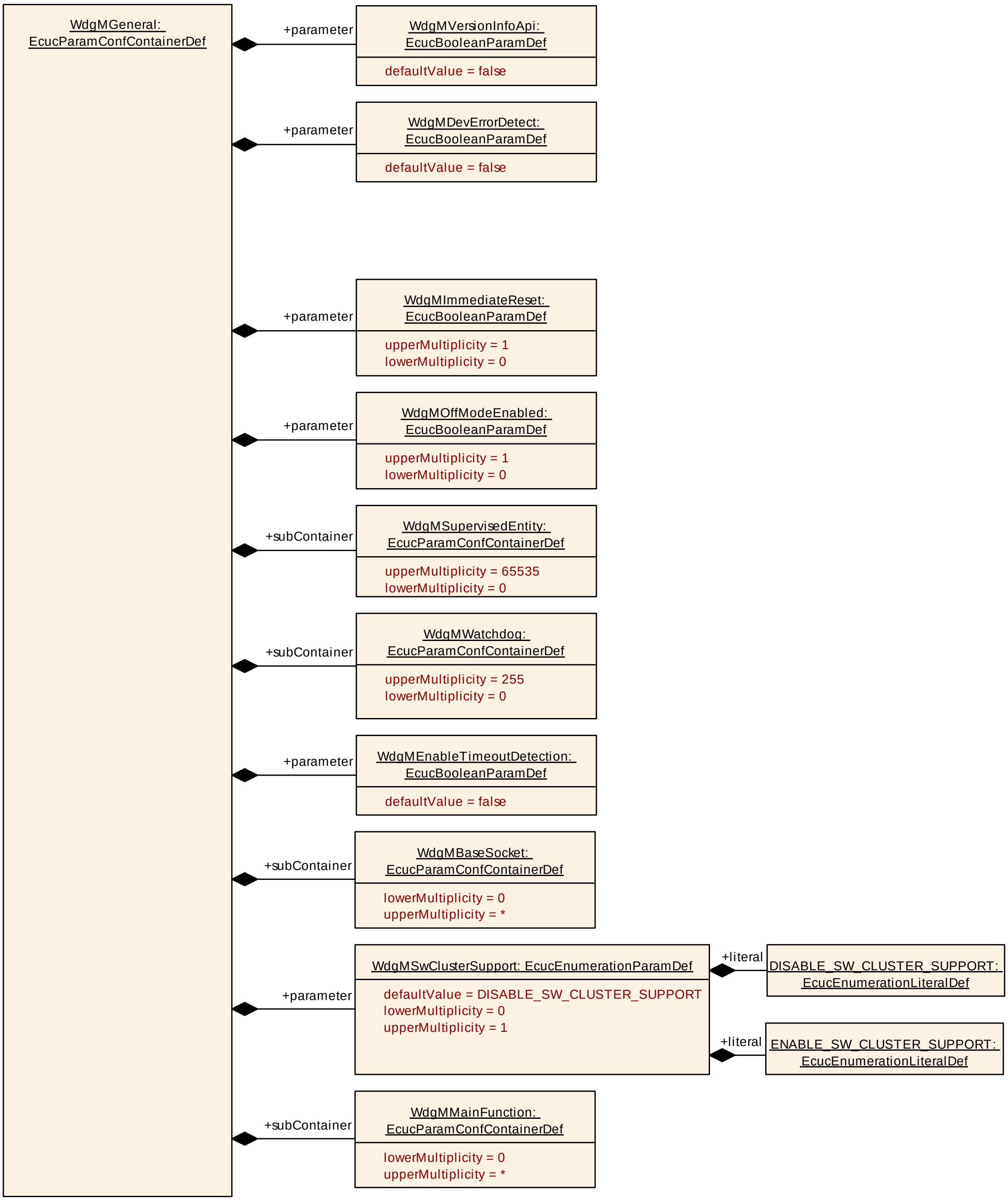
WdgMGeneral: EcucParamConfContainerDef
+parameter
WdgMVersionInfoApi: EcucBooleanParamDef
defaultValue = false
+parameter
WdgMDevErrorDetect: EcucBooleanParamDef
defaultValue = false
+parameter
WdgMImmediateReset: EcucBooleanParamDef
upperMultiplicity = 1
lowerMultiplicity = 0
+parameter
WdgMOffModeEnabled: EcucBooleanParamDef
upperMultiplicity = 1
lowerMultiplicity = 0
+subContainer
WdgMSupervisedEntity: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
WdgMWatchdog: EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 0
+parameter
WdgMEnableTimeoutDetection: EcucBooleanParamDef
defaultValue = false
+subContainer
WdgMBaseSocket: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
WdgMSwClusterSupport: EcucEnumerationParamDef
defaultValue = DISABLE_SW_CLUSTER_SUPPORT
lowerMultiplicity = 0
upperMultiplicity = 1
+literal
DISABLE_SW_CLUSTER_SUPPORT: EcucEnumerationLiteralDef
+literal
ENABLE_SW_CLUSTER_SUPPORT: EcucEnumerationLiteralDef
+subContainer
WdgMMainFunction: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *

Figure 18: Configuration Container WdgMGeneral

10.2.4 WdgMSupervisedEntity

SWS Item	[ECUC_WdgM_00303]
Container Name	WdgMSupervisedEntity

Parent Container	WdgMGeneral
Description	This container collects all common (mode-independent) parameters of a Supervised Entity to be supervised by the Watchdog Manager.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00304]		
Parameter Name	WdgMSupervisedEntityId		
Parent Container	WdgMSupervisedEntity		
Description	This parameter shall contain the unique identifier of the supervised entity.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00343]		
Parameter Name	WdgMInternalCheckpointInitialRef		
Parent Container	WdgMSupervisedEntity		
Description	This is the reference to the initial Checkpoint for this Supervised Entity.		
Multiplicity	0..65535		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00344]		
Parameter Name	WdgMInternallCheckpointFinalRef		
Parent Container	WdgMSupervisedEntity		
Description	This is the reference to the final Checkpoint(s) for this Supervised Entity.		
Multiplicity	0..65535		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00368]		
Parameter Name	WdgMMainFunctionRef		
Parent Container	WdgMSupervisedEntity		
Description	Reference to the WdgMInstanceMainFunction which this Supervised Entity belongs to. Relevant to Alive Supervision and Deadline Supervision **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Reference to WdgMMainFunction		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration	**Pre-compile time**	X	All Variants

Class	**Link time**	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00361]		
Parameter Name	WdgMOSCounter		
Parent Container	WdgMSupervisedEntity		
Description	OS counter used by Watchdog Manager to perform the deadline supervision of the Supervised Entity.		
Multiplicity	0..1		
Type	Reference to OsCounter		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WdgMCheckpoint	1..65535	This container collects all Checkpoints of this Supervised Entity. Each Supervised Entity has at least one Checkpoint.
WdgMInternal-Transition	0..65535	This container defines the graph of Internal Transitions within this Supervised Entity.

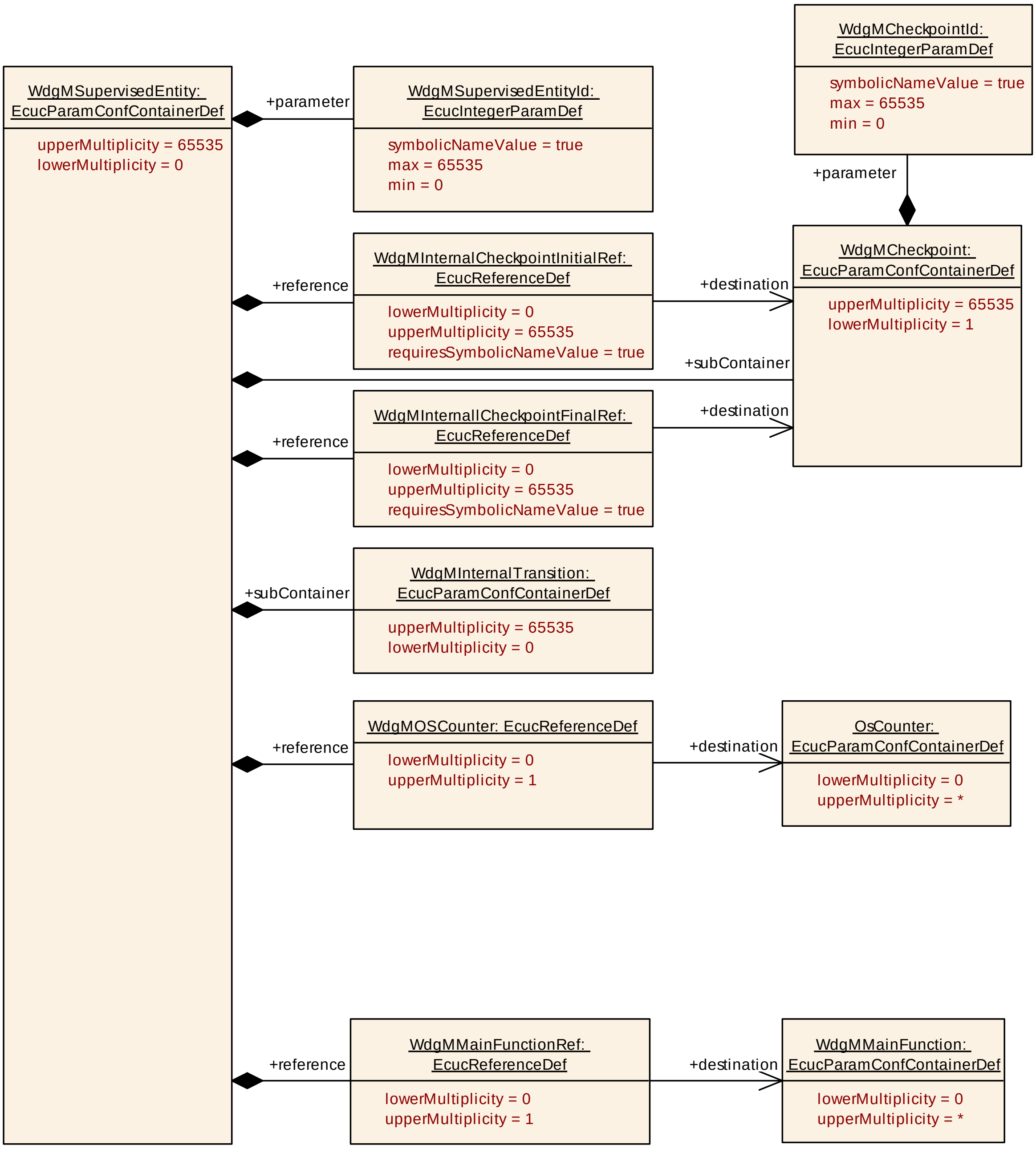
WdgMSupervisedEntity: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+parameter
WdgMSupervisedEntityId: EcucIntegerParamDef
symbolicNameValue = true
max = 65535
min = 0
WdgMCheckpointId: EcucIntegerParamDef
symbolicNameValue = true
max = 65535
min = 0
+parameter
+reference
WdgMInternalCheckpointInitialRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 65535
requiresSymbolicNameValue = true
+destination
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+subContainer
+reference
WdgMInternallCheckpointFinalRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 65535
requiresSymbolicNameValue = true
+destination
+subContainer
WdgMInternalTransition: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+reference
WdgMOSCounter: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
OsCounter: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+reference
WdgMMainFunctionRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
WdgMMainFunction: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *

Figure 19: Configuration Container WdgMSupervisedEntity

10.2.5 WdgMCheckpoint

SWS Item	[ECUC_WdgM_00305]
Container Name	WdgMCheckpoint
Parent Container	WdgMSupervisedEntity

Description	This container collects all Checkpoints of this Supervised Entity. Each Supervised Entity has at least one Checkpoint.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00306]		
Parameter Name	WdgMCheckpointId		
Parent Container	WdgMCheckpoint		
Description	This parameter shall contain the unique identifier of Checkpoint.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

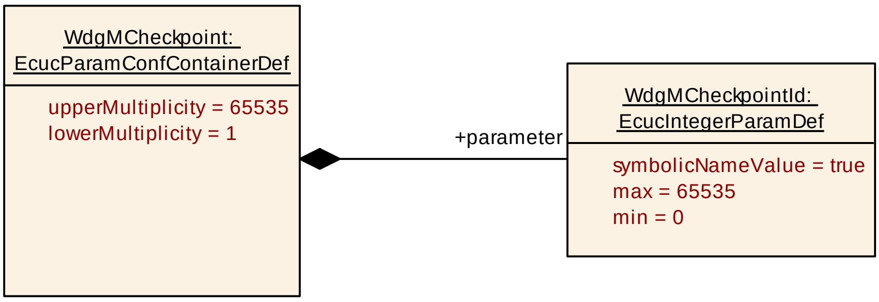
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+parameter
WdgMCheckpointId: EcucIntegerParamDef
symbolicNameValue = true
max = 65535
min = 0

Figure 20: Configuration Container WdgMCheckpoint

10.2.6 WdgMInternalTransition

SWS Item	[ECUC_WdgM_00345]

Container Name	WdgMInternalTransition
Parent Container	WdgMSupervisedEntity
Description	This container defines the graph of Internal Transitions within this Supervised Entity.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00351]		
Parameter Name	WdgMInternalTransitionDestRef		
Parent Container	WdgMInternalTransition		
Description	This is the reference to the destination Checkpoint of a Internal Transition within this Supervised Entity.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00350]		
Parameter Name	WdgMInternalTransitionSourceRef		
Parent Container	WdgMInternalTransition		
Description	This is the reference to the source Checkpoint of a Internal Transition within this Supervised Entity.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

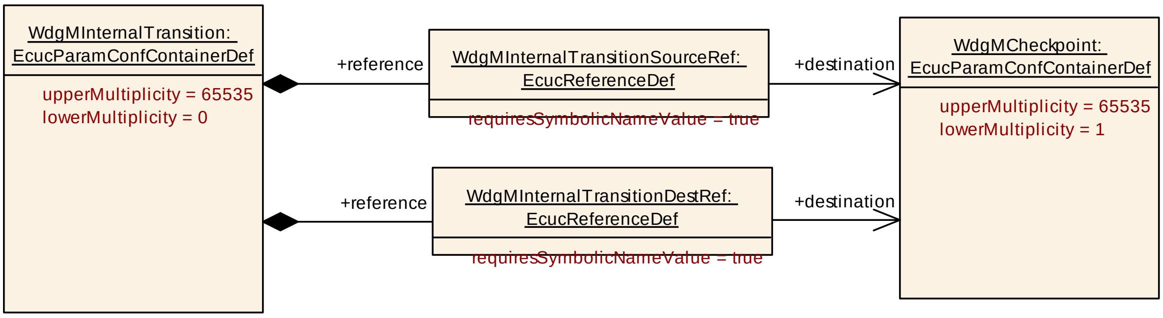
WdgMInternalTransition: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+reference
WdgMInternalTransitionSourceRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+reference
WdgMInternalTransitionDestRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination

Figure 21: Configuration Container WdgMInternalTransition

10.2.7 WdgMWatchdog

SWS Item	[ECUC_WdgM_00347]
Container Name	WdgMWatchdog
Parent Container	WdgMGeneral
Description	This container collects all common (mode-independent) parameters of a Watchdog to be triggered by the Watchdog Manager.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00348]		
Parameter Name	WdgMWatchdogName		
Parent Container	WdgMWatchdog		
Description	This parameter shall contain the name of the watchdog instance.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
Regular Expression	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	[ECUC_WdgM_00349]		
Parameter Name	WdgMWatchdogDeviceRef		
Parent Container	WdgMWatchdog		
Description	Reference to one device container of Watchdog Interface. In the referenced container WdgIfDevice, the parameter WdgIfDeviceIndex contains the Index parameter that WdgM has to use for WdgIf_SetTriggerCondition calls for that watchdog instance.		
Multiplicity	1		
Type	Symbolic name reference to WdgIfDevice		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

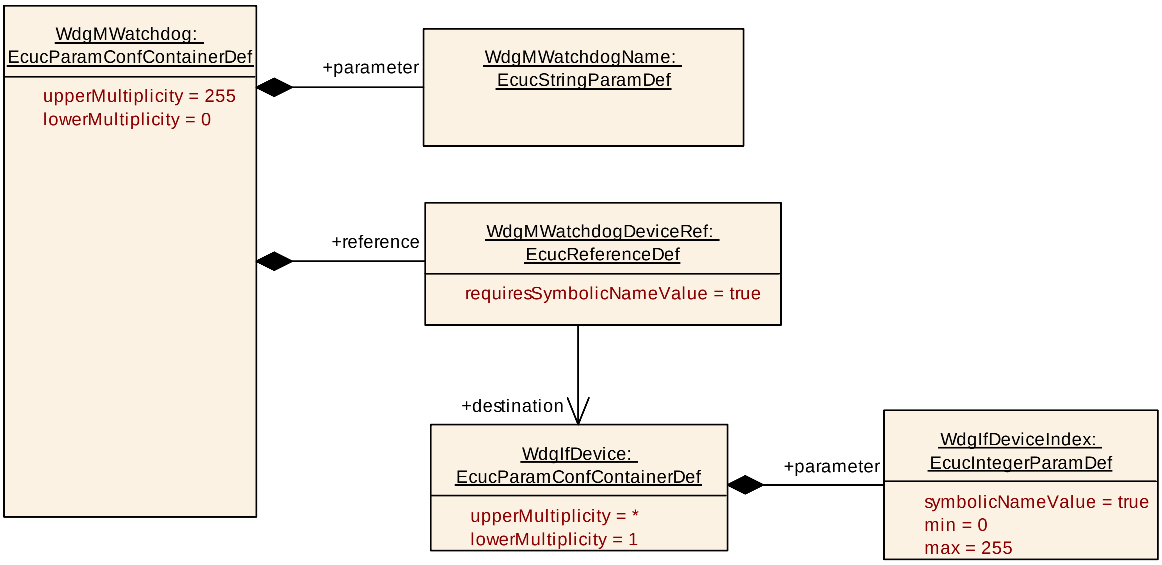
WdgMWatchdog: EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 0
+parameter
WdgMWatchdogName: EcucStringParamDef
+reference
WdgMWatchdogDeviceRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
WdgIfDevice: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+parameter
WdgIfDeviceIndex: EcucIntegerParamDef
symbolicNameValue = true
min = 0
max = 255

Figure 22: Configuration Container WdgMWatchdog

10.2.8 WdgMConfigSet

SWS Item	[ECUC_WdgM_00337]
Container Name	WdgMConfigSet
Parent Container	WdgM
Description	This container describes one of multiple configuration sets of WdgM.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00336]		
Parameter Name	WdgMInitialMode		
Parent Container	WdgMConfigSet		
Description	The mode that the Watchdog Manager is in after it has been initialized.		
Multiplicity	1		
Type	Symbolic name reference to WdgMMode		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WdgMDem-Event-Parameter-Refs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
WdgMMode	1..255	The container describes one of several modes of the Watchdog Manager.

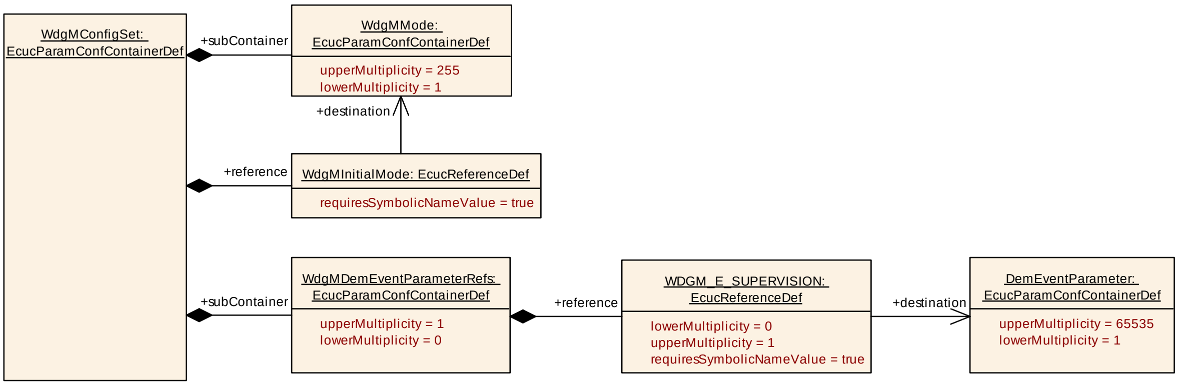
WdgMConfigSet: EcucParamConfContainerDef
+subContainer
WdgMMode: EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 1
+destination
+reference
WdgMInitialMode: EcucReferenceDef
requiresSymbolicNameValue = true
+subContainer
WdgMDemEventParameterRefs: EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
+reference
WDGM_E_SUPERVISION: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
DemEventParameter: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1

Figure 23: Configuration Container WdgMConfigSet

10.2.9 WdgMDemEventParameterRefs

SWS Item	[ECUC_WdgM_00353]
Container Name	WdgMDemEventParameterRefs
Parent Container	WdgMConfigSet
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00362]
Parameter Name	WDGM_E_SUPERVISION
Parent Container	WdgMDemEventParameterRefs
Description	Reference to the DemEventParameter which shall be issued when the error "Supervision has failed (Global Supervision Status has reached WDGM_GLOBAL_STATUS_STOPPED) and a watchdog reset will occur" has occurred.
Multiplicity	0..1
Type	Symbolic name reference to DemEventParameter
Post-Build Variant Multiplicity	true
Post-Build Variant Value	true

Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.10 WdgMMode

SWS Item	[ECUC_WdgM_00335]
Container Name	WdgMMode
Parent Container	WdgMConfigSet
Description	The container describes one of several modes of the Watchdog Manager.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00329]		
Parameter Name	WdgMExpiredSupervisionCycleTol		
Parent Container	WdgMMode		
Description	This parameter shall be used to define a value that fixes the amount of expired supervision cycles for how long the blocking of watchdog triggering shall be postponed, AFTER THE GLOBAL SUPERVISION STATUS HAS REACHED THE STATE EXPIRED.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	

Class	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	[ECUC_WdgM_00307]		
Parameter Name	WdgMModeId		
Parent Container	WdgMMode		
Description	This parameter fixes the identifier for the mode. This identifier is for instance passed as a parameter to the WdgM_SetMode service.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WdgMAlive-Supervision	0..65535	This container collects all configuration parameters of Alive-Supervision of one Checkpoint. Note that each Checkpoint may have different parameters. For example, it may have different min and max margin.
WdgMDeadline-Supervision	0..65535	This container collects all configuration parameters for Deadline Supervision for a Supervised Entity.
WdgMExternal-Logical-Supervision	0..65535	This container collects all configuration parameters for Logical Supervision for one external graph.
WdgMLocal-StatusParams	0..65535	This container collects all configuration parameters for the Local Status of a Supervised Entity.
WdgMMain-FunctionMode-Props	0..*	This container provides configuration values for a WdgMMain Function which apply in a specific WdgMMode. **Tags:** atp.Status=draft
WdgMTrigger	0..255	This container collects all configuration parameters for the

		triggering of hardware watchdogs.

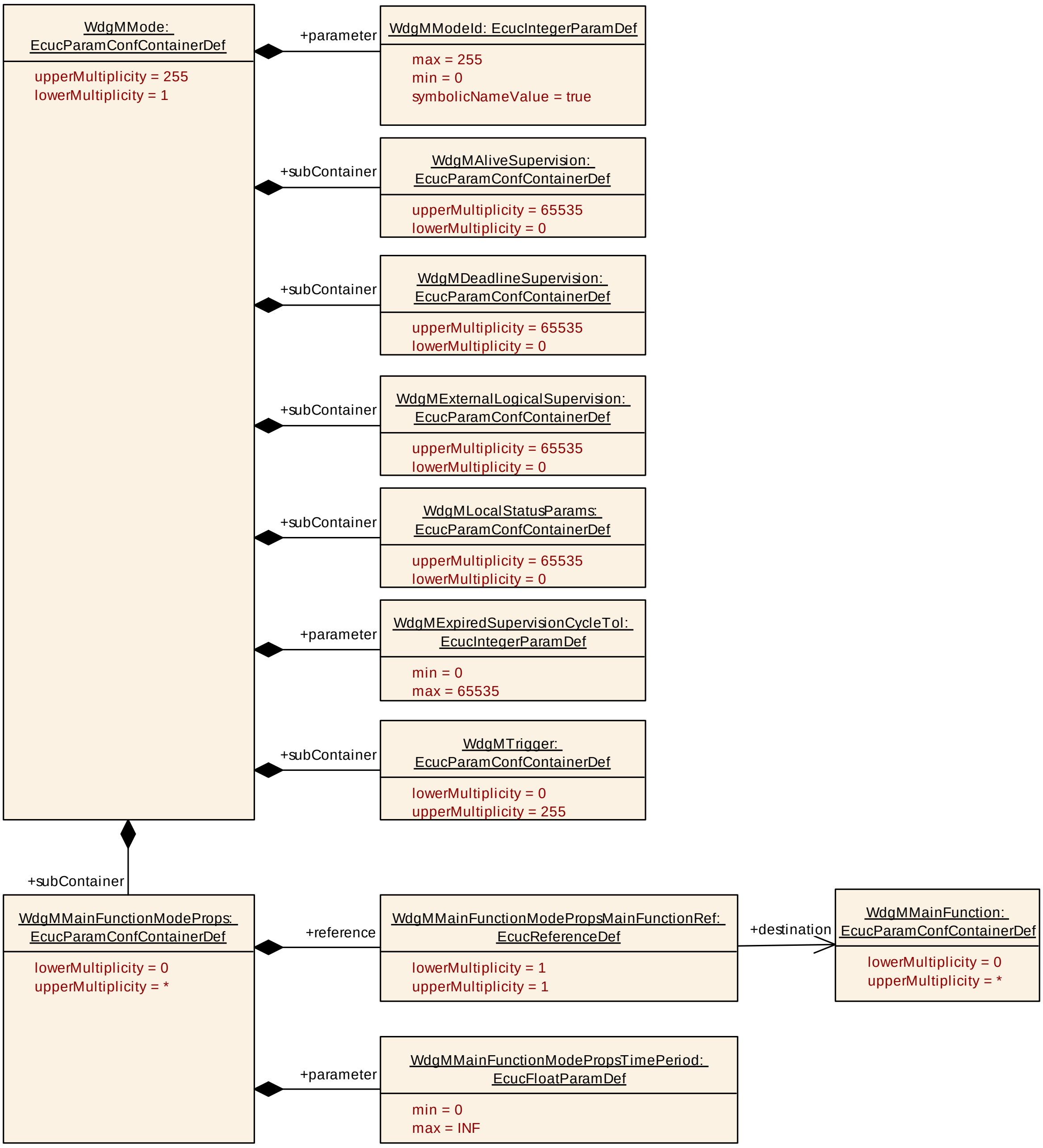
WdgMMode: EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 1
+parameter
WdgMModeId: EcucIntegerParamDef
max = 255
min = 0
symbolicNameValue = true
+subContainer
WdgMAliveSupervision: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
WdgMDeadlineSupervision: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
WdgMExternalLogicalSupervision: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
WdgMLocalStatusParams: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+parameter
WdgMExpiredSupervisionCycleTol: EcucIntegerParamDef
min = 0
max = 65535
+subContainer
WdgMTrigger: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 255
+subContainer
WdgMMainFunctionModeProps: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+reference
WdgMMainFunctionModePropsMainFunctionRef: EcucReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
+destination
WdgMMainFunction: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
WdgMMainFunctionModePropsTimePeriod: EcucFloatParamDef
min = 0
max = INF

Figure 24: Configuration Container WdgMMode

10.2.11 WdgMAliveSupervision

SWS Item	[ECUC_WdgM_00308]
Container Name	WdgMAliveSupervision

Parent Container	WdgMMode
Description	This container collects all configuration parameters of Alive-Supervision of one Checkpoint. Note that each Checkpoint may have different parameters. For example, it may have different min and max margin.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00311]		
Parameter Name	WdgMExpectedAliveIndications		
Parent Container	WdgMAliveSupervision		
Description	This parameter contains the amount of expected alive indications of the Checkpoint within the referenced amount of defined supervision cycles according to corresponding SE.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00313]	
Parameter Name	WdgMMaxMargin	
Parent Container	WdgMAliveSupervision	
Description	This parameter contains the amount of alive indications of the Checkpoint that are acceptable to be additional to the expected alive indications within the corresponding supervision reference cycle.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	--	
Post-Build	true	

Variant Value			
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00312]		
Parameter Name	WdgMMinMargin		
Parent Container	WdgMAliveSupervision		
Description	This parameter contains the amount of alive indications of the Checkpoint that are acceptable to be missed from the expected alive indications within the corresponding supervision reference cycle.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00310]	
Parameter Name	WdgMSupervisionReferenceCycle	
Parent Container	WdgMAliveSupervision	
Description	This parameter shall contain the amount of supervision cycles to be used as reference by the alive-supervision mechanism to perform the checkup with counted alive indications according to corresponding SE.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 65535	
Default value	--	

Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00309]		
Parameter Name	WdgMAliveSupervisionCheckpointRef		
Parent Container	WdgMAliveSupervision		
Description	Reference to Checkpoint within a Supervised Entity that shall be supervised.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

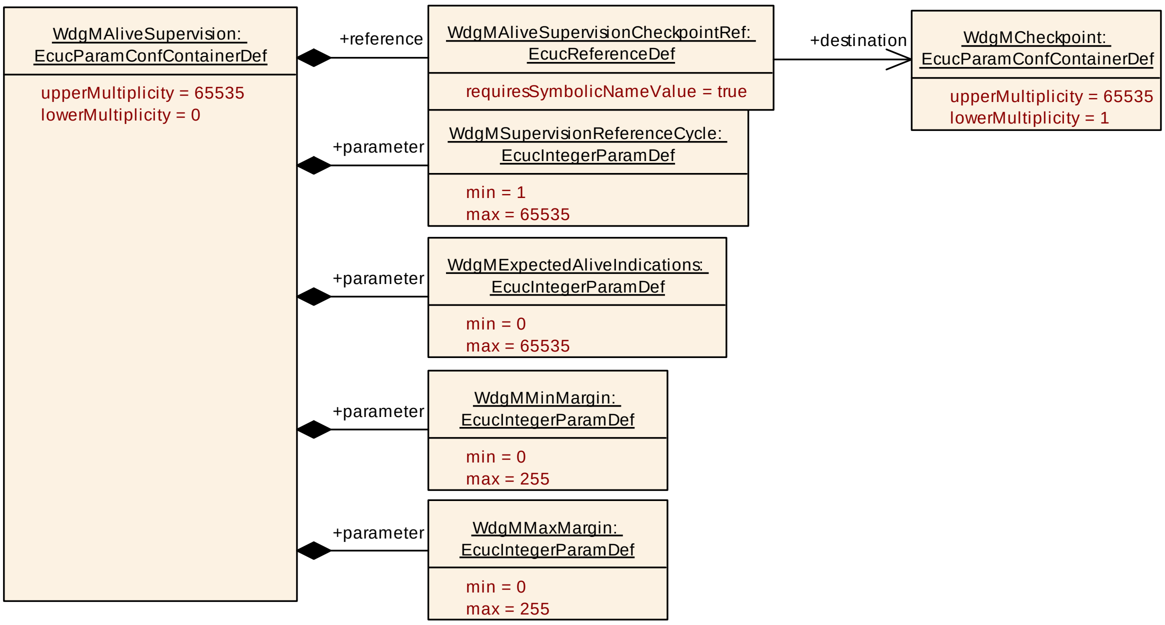

Figure 25: Configuration Container WdgMAliveSupervision

10.2.12 WdgMDeadlineSupervision

SWS Item	[ECUC_WdgM_00314]
Container Name	WdgMDeadlineSupervision
Parent Container	WdgMMode
Description	This container collects all configuration parameters for Deadline Supervision for a Supervised Entity.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00318]	
Parameter Name	WdgMDeadlineMax	
Parent Container	WdgMDeadlineSupervision	
Description	This parameter contains the longest time span after which the deadline is considered to be met. Unit: [s]	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	

Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00317]		
Parameter Name	WdgMDeadlineMin		
Parent Container	WdgMDeadlineSupervision		
Description	This parameter contains the shortest time span after which the deadline is considered to be met. Unit: [s]		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00315]		
Parameter Name	WdgMDeadlineStartRef		
Parent Container	WdgMDeadlineSupervision		
Description	This is the reference to the start Checkpoint for Deadline Supervision.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00316]		
Parameter Name	WdgMDeadlineStopRef		
Parent Container	WdgMDeadlineSupervision		
Description	This is the reference to the stop Checkpoint for Deadline Supervision.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

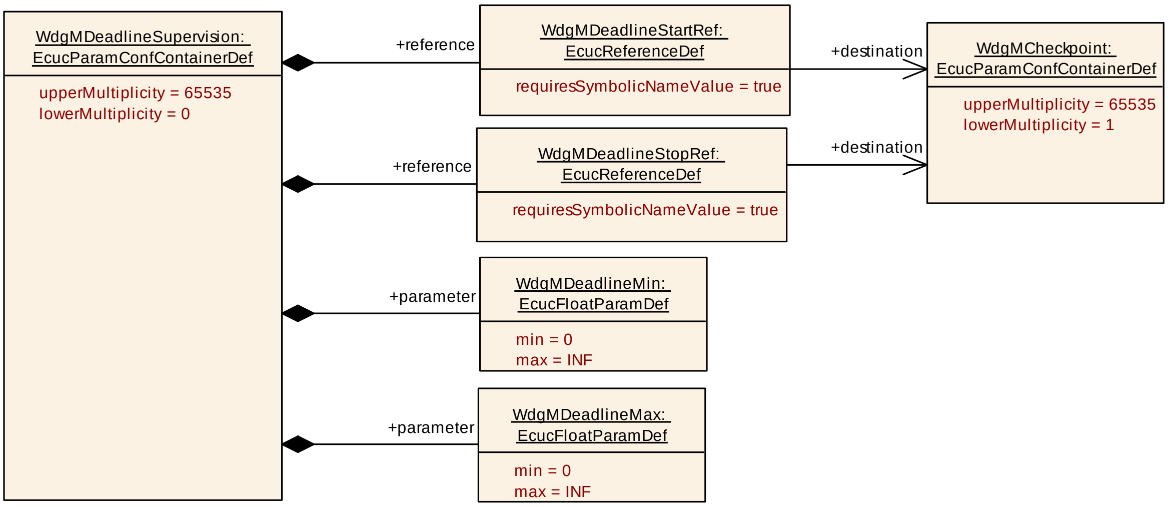
WdgMDeadlineSupervision: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+reference
WdgMDeadlineStartRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+reference
WdgMDeadlineStopRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
+parameter
WdgMDeadlineMin: EcucFloatParamDef
min = 0
max = INF
+parameter
WdgMDeadlineMax: EcucFloatParamDef
min = 0
max = INF

Figure 26: Configuration Container WdgMDeadlineSupervision

10.2.13 WdgMExternalLogicalSupervision

SWS Item	[ECUC_WdgM_00319]
Container	WdgMExternalLogicalSupervision

Name	
Parent Container	WdgMMode
Description	This container collects all configuration parameters for Logical Supervision for one external graph.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00324]		
Parameter Name	WdgMExternalCheckpointFinalRef		
Parent Container	WdgMExternalLogicalSupervision		
Description	This is the reference to the final Checkpoint(s) for this External Graph which can end with a WdgMCheckpoint or in case of cross cluster transitions with a Wdg MTransitionProxy. Both WdgMCheckpoint(s) and WdgMTransitionProxy(s) could be mixed inside the same WdgMExternalLogicalSupervision.		
Multiplicity	1..65535		
Type	Choice symbolic name reference to [WdgMCheckpoint, WdgMTransitionProxy]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00323]
Parameter Name	WdgMExternalCheckpointInitialRef
Parent Container	WdgMExternalLogicalSupervision
Description	This is the reference to the initial Checkpoint(s) for this External Graph which can start with a WdgMCheckpoint or in case of cross cluster transitions with a Wdg MTransitionProxy. Both WdgMCheckpoint(s) and WdgMTransitionProxy(s) could

	be mixed inside the same WdgMExternalLogicalSupervision.		
Multiplicity	1..65535		
Type	Choice symbolic name reference to [WdgMCheckpoint, WdgMTransitionProxy]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WdgMCross-Cluster-Transition	0..65535	This container configures a cross cluster transition. A WdgMCrossClusterTransition can be configured • from a WdgMCheckpoint to a WdgMTransitionProxy • from a WdgMTransitionProxy to a WdgMCheckpoint • from a WdgMTransitionProxy to another WdgMTransitionProxy (in Host Software Cluster only) • from a WdgMTransitionProxy to the identical WdgMTransition Proxy (in Application Software Cluster only for the case that no WdgMCheckpoint has to be reached in the Application Software Cluster) • from a WdgMCheckpoint to a WdgMCheckpoint (in case the cross cluster transition graph is entirely described with Wdg MCrossClusterTransition containers) **Tags:** atp.Status=draft
WdgM-External-Transition	0..65535	This container collects the Checkpoints for an External Transition across Supervised Entities.
WdgM-Transition-Proxy	0..65535	The WdgMTransitionProxy defines a proxy for a transition between the Host Software Cluster and an Application Software Cluster and vice versa. From the Host Software Cluster perspective a Cross Cluster Transition graph leaves the host after the transition which has the Wdg

		MTransitionProxy as a destination or initial reference and returns in this WdgMTransitionProxy after the configured transitions are occurred in the related Application Software Cluster. Afterwards the transition in the host are expected which are referencing the WdgMTransitionProxy by a source or final reference. **Tags:** atp.Status=draft

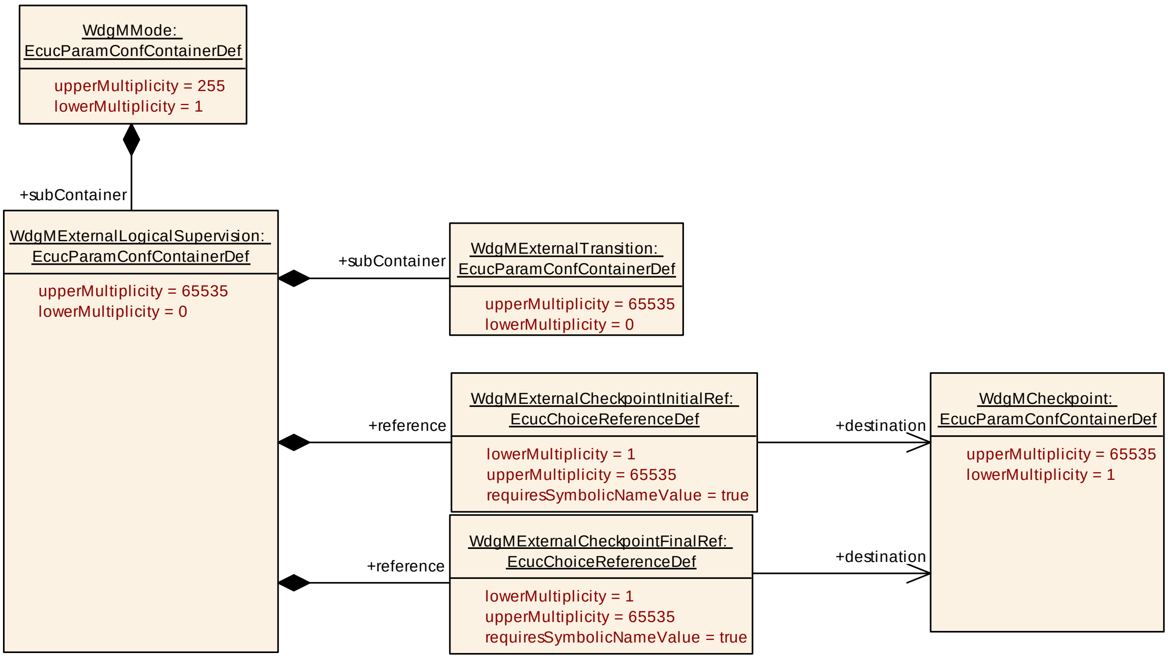
WdgMMode: EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 1
+subContainer
WdgMExternalLogicalSupervision: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
WdgMExternalTransition: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+reference
WdgMExternalCheckpointInitialRef: EcucChoiceReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 65535
requiresSymbolicNameValue = true
+destination
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+reference
WdgMExternalCheckpointFinalRef: EcucChoiceReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 65535
requiresSymbolicNameValue = true
+destination

Figure 27: Configuration Container WdgMExternalLogicalSupervision

10.2.14 WdgMExternalTransition

SWS Item	[ECUC_WdgM_00320]
Container Name	WdgMExternalTransition
Parent Container	WdgMExternalLogicalSupervision
Description	This container collects the Checkpoints for an External Transition across Supervised Entities.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00322]
Parameter Name	WdgMExternalTransitionDestRef
Parent Container	WdgMExternalTransition

Description	This is the reference to the destination Checkpoint of an External Transition.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00321]		
Parameter Name	WdgMExternalTransitionSourceRef		
Parent Container	WdgMExternalTransition		
Description	This is the reference to the source Checkpoint of an External Transition.		
Multiplicity	1		
Type	Symbolic name reference to WdgMCheckpoint		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

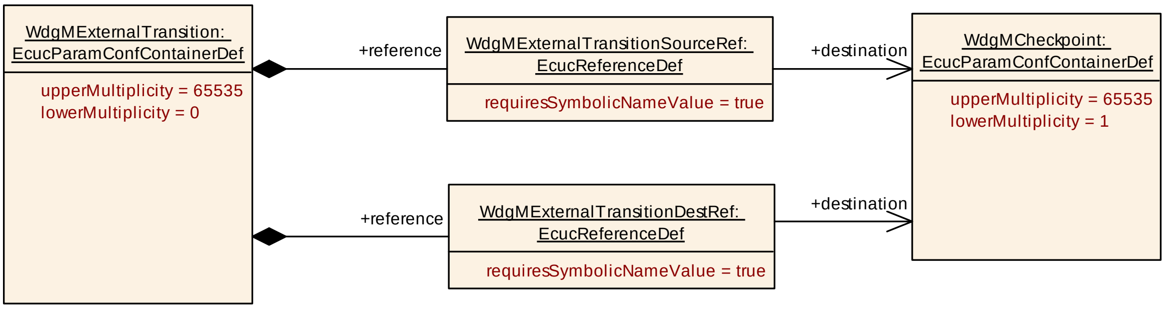
WdgMExternalTransition: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+reference
WdgMExternalTransitionSourceRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+reference
WdgMExternalTransitionDestRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination

Figure 28: Configuration Container WdgMExternalTransition

10.2.15 WdgMTrigger

SWS Item	[ECUC_WdgM_00331]
Container Name	WdgMTrigger
Parent Container	WdgMMode
Description	This container collects all configuration parameters for the triggering of hardware watchdogs.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00333]		
Parameter Name	WdgMTriggerConditionValue		
Parent Container	WdgMTrigger		
Description	This parameter shall contain the value that is passed to WdgIf_SetTrigger Condition for this watchdog.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00332]
Parameter Name	WdgMWatchdogMode
Parent Container	WdgMTrigger
Description	This parameter contains the watchdog mode that shall be used for the referenced watchdog in this Watchdog Manager mode. Implementation Type: WdgIf_ModeType
Multiplicity	1

Type	EcucEnumerationParamDef		
Range	WDGIF_FAST_MODE	--	
	WDGIF_OFF_MODE	--	
	WDGIF_SLOW_MODE	--	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00334]		
Parameter Name	WdgMTriggerWatchdogRef		
Parent Container	WdgMTrigger		
Description	This parameter is a reference to the configured watchdog.		
Multiplicity	1		
Type	Reference to WdgMWatchdog		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

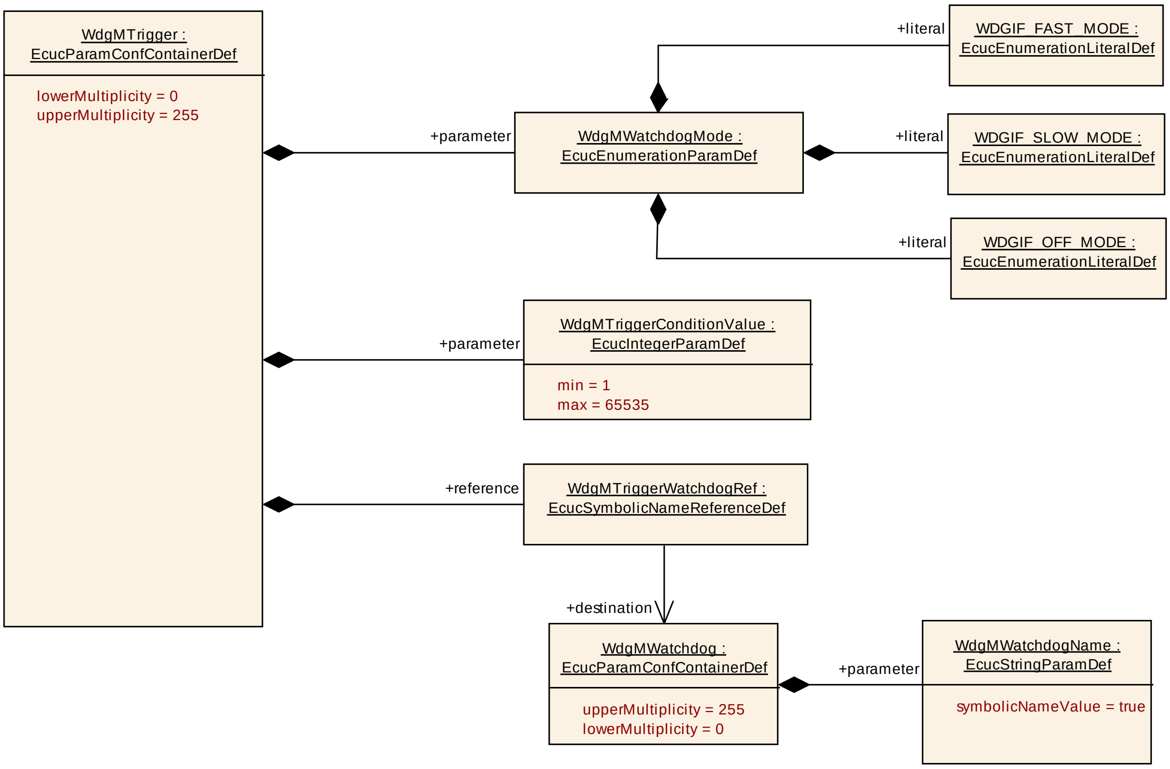
WdgMTrigger : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 255
+parameter
WdgMWatchdogMode : EcucEnumerationParamDef
+literal
WDGIF_FAST_MODE : EcucEnumerationLiteralDef
+literal
WDGIF_SLOW_MODE : EcucEnumerationLiteralDef
+literal
WDGIF_OFF_MODE : EcucEnumerationLiteralDef
+parameter
WdgMTriggerConditionValue : EcucIntegerParamDef
min = 1
max = 65535
+reference
WdgMTriggerWatchdogRef : EcucSymbolicNameReferenceDef
+destination
WdgMWatchdog : EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 0
+parameter
WdgMWatchdogName : EcucStringParamDef
symbolicNameValue = true

Figure 29: Configuration Container WdgMTrigger

10.2.16 WdgMLocalStatusParams

SWS Item	[ECUC_WdgM_00325]
Container Name	WdgMLocalStatusParams
Parent Container	WdgMMode
Description	This container collects all configuration parameters for the Local Status of a Supervised Entity.
Configuration Parameters	

SWS Item	[ECUC_WdgM_00327]	
Parameter Name	WdgMFailedAliveSupervisionRefCycleTol	
Parent Container	WdgMLocalStatusParams	
Description	This parameter shall contain the acceptable amount of reference cycles with incorrect/failed alive supervisions for this Supervised Entity.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	

Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00326]		
Parameter Name	WdgMLocalStatusSupervisedEntityRef		
Parent Container	WdgMLocalStatusParams		
Description	This is the reference to the Supervised Entity for which the Local Status parameters are specified.		
Multiplicity	1		
Type	Symbolic name reference to WdgMSupervisedEntity		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

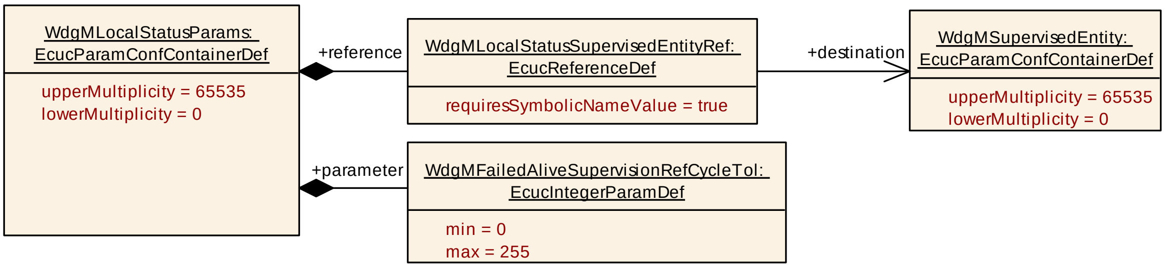
WdgMLocalStatusParams: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+reference
WdgMLocalStatusSupervisedEntityRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
WdgMSupervisedEntity: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+parameter
WdgMFailedAliveSupervisionRefCycleTol: EcucIntegerParamDef
min = 0
max = 255

Figure 30: Configuration Container WdgMLocalStatusParams

10.2.17 WdgMMainFunction

SWS Item	[ECUC_WdgM_00373]		
Container Name	WdgMMainFunction		
Parent Container	WdgMGeneral		
Description	Reference to the WdgMInstanceMainFunction which this Supervised Entity belongs to. Relevant to Alive Supervision and Deadline Supervision **Tags:** atp.Status=draft		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	[ECUC_WdgM_00369]		
Parameter Name	WdgMMainFunctionPartitionRef		
Parent Container	WdgMMainFunction		
Description	Reference to EcucPartition, where the according WdgM_MainFunction instance is assigned to. For the software architecture with single partition, this reference is unnecessary. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

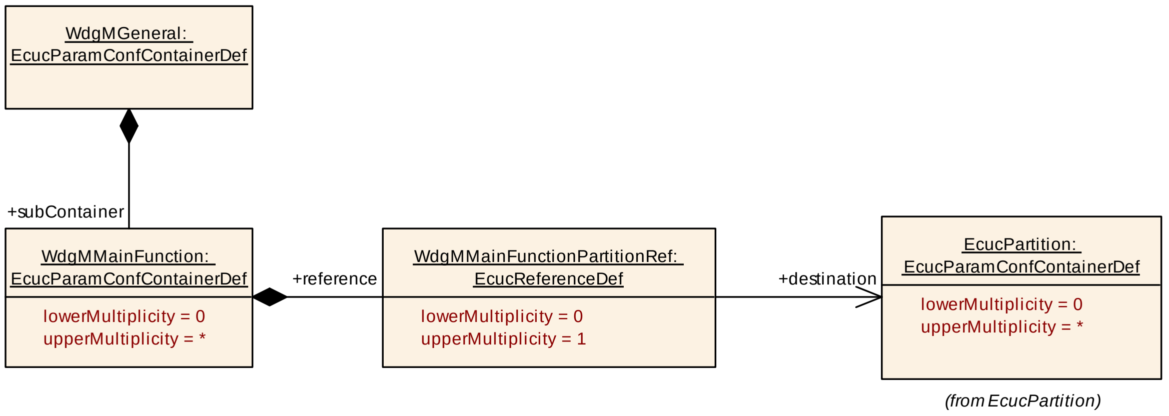

Figure 31: Configuration Container WdgMMainFunction

10.2.18 WdgMMainFunctionModeProps

SWS Item	[ECUC_WdgM_00372]		
Container Name	WdgMMainFunctionModeProps		
Parent Container	WdgMMode		
Description	This container provides configuration values for a WdgMMainFunction which apply in a specific WdgMMode. **Tags:** atp.Status=draft		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	[ECUC_WdgM_00370]
Parameter Name	WdgMMainFunctionModePropsTimePeriod
Parent Container	WdgMMainFunctionModeProps
Description	The period between successive calls to according instance of WdgM_MainFunction in seconds. This parameter may be used by the WdgM generator to transform the values of the WdgMModes and/or WdhMSupervisedEntities timing configuration parameters of the WdgM module to internal implementation specific counter or tick values. The WdgM module's internal timing handling is implementation specific. The WdgM module (generator) may rely on the fact that Wdg_MainFunction is scheduled according to the value configured here. Unit: [s] **Tags:** atp.Status=draft
Multiplicity	1

Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00371]		
Parameter Name	WdgMMainFunctionModePropsMainFunctionRef		
Parent Container	WdgMMainFunctionModeProps		
Description	Reference to the WdgMMainFunction for which the WdgMMainFunction ModeProps apply. **Tags:** atp.Status=draft		
Multiplicity	1		
Type	Reference to WdgMMainFunction		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.19 WdgMCrossClusterTransition

SWS Item	[ECUC_WdgM_00376]
Container Name	WdgMCrossClusterTransition
Parent	WdgMExternalLogicalSupervision

Container	
Description	This container configures a cross cluster transition. A WdgMCrossClusterTransition can be configured • from a WdgMCheckpoint to a WdgMTransitionProxy • from a WdgMTransitionProxy to a WdgMCheckpoint • from a WdgMTransitionProxy to another WdgMTransitionProxy (in Host Software Cluster only) • from a WdgMTransitionProxy to the identical WdgMTransitionProxy (in Application Software Cluster only for the case that no WdgMCheckpoint has to be reached in the Application Software Cluster) • from a WdgMCheckpoint to a WdgMCheckpoint (in case the cross cluster transition graph is entirely described with WdgMCrossClusterTransition containers) **Tags:** atp.Status=draft
Configuration Parameters	

SWS Item	[ECUC_WdgM_00375]		
Parameter Name	WdgMCrossClusterTransitionDestRef		
Parent Container	WdgMCrossClusterTransition		
Description	This is the reference to the destination of a cross cluster transition. **Tags:** atp.Status=draft		
Multiplicity	1		
Type	Choice symbolic name reference to [WdgMCheckpoint, WdgM-TransitionProxy]		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_WdgM_00374]
Parameter Name	WdgMCrossClusterTransitionSourceRef
Parent Container	WdgMCrossClusterTransition
Description	This is the reference to the source of a cross cluster transition. **Tags:** atp.Status=draft
Multiplicity	1
Type	Choice symbolic name reference to [WdgMCheckpoint, WdgM-TransitionProxy]

Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

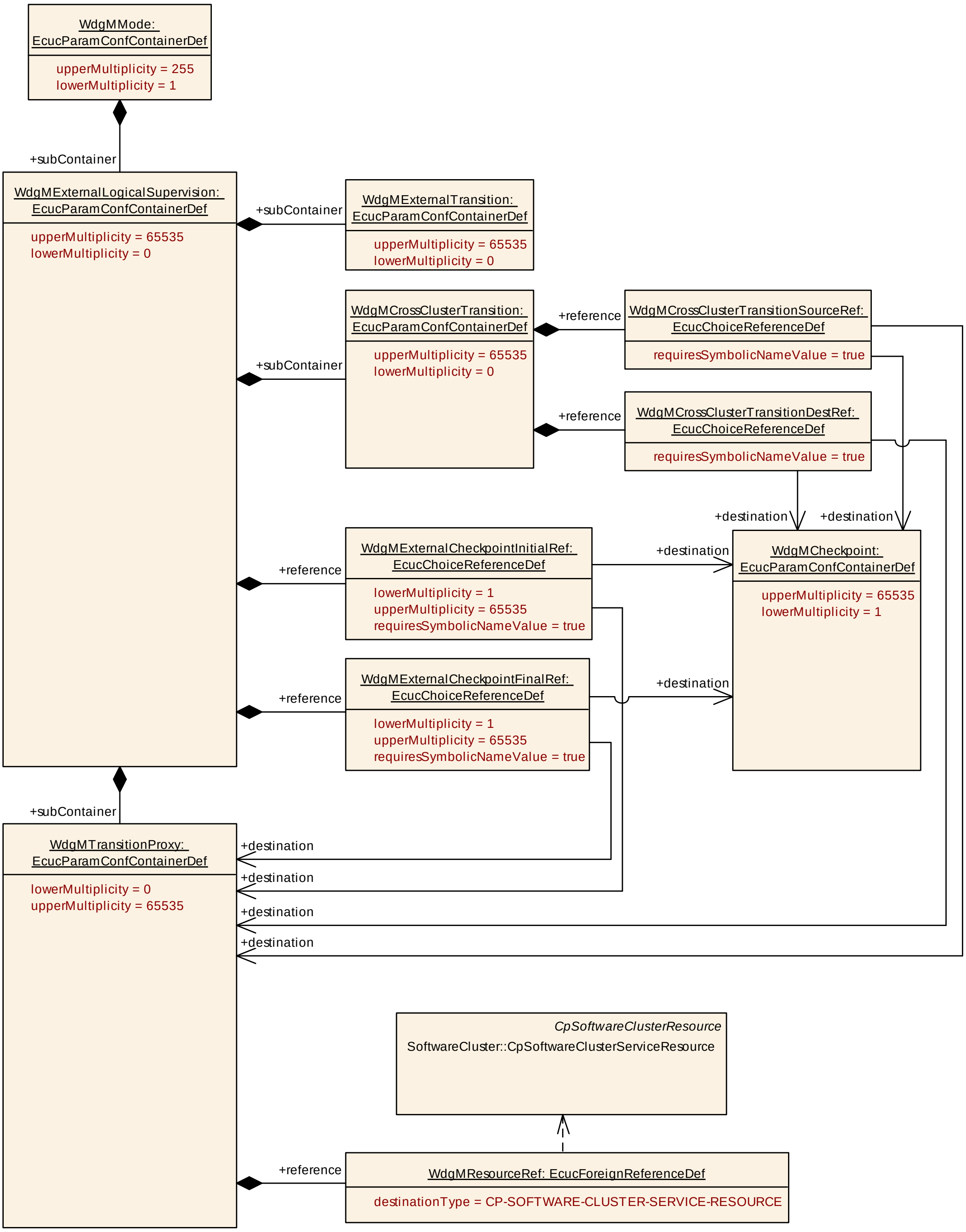
WdgMMode: EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 1
+subContainer
WdgMExternalLogicalSupervision: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
WdgMExternalTransition: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
WdgMCrossClusterTransition: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 0
+subContainer
+reference
WdgMCrossClusterTransitionSourceRef: EcucChoiceReferenceDef
requiresSymbolicNameValue = true
+reference
WdgMCrossClusterTransitionDestRef: EcucChoiceReferenceDef
requiresSymbolicNameValue = true
+destination
+destination
WdgMExternalCheckpointInitialRef: EcucChoiceReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 65535
requiresSymbolicNameValue = true
+reference
+destination
WdgMCheckpoint: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
WdgMExternalCheckpointFinalRef: EcucChoiceReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 65535
requiresSymbolicNameValue = true
+reference
+destination
+subContainer
WdgMTransitionProxy: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 65535
+destination
+destination
+destination
+destination
CpSoftwareClusterResource
SoftwareCluster::CpSoftwareClusterServiceResource
+reference
WdgMResourceRef: EcucForeignReferenceDef
destinationType = CP-SOFTWARE-CLUSTER-SERVICE-RESOURCE

Figure 32: Configuration Container WdgMCrossClusterTransition (for Clustered Software Architecture)

10.2.20 WdgMTransitionProxy

SWS Item	[ECUC_WdgM_00364]
Container	WdgMTransitionProxy

Name	
Parent Container	WdgMExternalLogicalSupervision
Description	The WdgMTransitionProxy defines a proxy for a transition between the Host Software Cluster and an Application Software Cluster and vice versa. From the Host Software Cluster perspective a Cross Cluster Transition graph leaves the host after the transition which has the WdgMTransitionProxy as a destination or initial reference and returns in this WdgMTransitionProxy after the configured transitions are occurred in the related Application Software Cluster. Afterwards the transition in the host are expected which are referencing the WdgMTransitionProxy by a source or final reference. **Tags:** atp.Status=draft
Configuration Parameters	

SWS Item	[ECUC_WdgM_00367]
Parameter Name	WdgMResourceRef
Parent Container	WdgMTransitionProxy
Description	Reference to the CpSoftwareClusterServiceResource. **Tags:** atp.Status=draft
Multiplicity	1
Type	Foreign reference to CP-SOFTWARE-CLUSTER-SERVICE-RESOURCE
Scope / Dependency	scope: ECU

No Included Containers

10.2.21 WdgMBaseSocket

SWS Item	[ECUC_WdgM_00377]
Container Name	WdgMBaseSocket
Parent Container	WdgMGeneral
Description	This container configures how many EcucPartitions specific infrastructure links are required for the WdgM instances in Application Software Clusters provided by the Host Software Cluster. Such infrastructure links serve for: the initialization of Application Software Cluster WdgM instances by Host WdgM instance the transmission of supervision results from Application Software Cluster WdgM instances to Host WdgM instance any other implementation specific purpose which is need for the interaction of Application Software Cluster WdgM instances and Host WdgM instance If the infrastructure connection is specific to one or several EcucPartition(s) the WdgMSocketEcucPartitionRef(s) denotes the applicable EcucPartition. **Tags:** atp.Status=draft

Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	[ECUC_WdgM_00378]
Parameter Name	WdgMResourceRef
Parent Container	WdgMBaseSocket
Description	Reference to the CpSoftwareClusterServiceResource. **Tags:** atp.Status=draft
Multiplicity	1
Type	Foreign reference to CP-SOFTWARE-CLUSTER-SERVICE-RESOURCE
Scope / Dependency	scope: ECU

SWS Item	[ECUC_WdgM_00366]		
Parameter Name	WdgMSocketEcucPartitionRef		
Parent Container	WdgMBaseSocket		
Description	Reference to the EcucPartition. **Tags:** atp.Status=draft		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

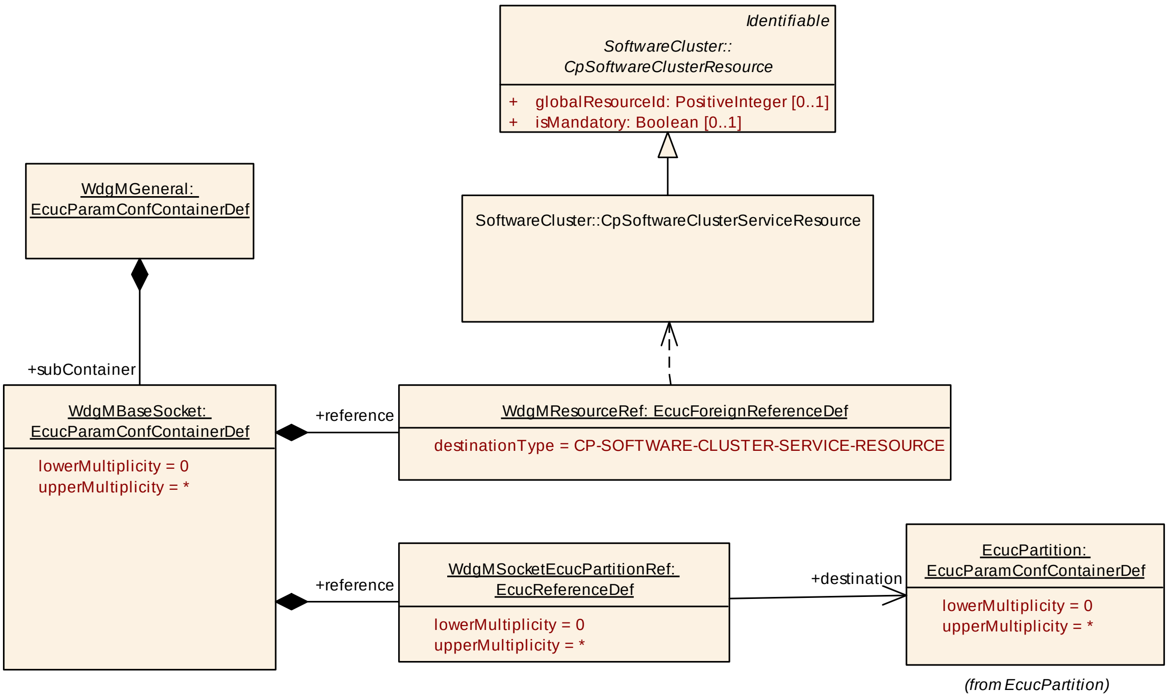
Identifiable
SoftwareCluster::
CpSoftwareClusterResource
+ globalResourceId: PositiveInteger [0..1]
+ isMandatory: Boolean [0..1]
WdgMGeneral:
EcucParamConfContainerDef
SoftwareCluster::CpSoftwareClusterServiceResource
+subContainer
WdgMBaseSocket:
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+reference
WdgMResourceRef: EcucForeignReferenceDef
destinationType = CP-SOFTWARE-CLUSTER-SERVICE-RESOURCE
+reference
WdgMSocketEcucPartitionRef:
EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = *
+destination
EcucPartition:
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
(from EcucPartition)

Figure 33: Configuration Container WdgMBaseSocket (for Clustered Software Architecture)

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral.*

11 Annex A: Example Implementation of Alive Supervision Algorithm

For the *Alive Supervision*, an algorithm to detect mismatching timing constraints of the *Checkpoints* is provided in order to clearly define the parameters needed for the *Alive Supervision*.

Doing this with incremental *Alive Counters* for the *Checkpoints* brings up a representation of aliveness by a counted number of *alive indications* in relationship with the *Alive Supervision* period.

With this approach, it must be possible to deal with two different scenarios:

A) The *alive indications* of a *Checkpoint* are expected to occur at least one time within one *supervision cycle*. The number of *alive indications (AI)* within one *supervision cycle (SC)* shall be counted.

B) The *alive indication* of a *Checkpoint* is expected to occur less often than the *supervision cycle*. The number of *supervision cycles (SC)* between two *alive indications (AI)* has to be counted.

To cope with these two scenarios, it is necessary to count both AI and SC.

We also need the parameter `WdgMExpectedAliveIndications` [ECUC_WdgM_00311] (EAI) which represents the expected amount of *alive indications* of the *Checkpoint* within the referenced amount of *supervision cycles* also called *supervision reference cycle* [ECUC_WdgM_00310] (SRC). The value of this parameter should have been determined during the design phase and defined by configuration.

To avoid the detection of too many supervision errors for the *Checkpoints*, there are parameters `WdgMMinMargin` [ECUC_WdgM_00312] and `WdgMMaxMargin` [ECUC_WdgM_00313] to define tolerances on the timing constraints.

`WdgMMinMargin` represents the allowed number of missing executions of the *Checkpoint*.

`WdgMMaxMargin` represents the allowed number of additional executions of the *Checkpoint*.

Therefore, the algorithm becomes:

(n (AI) – n (SC) + f(EAI, SRC) <= WdgMMaxMargin) and

(n (AI) – n (SC) + f(EAI, SRC) >= - WdgMMinMargin),

where the function f is defined as

f(EAI, SRC) = SRC - EAI .

Note that f(EAI, SRC) has a constant value and can be preliminary computed if EAI and SRC are constant.

11.1 Scenario A

The *alive indications* (AI) of a *Checkpoint* are expected to occur at least one time within one *supervision cycle*.

Example: 2 alive indications are expected in one supervision cycle which represents the supervision reference cycle then the value of f(EAI, SRC) is:

f(EAI, SRC) = 1 - 2 = -1

When SC occurs, the number of supervision cycles is incremented (n (SC) = 1) and the regularly checkup is performed during each supervision cycle (supervision reference cycle = 1 supervision cycle) with the algorithm.

After performing the check, the current numbers of alive indications and supervision cycles are reset.

For our examples, Max and Min margins are set to 0 for more simplicity, so the algorithm used is

n (AI) – n (SC) + f(EAI, SRC) = 0.

This brings the compare algorithm to a negative result if not enough alive indications occurred before the supervision cycle. If the number of alive indications fits exactly to the expected number, the result is 0. If more alive indications have occurred, the number is bigger than 0.

The result of the algorithm represents exactly the number of "extra" alive indications within the last supervision cycle.

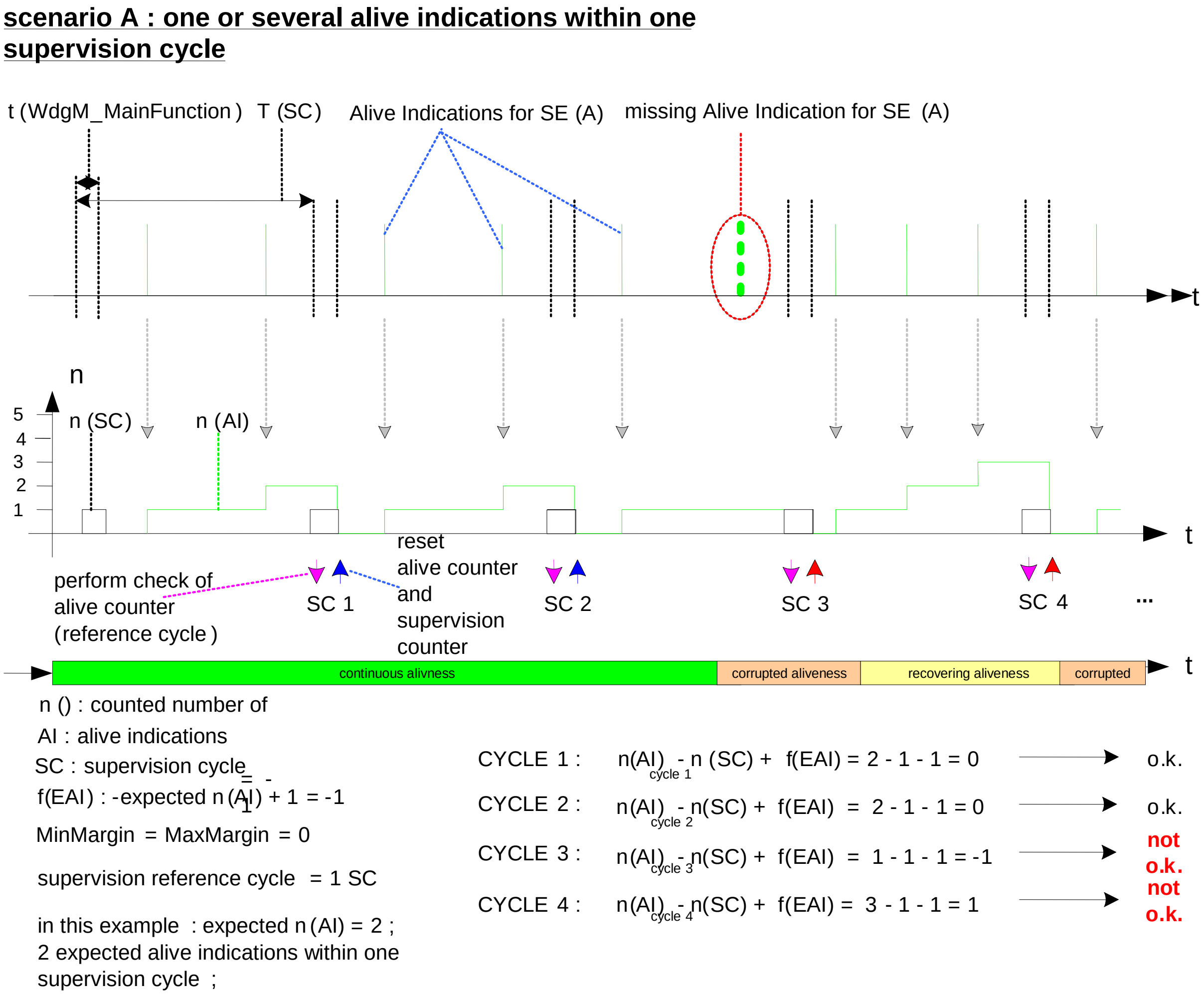
scenario A : one or several alive indications within one supervision cycle
t (WdgM_MainFunction)
T (SC)
Alive Indications for SE (A)
missing Alive Indication for SE (A)
t
n
5
4
3
2
1
n (SC)
n (AI)
t
perform check of alive counter (reference cycle)
reset alive counter and supervision counter
SC 1
SC 2
SC 3
SC 4
...
continuous alivness
corrupted aliveness
recovering aliveness
corrupted
t
n () : counted number of
AI : alive indications
SC : supervision cycle
f(EAI) : -expected n (AI) + 1 = -1
MinMargin = MaxMargin = 0
supervision reference cycle = 1 SC
in this example : expected n (AI) = 2 ; 2 expected alive indications within one supervision cycle ;
CYCLE 1 : n(AI)cycle 1 - n (SC) + f(EAI) = 2 - 1 - 1 = 0 o.k.
CYCLE 2 : n(AI)cycle 2 - n(SC) + f(EAI) = 2 - 1 - 1 = 0 o.k.
CYCLE 3 : n(AI)cycle 3 - n(SC) + f(EAI) = 1 - 1 - 1 = -1 not o.k.
CYCLE 4 : n(AI)cycle 4 - n(SC) + f(EAI) = 3 - 1 - 1 = 1 not o.k.

Figure 34: Alive-supervision algorithm – Scenario A

11.2 Scenario B

The *supervision cycle* is expected more often than the *alive indication*. In this case, we have to count the *supervision cycles*, which have occurred, until the *Alive Counter* is incremented again. The check of aliveness should be performed during each *supervision reference cycle* and the same algorithm should be used:

n (AI) – n (SC) + f(EAI, SRC) = 0

The *alive indication* must occur at least within a predefined number of *supervision cycles* which represent the *supervision reference cycle*.

Example: one *alive indication* is expected within 2 *supervision cycles* (*supervision reference cycle* = 2 *supervision cycles*):

f(EAI, SRC) = 2 – 1 = +1

The *Alive Counter* has to be incremented by 1 with every *alive indication*. Aliveness should be evaluated in the *supervision cycle* corresponding to the *supervision reference cycle*. The compare-conditions of the algorithm remain in the same manner, but the detected incrementation of the *Alive Counter* should also invoke a reset of the *Alive Counter* and *Supervision Counter* after this compare-operation.

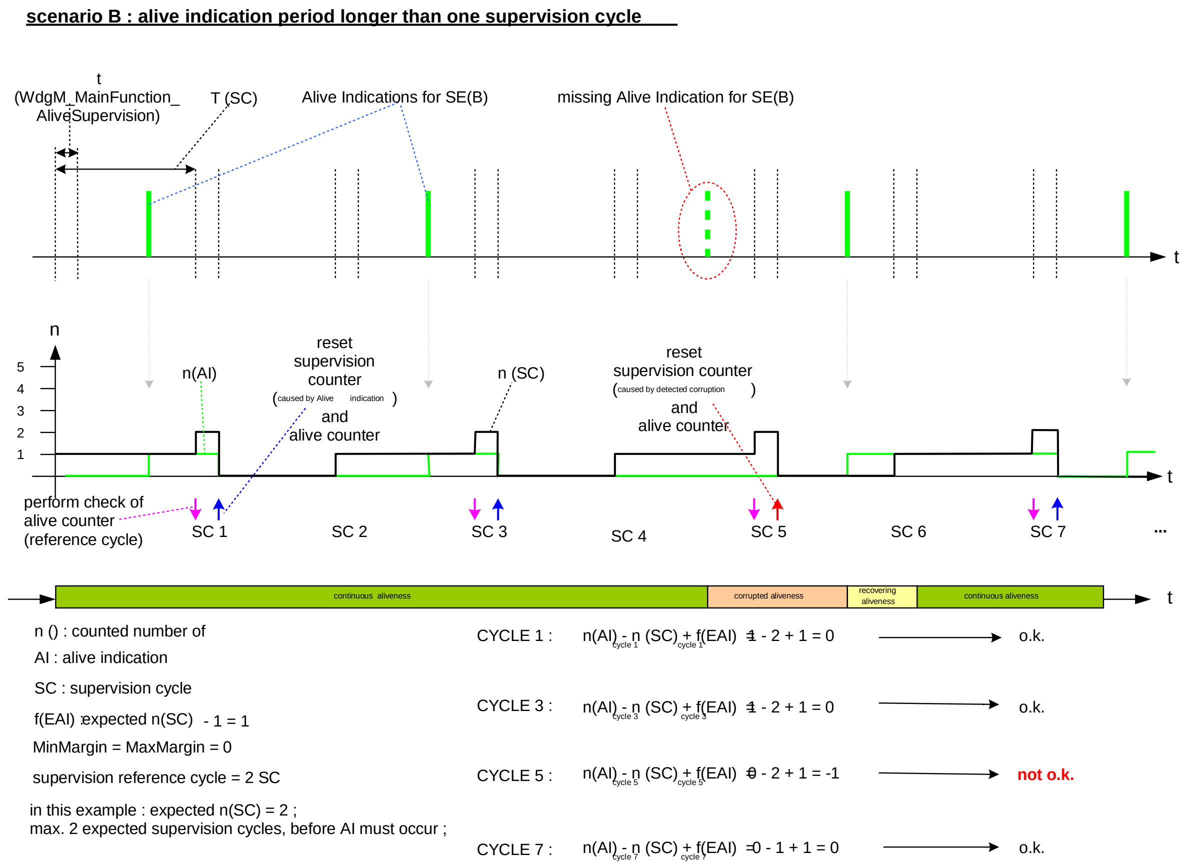
scenario B : alive indication period longer than one supervision cycle
t
(WdgM_MainFunction_
AliveSupervision)
T (SC)
Alive Indications for SE(B)
missing Alive Indication for SE(B)
t
n
5
4
3
2
1
n(AI)
reset
supervision
counter
(caused by Alive indication)
and
alive counter
n (SC)
reset
supervision counter
(caused by detected corruption)
and
alive counter
t
perform check of
alive counter
(reference cycle)
SC 1
SC 2
SC 3
SC 4
SC 5
SC 6
SC 7
...
continuous aliveness
corrupted aliveness
recovering aliveness
continuous aliveness
t
n () : counted number of
AI : alive indication
SC : supervision cycle
f(EAI) :expected n(SC) - 1 = 1
MinMargin = MaxMargin = 0
supervision reference cycle = 2 SC
in this example : expected n(SC) = 2 ;
max. 2 expected supervision cycles, before AI must occur ;
CYCLE 1 : n(AI)cycle 1 - n (SC)cycle 1 + f(EAI) = 1 - 2 + 1 = 0 o.k.
CYCLE 3 : n(AI)cycle 3 - n (SC)cycle 3 + f(EAI) = 1 - 2 + 1 = 0 o.k.
CYCLE 5 : n(AI)cycle 5 - n (SC)cycle 5 + f(EAI) = 0 - 2 + 1 = -1 not o.k.
CYCLE 7 : n(AI)cycle 7 - n (SC)cycle 7 + f(EAI) = 0 - 1 + 1 = 0 o.k.

Figure 35: Alive Supervision algorithm – Scenario B

12 Not applicable requirements

[SWS_WdgM_NA_00345] ⌈These requirements are not applicable to this specification.⌋ (SRS_BSW_00003, SRS_BSW_00004, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00159, SRS_BSW_00160, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00344, SRS_BSW_00346, SRS_BSW_00347, SRS_BSW_00348, SRS_BSW_00351, SRS_BSW_00353, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00369, SRS_BSW_00374, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00407, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00413, SRS_BSW_00414, SRS_BSW_00415, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00447, SRS_BSW_00448, SRS_BSW_00449, SRS_BSW_00451, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00457, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00472, SRS_BSW_00473, SRS_BSW_00477, SRS_BSW_00478, SRS_BSW_00479, SRS_BSW_00482, SRS_BSW_00483, SRS_BSW_00484, SRS_BSW_00485, SRS_BSW_00486, SRS_BSW_00488, SRS_BSW_00489, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00492, SRS_BSW_00493, SRS_BSW_00494)