Document Title	Specification of Watchdog Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	080
Document Classification	Standard
Document Version	2.5.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	2.5.0	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
31.10.2013	2.4.0	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
25.02.2013	2.3.0	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
02.11.2011	2.2.0	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

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
07.10.2010	2.1.0	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
07.12.2009	2.0.0	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
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
05.12.2007	1.2.0	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	**Version**	**Changed by**	**Change Description**
31.01.2007	1.1.0	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
12.05.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

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 monitoring – for aperiodic software
3. Logical monitoring – 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-Cs or a Runnable within an SW-C, a BSW module or CDD 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 Manger 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 settings 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 verifies also 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 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 Watchodog Manager checks if some steps in a Supervised Entity take a time that is within the configured minimum and maximum

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 ISO26262) 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, 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 Partition Shutdown

If the Watchdog Manager module detects a supervision failure in a *Supervised Entity* which is located in a non-trusted partition, the Watchdog Manager module may request a partition shutdown by calling the BswM.

1.5.3 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.

1.5.4 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	Development Error Tracer
FiM	Function Inhibition Manager
EAI	Expected Alive Indications
EcuM	ECU State Manager
HW	Hardware
ID	Identifier
MCU	Micro Controller Unit
OS	Operating System
SC	Supervision Cycle
SE	Supervised Entity
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 then 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 a 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 Indications*.

Term	Description
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.
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 a Software Component or Basic Software Module. There may be zero, one or more *Supervised Entities* in a Software Component or Basic Software Module.
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 check against counted *Alive Indications*.
Supervision Cycle	The time period of Watchdog Manager, where the cyclic Alive Supervision is performed. This is done by the main function of Watchdog Manager.
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*).

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:

- For Logical Supervision, Watchdog manager does not support any overlapping graphs - a checkpoint shall belong to maximum one Graph. This is required to be able to allocate a received Checkpoint notification to a Graph. This means that:
  - No checkpoint shall belong to two external graphs,
  - No checkpoint shall belong to two internal graphs,
  - No checkpoint shall belong to one internal and one external graphs.
- Watchdog Manager does not support Logical Supervision of concurrently executed Supervised Entities, because it follows only one instance of a Graph at a time. This means that the current specification of Watchdog Manager does not support the following:
  - Logical Supervision of functions of BSW modules that are executed in more than one task.
- Libraries cannot call BSWs, so libraries cannot be supervised by Watchdog Manager.
- It is not standardized how BSW modules are identified with Supervised Entity IDs.
- The Deadline Supervision has a weakness: it only detects the delays (when the End Checkpoint is reported), but it does not detect the timeouts (when the End Checkpoint is not reported at all).
- 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 Supervision Entity.
- In order to shutdown or restart (as error reaction) a partition containing Supervised Entities, the integrator code (OS Application's restart task) must deactivate (or deactivate + activate) all Supervised Entities of the involved partition, by calling available functions of Watchdog Manager. This is a bit complex, in future releases of this document it is considered to add a new function of Watchdog Manager for this.

Further limitations:

- The Watchdog Manager does not encapsulate the Watchdog Driver initialization. The Watchdog Driver initialization will be performed by the ECU State Manager [6] early in the startup process.
- 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.
- The following is neded for the operation of WdgM monitoring:
    - Initialized Wdg Interface,
    - Initialized OS (because of possible usage of OSCounter)
    - Initialized WdgM (done by calling WdgM_Init)
    - Periodic invocation of WdgM_MainFunction preferrably by AUTOSAR scheduler; during startup the invocation may be done by another module.
- 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.

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.
- Development Error Tracer (Det)
  If development error detection is enabled, the Watchdog Manager module informs the Development 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.
- BSW Mode Manager (BswM)
  The Basic Software Mode Manager is responsible for restarting a non-trusted partition. A Supervised Entity can be associated to an OS Application. If the supervision of the Supervised Entity fails, the Watchdog Manager requests a restart of the corresponding partition.
- Operating system (OS)
  The Operating System is used by Watchog Manager to provide the timestamp.

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.*

5.1.2 Header File Structure

[SWS_WdgM_00369]⌈ The module header file WdgM.h shall include Rte_WdgM_Type.h to include the types which are common used by BSW Modules and Software Components. WdgM.h file shall only contain types, that are not already defined or included via in Rte_WdgM_Type.h.⌋(SRS_BSW_00447)

[SWS_WdgM_00014]⌈The file include structure shall be as follows:

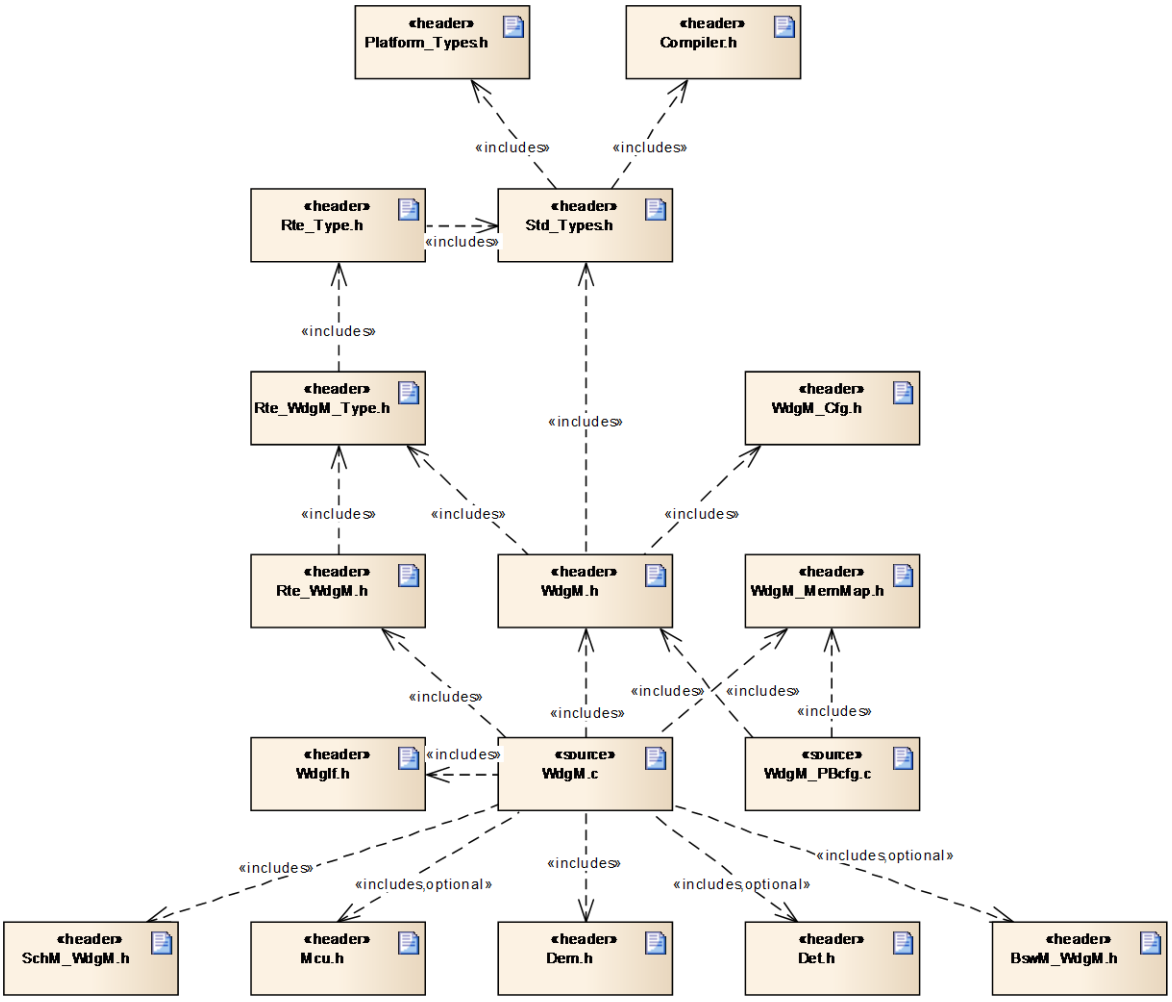

Figure 1: File include structure for the Watchdog Manager

Gray boxes are optional.⌋(SRS_BSW_00301, SRS_BSW_00346, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00381, SRS_BSW_00383, SRS_BSW_00409, SRS_BSW_00412, SRS_BSW_00415, SRS_BSW_00435, SRS_BSW_00436, SRS_BSW_00158)

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
-	-	SWS_WdgM_00029
-	-	SWS_WdgM_00146
-	-	SWS_WdgM_00147
-	-	SWS_WdgM_00149
-	-	SWS_WdgM_00161
-	-	SWS_WdgM_00162
-	-	SWS_WdgM_00170
-	-	SWS_WdgM_00171
-	-	SWS_WdgM_00178
-	-	SWS_WdgM_00179
-	-	SWS_WdgM_00181
-	-	SWS_WdgM_00182
-	-	SWS_WdgM_00186
-	-	SWS_WdgM_00195
-	-	SWS_WdgM_00196
-	-	SWS_WdgM_00197
-	-	SWS_WdgM_00198
-	-	SWS_WdgM_00199
-	-	SWS_WdgM_00200
-	-	SWS_WdgM_00201
-	-	SWS_WdgM_00202
-	-	SWS_WdgM_00203
-	-	SWS_WdgM_00204
-	-	SWS_WdgM_00205
-	-	SWS_WdgM_00206
-	-	SWS_WdgM_00207
-	-	SWS_WdgM_00208
-	-	SWS_WdgM_00209
-	-	SWS_WdgM_00212
-	-	SWS_WdgM_00216
-	-	SWS_WdgM_00217
-	-	SWS_WdgM_00218
-	-	SWS_WdgM_00221
-	-	SWS_WdgM_00225
-	-	SWS_WdgM_00228
-	-	SWS_WdgM_00229

-	-	SWS_WdgM_00232
-	-	SWS_WdgM_00233
-	-	SWS_WdgM_00238
-	-	SWS_WdgM_00239
-	-	SWS_WdgM_00240
-	-	SWS_WdgM_00241
-	-	SWS_WdgM_00245
-	-	SWS_WdgM_00268
-	-	SWS_WdgM_00269
-	-	SWS_WdgM_00275
-	-	SWS_WdgM_00282
-	-	SWS_WdgM_00283
-	-	SWS_WdgM_00285
-	-	SWS_WdgM_00286
-	-	SWS_WdgM_00290
-	-	SWS_WdgM_00291
-	-	SWS_WdgM_00293
-	-	SWS_WdgM_00294
-	-	SWS_WdgM_00295
-	-	SWS_WdgM_00296
-	-	SWS_WdgM_00297
-	-	SWS_WdgM_00298
-	-	SWS_WdgM_00299
-	-	SWS_WdgM_00300
-	-	SWS_WdgM_00304
-	-	SWS_WdgM_00305
-	-	SWS_WdgM_00306
-	-	SWS_WdgM_00308
-	-	SWS_WdgM_00309
-	-	SWS_WdgM_00310
-	-	SWS_WdgM_00311
-	-	SWS_WdgM_00313
-	-	SWS_WdgM_00314
-	-	SWS_WdgM_00315
-	-	SWS_WdgM_00316
-	-	SWS_WdgM_00318
-	-	SWS_WdgM_00319
-	-	SWS_WdgM_00320
-	-	SWS_WdgM_00321

-	-	SWS_WdgM_00322
-	-	SWS_WdgM_00323
-	-	SWS_WdgM_00324
-	-	SWS_WdgM_00325
-	-	SWS_WdgM_00326
-	-	SWS_WdgM_00327
-	-	SWS_WdgM_00328
-	-	SWS_WdgM_00329
-	-	SWS_WdgM_00331
-	-	SWS_WdgM_00332
-	-	SWS_WdgM_00333
-	-	SWS_WdgM_00334
-	-	SWS_WdgM_00335
-	-	SWS_WdgM_00336
-	-	SWS_WdgM_00338
-	-	SWS_WdgM_00344
-	-	SWS_WdgM_00346
-	-	SWS_WdgM_00347
-	-	SWS_WdgM_00348
-	-	SWS_WdgM_00349
-	-	SWS_WdgM_00351
-	-	SWS_WdgM_00354
-	-	SWS_WdgM_00355
-	-	SWS_WdgM_00356
-	-	SWS_WdgM_00357
-	-	SWS_WdgM_00358
-	-	SWS_WdgM_00359
-	-	SWS_WdgM_00360
-	-	SWS_WdgM_00366
-	-	SWS_WdgM_00370
-	-	SWS_WdgM_00371
-	-	SWS_WdgM_00372
-	-	SWS_WdgM_00373
-	-	SWS_WdgM_00374
BSW00431	-	SWS_WdgM_00345
BSW00434	-	SWS_WdgM_00345
BSW09111	-	SWS_WdgM_00119, SWS_WdgM_00120, SWS_WdgM_00121, SWS_WdgM_00122, SWS_WdgM_00292
BSW09142	-	SWS_WdgM_00083

SRS_BSW_00005	Modules of the æC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_WdgM_00345
SRS_BSW_00006	The source code of software modules above the æC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_WdgM_00345
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	SWS_WdgM_00345
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_WdgM_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_00345
SRS_BSW_00158	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation	SWS_WdgM_00014
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_WdgM_00345
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_WdgM_00345
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_WdgM_00345
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_WdgM_00345
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_WdgM_00345

SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_WdgM_00345
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_WdgM_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_00345
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_WdgM_00345
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_WdgM_00014
SRS_BSW_00304	-	SWS_WdgM_00345
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_WdgM_00345
SRS_BSW_00307	Global variables naming convention	SWS_WdgM_00345
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_WdgM_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_00345
SRS_BSW_00310	API naming convention	SWS_WdgM_00151, SWS_WdgM_00153, SWS_WdgM_00154, SWS_WdgM_00155, 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_00345
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_WdgM_00345
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_WdgM_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_00027, SWS_WdgM_00028, 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_00255, SWS_WdgM_00256, SWS_WdgM_00257, SWS_WdgM_00258, SWS_WdgM_00270, SWS_WdgM_00278, SWS_WdgM_00279, SWS_WdgM_00284, SWS_WdgM_00288
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_WdgM_00345
SRS_BSW_00326	-	SWS_WdgM_00345
SRS_BSW_00327	Error values naming convention	SWS_WdgM_00004, SWS_WdgM_00364
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_WdgM_00345
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_WdgM_00345
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_WdgM_00345
SRS_BSW_00335	Status values naming convention	SWS_WdgM_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_00364
SRS_BSW_00338	-	SWS_WdgM_00010, SWS_WdgM_00020, SWS_WdgM_00021, SWS_WdgM_00027, SWS_WdgM_00028, SWS_WdgM_00030, SWS_WdgM_00031, SWS_WdgM_00039

SRS_BSW_00339	Reporting of production relevant error status	SWS_WdgM_00129, SWS_WdgM_00142
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_WdgM_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_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_00345
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_WdgM_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 of module files	SWS_WdgM_00014
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_WdgM_00345
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_WdgM_00014
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_00014
SRS_BSW_00355	-	SWS_WdgM_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	SWS_WdgM_00345

	avoid return types other than void if possible	
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_WdgM_00345
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_WdgM_00014
SRS_BSW_00370	-	SWS_WdgM_00345
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_WdgM_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_00375	Basic Software Modules shall report wake-up reasons	SWS_WdgM_00345
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_WdgM_00345
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_WdgM_00345
SRS_BSW_00381	The pre-compile time parameters shall be placed into a separate configuration header file	SWS_WdgM_00014
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_00014
SRS_BSW_00385	List possible error notifications	SWS_WdgM_00004, SWS_WdgM_00364
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_WdgM_00345
SRS_BSW_00387	The Basic Software Module specifications shall specify how the callback function is to be	SWS_WdgM_00345

	implemented	
SRS_BSW_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_WdgM_00345
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_WdgM_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_00028, SWS_WdgM_00039
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_00014
SRS_BSW_00412	References to c-configuration parameters shall be placed into a separate h-file	SWS_WdgM_00014
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_WdgM_00345
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_WdgM_00014
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_WdgM_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_00345
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_WdgM_00345
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_WdgM_00345
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_WdgM_00345
SRS_BSW_00425	The BSW module	SWS_WdgM_00345

	description template shall provide means to model the defined trigger conditions of schedulable objects	
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_WdgM_00345
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_WdgM_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_00345
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_WdgM_00345
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_WdgM_00345
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_WdgM_00345
SRS_BSW_00435	-	SWS_WdgM_00014
SRS_BSW_00436	-	SWS_WdgM_00014
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_WdgM_00345
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_WdgM_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_00345
SRS_BSW_00447	Standardizing Include file structure of BSW Modules Implementing Autosar Service	SWS_WdgM_00369
SRS_ModeMgm_09028	The Watchdog Manager shall support multiple	SWS_WdgM_00002

	watchdog instances	
SRS_ModeMgm_09106	The list of entities supervised by the Watchdog Manager shall be configurable at pre-compile time	SWS_WdgM_00042, 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_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
SRS_ModeMgm_09125	The Watchdog Manager shall provide a service allowing the Update temporal program flow monitoring	SWS_WdgM_00155
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 selectable configuration sets for the Watchdog Manager	SWS_WdgM_00145
SRS_ModeMgm_09159	The Watchdog Manager shall report failure of temporal or program flow monitoring to DEM	SWS_WdgM_00129
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	SWS_WdgM_00150

	watchdog reset	
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
SRS_ModeMgm_09221	The Watchdog Manager shall check the correct sequence of code execution in supervised entities	SWS_WdgM_00242, SWS_WdgM_00246, SWS_WdgM_00252, SWS_WdgM_00271, SWS_WdgM_00273, SWS_WdgM_00274
SRS_ModeMgm_09222	The Watchdog Manager shall provide a service allowing the Update logical program flow monitoring	SWS_WdgM_00242, SWS_WdgM_00246, SWS_WdgM_00252, SWS_WdgM_00271, SWS_WdgM_00273, SWS_WdgM_00274
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 7.1.5)
- Deadline Supervision (see Chapter 7.1.6)
- Logical Supervision (see Chapter 7.1.7)

Each of three Supervision Functions results with a list of *Results of Supervision Function* for each *Supervised Entity* (highlighted in Blue on Figure 2), where each *Result* is either `correct` or `incorrect`. At Watchdog Manager initialization, all the R*esults* 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.

In a given mode, each Supervised entity may have zero, one or more Logical Supervisions (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.

Based on the results of *Supervisions Functions* (`correct`/`incorrect`), the *Local Status* of each Supervision Entity (highlighted in Green on Figure 2) 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 2) is determined by means of *Global Supervision Status* state machine (see Chapter 7.1.4).

Based on the Global Supervision Status, the error handling (see Chapter 7.2) and watchdog handling (see Chapter 7.2) take place.

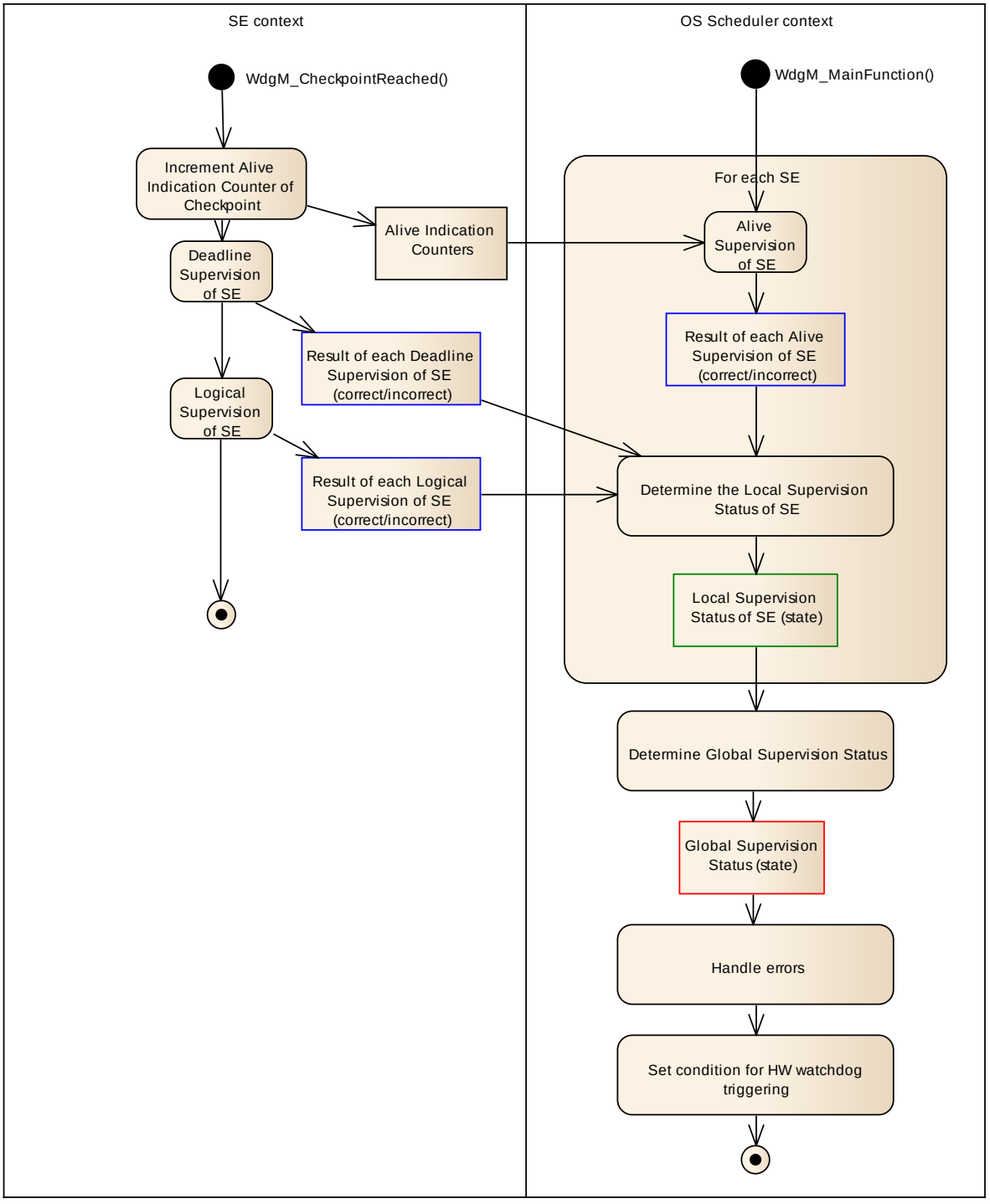

Figure 2: Overview of Watchdog Manager Monitoring

The determination of supervision result for *Deadline Supervision* and *Logical Supervision* is executed within the function `WdgM_CheckpointReached`. During one execution of this function, it updates the result for one particular *Supervision Entity* only.

The determination of supervision result for *Alive Supervision* is executed within the function `WdgM_MainFunction`. During one execution of this function, it updates the Results of *Alive Supervision* for all *Supervised Entities*.

7.1.2 Core Configurable Parameters

Supervised Entities are be defined within the container `WdgMGeneral`. (see `WdgMSupervisedEntity` [ECUC_WdgM_00303]). Supervised Entities contain *Checkpoints* (see `WdgMCheckpoint`).

7.1.3 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.

The change in the Local Status state machine is done by function `WdgM_MainFunction`. The state machine is initialized by the function `WdgM_Init`.

For the *Alive Supervision*, the state machine provides fault tolerance by means of the state WDGM_LOCAL_STATUS_FAILED and the configuration parameter WdgMFailedSupervisionRefCycleTol, allowing some failed reference cycles of deadline and

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

Possible values of the *Local Supervision Status* are described in WdgM_LocalStatusType (see Chapter 8.2.5).

[SWS_WdgM_00238]⌜The *Local Supervision Status* of each *Supervised Entity* shall be available for debugging within the Watchdog Manager module.⌟()

See chapter 7.8 for additional debugging requirements.

Figure 3 shows the state machine for *Local Supervision Status* of a *Supervised Entity* with all possible states.

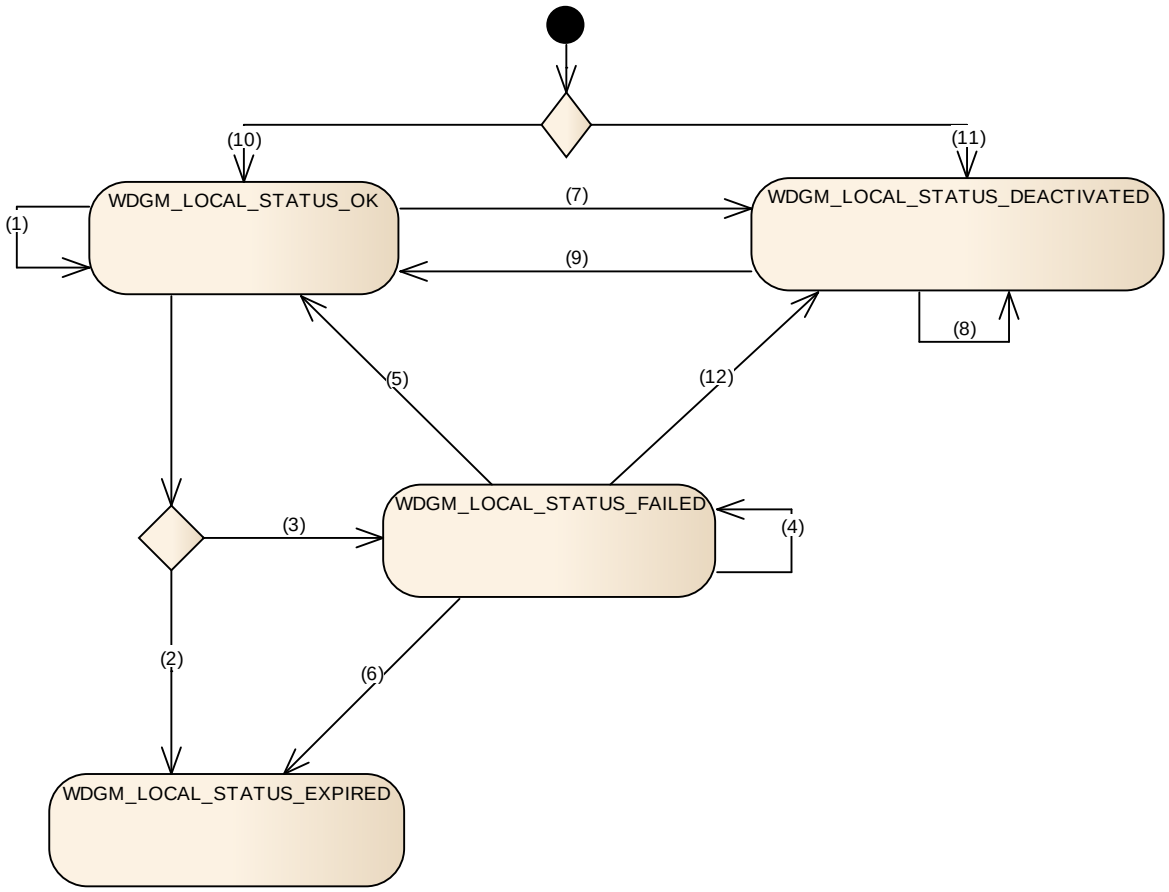

Figure 3: Local Supervision Status

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 Monitoring Status for this Supervised Entity to WDGM_LOCAL_STATUS_OK. (see Transition (10) in Figure 2).⌟()

[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 Monitoring 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 3)⌋()

[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 leave the *Supervised Entity* in the *Local Supervision Status* `WDGM_LOCAL_STATUS_OK` (see Transition (1) in Figure 3).⌋()

[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 3).⌋()

The below requirements shows 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 3).⌋()

[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* does not exceed 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 3).」()

[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 3).」()

[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 3).」()

[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* exceeds 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 3).⌋()

[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 3).⌋()

[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 3).⌋()

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 leave the *Local Supervision Status* in the state WDGM_LOCAL_STATUS_DEACTIVATED. (see Transition (8) in Figure 3)⌋()

[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. (see Transition (9) in Figure 3)⌋()

7.1.4 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 4 shows the values and *Transitions* between them.

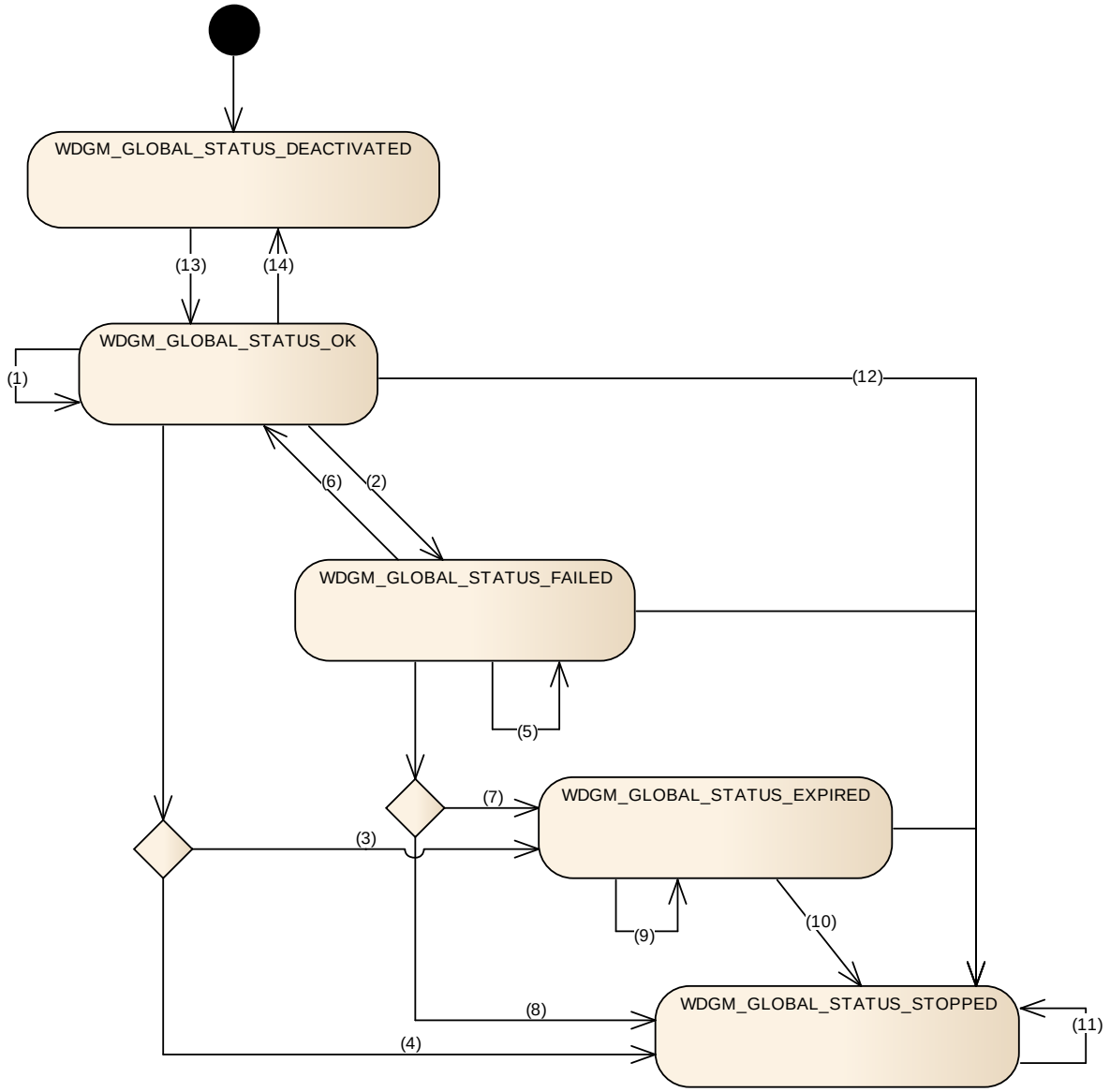

Figure 4: Global Supervision Status

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

[SWS_WdgM_00239]「The *Global Supervision Status* shall be available for debugging within the Watchdog Manager module.」()

See chapter 7.8 for additional debugging requirements.

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 (Supervision cycle is the period at which `Wdgm_MainFunction` is called), 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]⌈The function `Wdgm_MainFunction` shall calculate the *Global Supervision Status* in every Supervision cycle. The function shall compute the Global Supervision Cycle after it computed every *Local Supervision Status*.

The cyclic update of *Global Supervision Status* is necessary to trigger the timely transition from WDGM_GLOBAL_STATUS_EXPIRED to WDGM_GLOBAL_STATUS_STOPPED.⌋(SRS_ModeMgm_09112)

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 (see Transition (13) in Figure 9).⌋()

[SWS_WdgM_00286]⌈If the *Global Supervision Status* was WDGM_GLOBAL_STATUS_OK and the function *WdgM_DeInit* [SWS_WdgM_00261] was successfully called function shall change the *Global Supervision Status* to WDGM_GLOBAL_STATUS_ DEACTIVATED (see Transition (14) in Figure 9)⌋()

[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 4).⌋(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 4).⌋(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 (see Transition (3) in Figure 4).⌋(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 4).⌋()

[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 4)⌋()

[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 4).⌋()

[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 (see Transition (7) in Figure 4).⌋(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 4).」(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 or equal to 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 4).」(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 larger than 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 4).」(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 4).」()

[SWS_WdgM_00139]「If a call to `WdgIf_SetMode` fails (see chapter 7.10.2), 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. Supervision Functions

7.1.5 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*.

7.1.5.1 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 5.

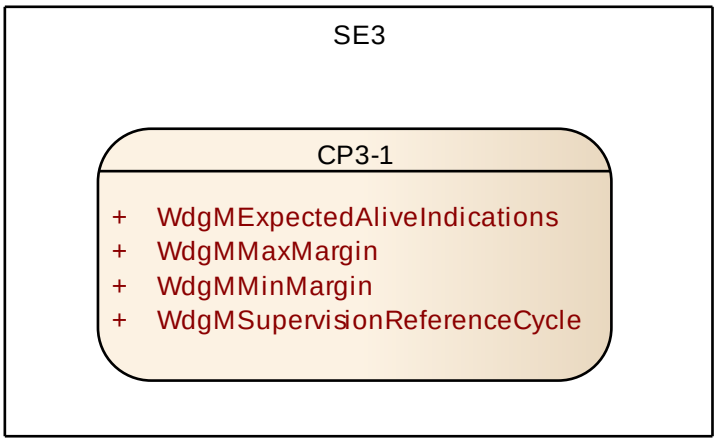

Figure 5: Simplest Alive Supervision Checkpoint Configuration

The above configuration provides backward compatibility to *Alive Supervision* as defined in versions before v2.0.0 of the Watchdog Manager module, 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 6.

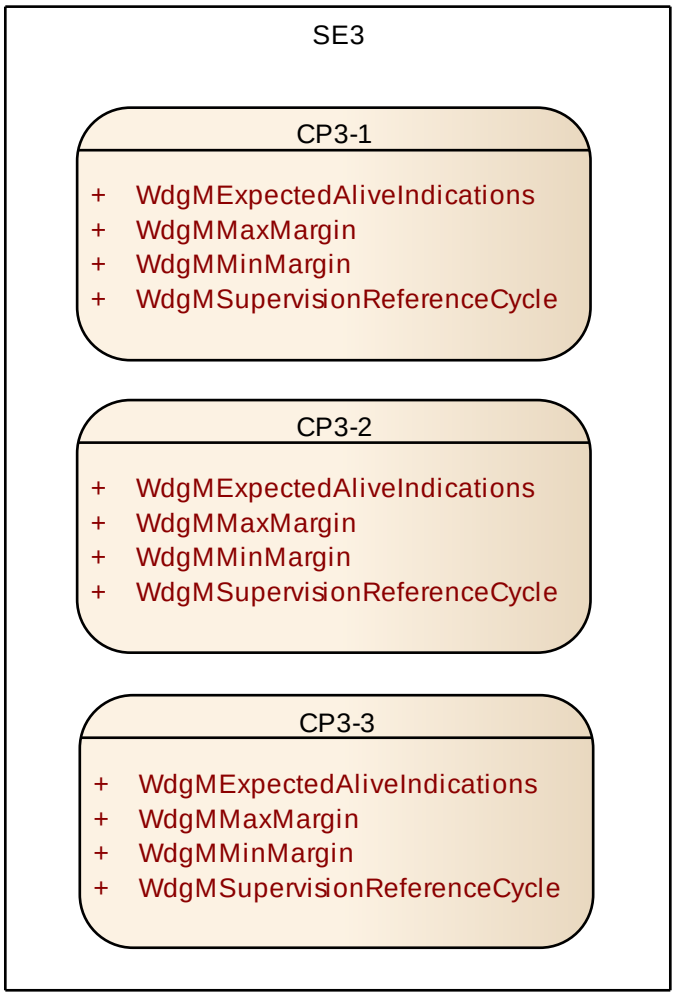

Figure 6: Multiple Checkpoints for Alive Supervision in one Supervised Entity

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 Watchog 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. The period length is defined by `WdgMSupervisionReferenceCycle` [ECUC_WdgM_00310].

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*.

[SWS_WdgM_00240]⌈The Alive Counters of each *Checkpoint* shall be available for debugging within the Watchdog Manager module.⌋()

See chapter 7.8 for additional debugging requirements.

7.1.5.2 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*.

This Main Function is executed by the AUTOSAR Scheduler with the period defined by the configuration parameter Supervision Cycle (see `WdgMSupervisionCycle`). 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 (see `WdgMSupervisionCycle`) is the property of the Watchdog Manager mode. This means that in a given mode, the function `WdgM_MainFunction` is executed with a given period. In contrary, the Supervision Reference Cycle (see `WdgMSupervisionReferenceCycle`) is the property 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`). During the intermediate *Supervision Cycles* (see `WdgMSupervisionCycle`) *of the Alive*

Supervision, the function `WdgM_MainFunction` shall not perform the examination of *Alive Counters*.」(SRS_ModeMgm_09112)

[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 Monitoring 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, BSW09142, SRS_ModeMgm_09143)

7.1.6 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).

7.1.6.1 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 7.

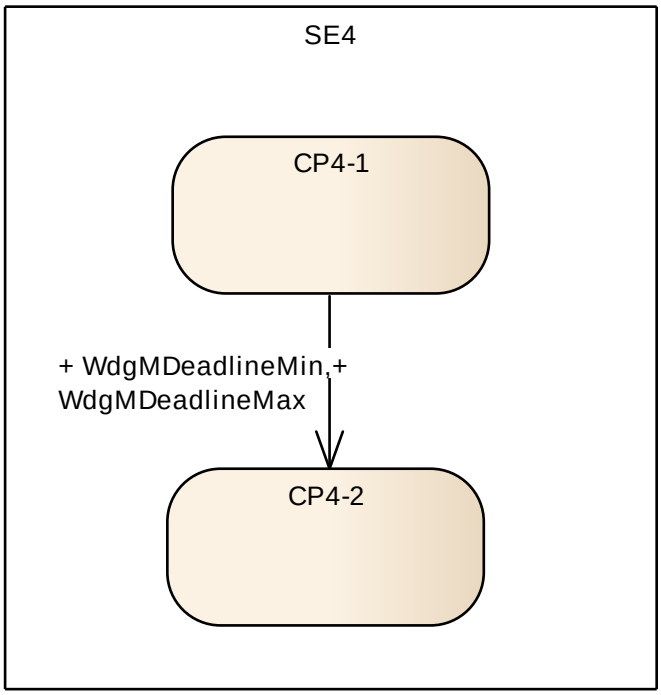

Figure 7: Simplest Deadline Supervision Configuration

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 8. Moreover, the Checkpoints can be chained.

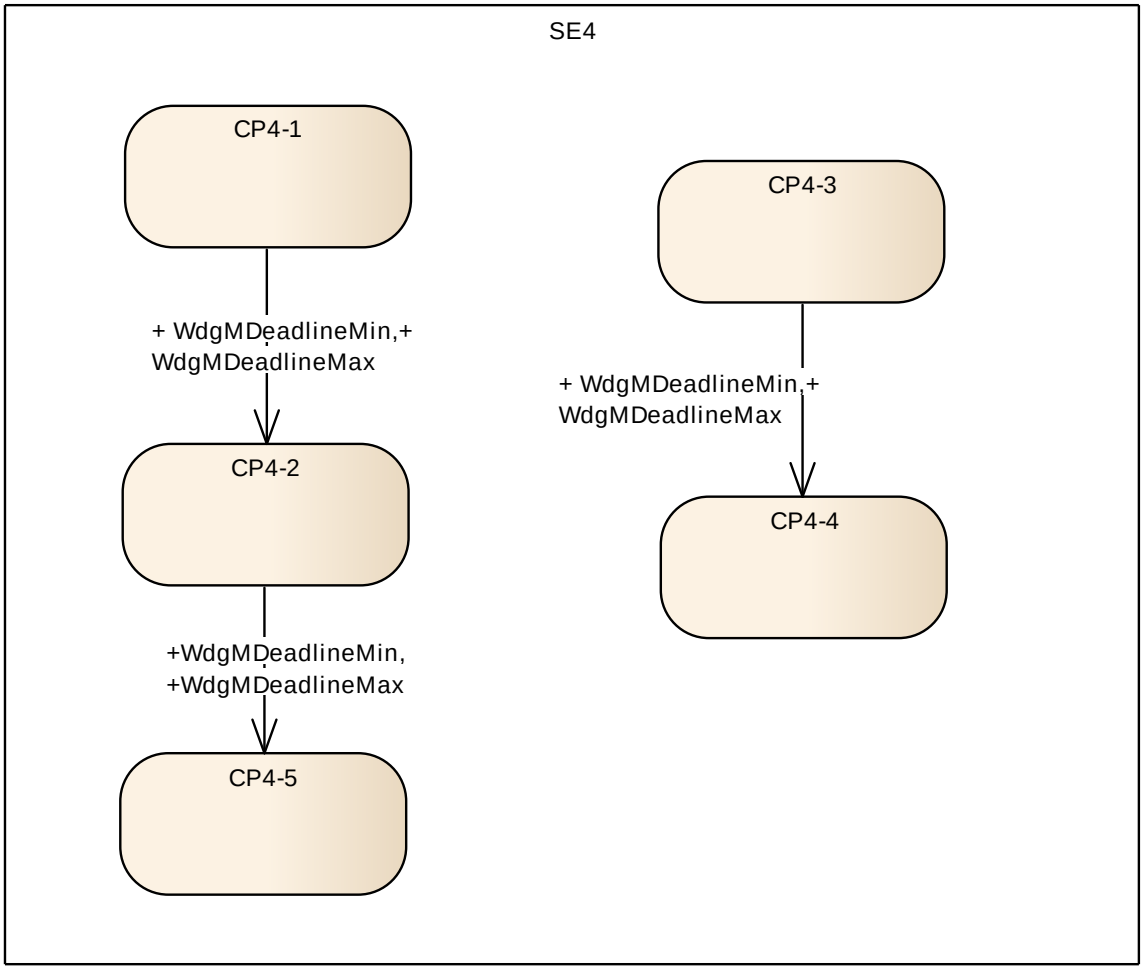

Figure 8: Multiple Transitions for Deadline Supervision in one Supervised Entity

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

The parameters of the Deadline Supervision (see `WdgMDeadlineSupervision`) depend on the Watchog 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 *WdgMAliveSupervisions.*⌋()

7.1.6.2 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 monitoring 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_00373]⌜ To determine the timestamp and to compute the timestamp differences, the function `WdgM_CheckpointReached` shall use OS functions (GetCounterValue or GetElapsedTime), using as 1st parameter the CounterID that is configured for the Supervised Entity.⌟()

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` shall use `OsSecondsPerTick` configuration parameter. ⌟()

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.⌟()

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 to 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_00229 means that the error is not detected if the *Deadline End Checkpoint* is never reached (because the *Deadline End Checkpoint* is needed to measure the time difference).

[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 (`WdgMDeadlineMin` and `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. ⌋()

[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_00241]⌈For each start *Checkpoint*, the timestamp when each *Checkpoint* has been reached and the result of Deadline Supervision for each *Supervised Entity* shall be available for debugging within the Watchdog Manager module.⌋()

7.1.7 Logical Supervision

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

7.1.7.1 Alive 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];`
CP0-4	`    else` `        a[i] = 0;`
CP0-5	`    i++;`
CP0-6	`}`

This *Supervised Entity* can be represented by the *Graph* shown by Figure 9.

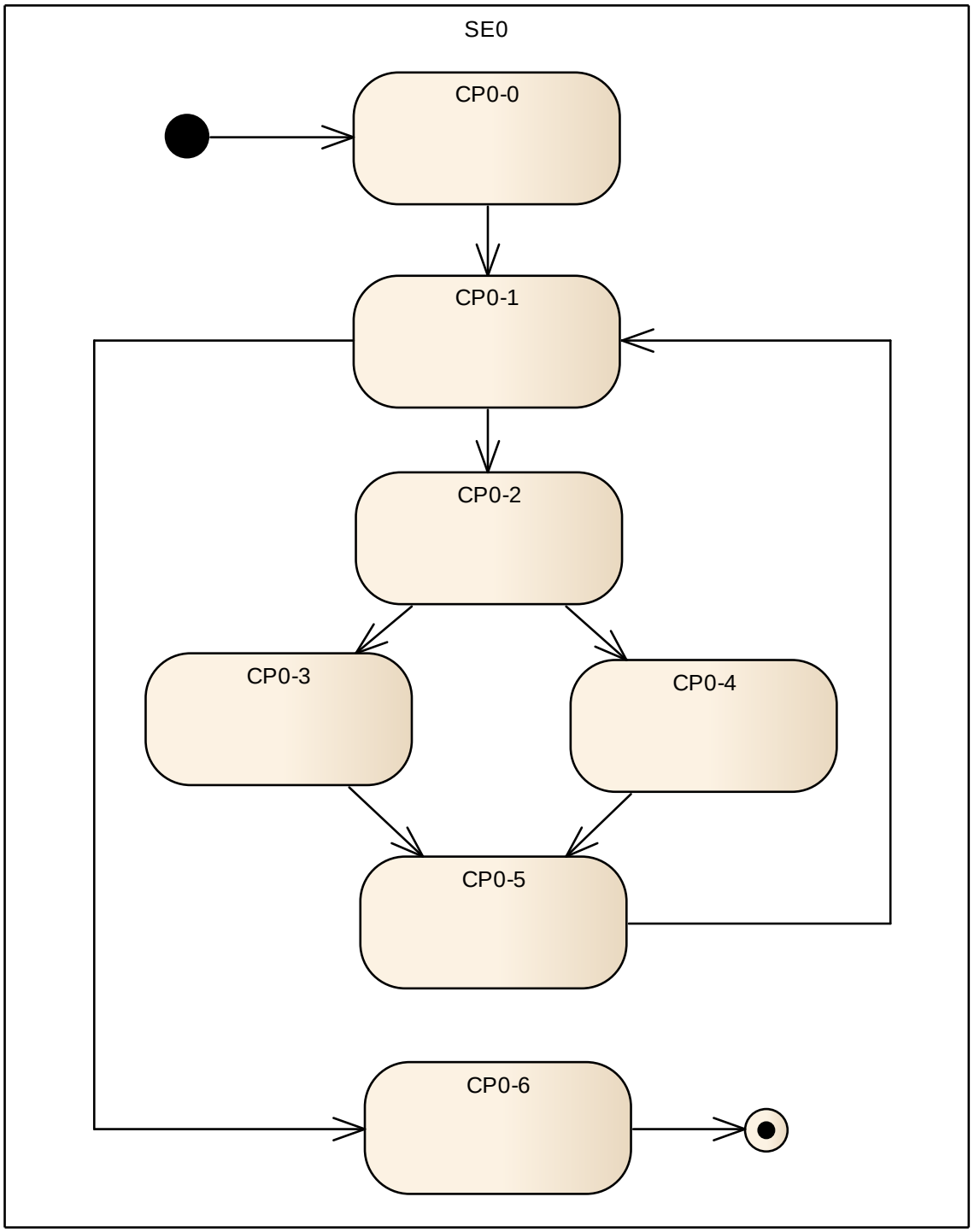

Figure 9: Example Control Flow Graph

A more abstract view of the *Supervised Entity* is given by the *Graph* shown in Figure 10, where the *Checkpoint* CP0-1 represents the complete while loop.

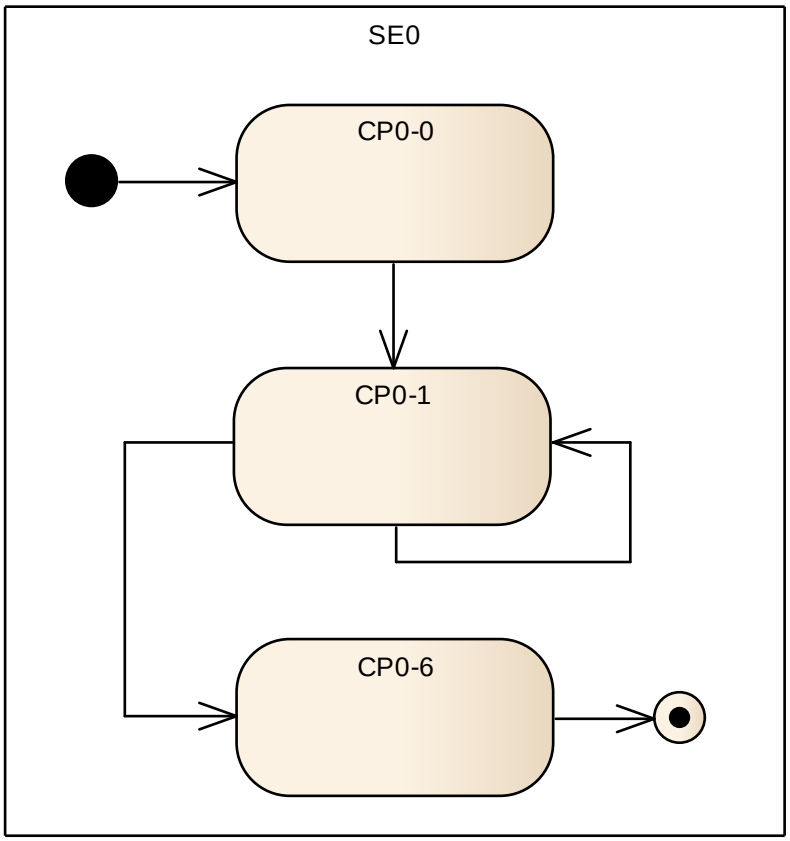

Figure 10: Abstracted Example Control Flow Graph

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*. There can be zero or one *Internal Graphs* per *Supervised Entity*.
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* and *External Graph* is that *Internal Graph* is a property of a *Supervised Entity* (it does not depend on *Watchdog Manager Mode*), whereas the *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 Logical 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 *WdgMModes* and their *WdgMExternalLogicalSupervisions and WdgMInternalTransitions.*⌋()

7.1.7.2 Logical Supervision Algorithm

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

Each Internal Graph represents as well one *Logical Supervision*. Assuming N internal graphs, this means that a Supervised Entity has N results from Logical Supervision for the 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 for each Graph an *Activity Flag*.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

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

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 only one *Graph*, the function `WdgM_CheckpointReached` is able to identify to which *Graph* a *Checkpoint* belongs.

[SWS_WdgM_00295]⌈The function `WdgM_CheckpointReached` shall identify to which one *Graph* a reached *Checkpoint* belongs.⌋()

[SWS_WdgM_00246]⌈The function `WdgM_CheckpointReached` shall store for each *External Graph* and for each *Internal Graph* the *Checkpoint* that has been most recently reported by a *Supervised Entity* (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 (i.e. *Activity Flag* is `true`), 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`.⌋(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 (i.e. activity Flag is true), 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`.

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_00332]⌈If the function `WdgM_CheckpointReached` the result correct, and the Checkpoint is defined as a initial one, then the function `WdgM_CheckpointReached` shall set the Activity Flag of the corresponding graph to `true`.⌋()

The reverse applies for the final Checkpoint.

[SWS_WdgM_00331]⌈If the function `WdgM_CheckpointReached` the result 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`.

After a final checkpoint, the only possible are initial checkpoints.⌋()

A Checkpoint can belong to either Internal or External Graph, this means that either the check defined in **SWS_WdgM_00274** or the one in **SWS_WdgM_00252** is executed. This means that in any execution of `WdgM_CheckpointReached`, if the reported checkpoint belongs to any Internal or External Graphs, the function can set the result of the Logical Supervision of one *Supervised Entity* to `correct` or `incorrect`.

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*.⌋()

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

[SWS_WdgM_00329]⌈If the function `WdgM_CheckpointReached` determines that the result of the Logical Supervision for the given Checkpoint is `true`, and the Checkpoint is the initial one (`WdgMInternalCheckpointFinalRef`), then shall set the Activity Flag of the *Graph* corresponding to the *Checkpoint* to `true`.⌋()

[SWS_WdgM_00242]⌈The information about the most recently reached Checkpoints for each Graph, the Activity Flag for Each Graph and the result of Logical Supervision for each Supervised Entity shall be available for debugging within the Watchdog Manager module.⌋(SRS_ModeMgm_09221, SRS_ModeMgm_09222)

See chapter 7.8 for additional debugging requirements

7.2 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.

7.2.1 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_00196], [SWS_WdgM_00197], [SWS_WdgM_00198]).

7.2.2 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]⌈When the *Global Supervision Status* has reached `WDGM_GLOBAL_STATUS_STOPPED` and if the configuration parameter `WdgMDemStoppedSupervisionReport` is set to TRUE, the Watchdog Manager module shall report an error status to the DEM.⌋(SRS_BSW_00339, SRS_ModeMgm_09159)

7.2.3 Partition Restart / Shutdown

If the Watchdog Manager module detects a supervision failure for a *Supervised Entity* that is located in a non-trusted partition it can restart/shutdown that partition by terminating the corresponding OS Application.

[SWS_WdgM_00225]⌈If the *Local Supervision Status* of a *Supervised Entity* changes to `WDGM_LOCAL_STATUS_FAILED` and this *Supervised Entity* has a corresponding OS Application configured (see configuration parameter `WdgMOsApplicationRef` [ECUC_WdgM_00346]), then the Watchdog Manager module shall call the API function `BswM_WdgM_RequestPartitionReset` of the Basic Software Mode Manager module to request a restart/shutdown of the corresponding partition for the configured OS Application.⌋()

7.2.4 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 7.3.2 for the corresponding requirements.

7.2.5 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.

The Watchdog Manager configuration tool should not allow to configure the parameter `WdgMImmediateReset` [ECUC_WdgM_00339] to TRUE if the `McuPerformResetApi` (defined in SWS_Mcu_Driver) is not 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)

7.3 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.

7.3.1 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)

7.3.2 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.

Following rules shall be used to derive the decision, how to set the triggering condition. In short:

1. For the states WDGM_GLOBAL_STATUS_OK, WDGM_GLOBAL_STATUS_FAILED and WDGM_GLOBAL_STATUS_EXPIRED, the function WdgM_MainFunction shall set correctly the trigger conditions.
2. For the states WDGM_GLOBAL_STATUS_DEACTIVATED, and WDGM_GLOBAL_STATUS_STOPPED, the function WdgM_MainFunction shall set the trigger conditions to 0, which results with a reset through HW watchdog(s).⌟(SRS_ModeMgm_09161, SRS_ModeMgm_09226)

[SWS_WdgM_00292]⌜If the *Global Supervision Status* has recomputed as WDGM_GLOBAL_STATUS_DEACTIVATED, then this means that the Watchdog

Manager module is not properly initialized and it shall not call `WdgIf_SetTriggerCondition`.」(BSW09111)

[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].」(BSW09111)

[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].」(BSW09111)

[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].」(BSW09111)

[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.」(BSW09111)

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

7.3.3 Configurable Parameters

Further parameters of the watchdog triggering are configurable and on the current mode of the Watchdog Manager module.

7.4 Development Errors

[SWS_WdgM_00004]⌈The Watchdog Manager module shall be able to detect the following development errors:

Type or error	*Related error code*	*Value*
API service used in wrong context (without module initialization)	WDGM_E_NO_INIT	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 a null pointer parameter	WDGM_E_INV_POINTER	0x14
Disabling of watchdog not allowed (e.g. in safety-related systems)	WDGM_E_DISABLE_NOT_ALLOWED	0x15
API service used with an invalid CheckpointId.	WDGM_E_CPID	0x16
Deprecated API service was used.	WDGM_E_DEPRECATED	0x17
Function WdgM_UpdateAliveIndication cannot determine the Checkpoint, because there are more than one alive supervisions configured in the current mode for the given Supervised Entity.	WDGM_E_AMBIGIOUS	0x18
API service used with a checkpoint of a Supervised Entity that is deactivated in the current Watchdog Manager mode.	WDGM_E_SEDEACTIVATED	0x19

⌋(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385)

7.5 Detection of Development Errors

For details refer to the chapter 7.3 “Error Detection” in *SWS_BSWGeneral.*

7.6 Production Errors

None

7.7 Extended Production Errors

[SWS_WdgM_00364]⌈The Watchdog Manager module shall be able to detect the following extended production errors:

Type or error	*Related error code*	*Value*
Supervision has failed and a watchdog reset will occur	`WDGM_E_SUPERVISION`	`assigned by DEM`
Watchdog drivers' mode switch has failed	`WDGM_E_SET_MODE`	`assigned by DEM`
Defensive behavior checks have detected an improper caller.	`WDGM E_IMPROPER_CALLER`	`assigned by DEM`

⌋(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385)

Values for production code Event Ids are assigned externally by the Diagnostic Event Manager (DEM).

7.8 Debugging Support

For details refer to the chapter 7.1.17 “Debugging support” in *SWS_BSWGeneral.*

For the Watchdog Manager module the following variables shall be accessible by AUTOSAR Debugging:

- *Local Supervision Status* of each *Supervised Entity* (see SWS_WdgM_00238)
- *Global Supervision Status* (see SWS_WdgM_00239)
- Alive Counters of each *Checkpoint* (see SWS_WdgM_00240)
- Timestamp when *Checkpoint* has been reached (see SWS_WdgM_00241)
- Reached *Checkpoints* (see SWS_WdgM_00242) ⌋ ()

7.9 Watchdog Manager Configuration

7.9.1 Mode-independent Supervision Settings

7.9.1.1 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_00304](configuration)⌈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 the Watchdog Manager module.⌋()

[SWS_WdgM_00306](configuration)⌈Each BSW module shall use its module ID as the Supervised Entity ID.⌋()

[SWS_WdgM_00305](configuration)⌈No SW-Cs shall have as Supervised Entity ID a value of any BSW Module ID, regardless which BSW Modules are deployed.⌋()

[SWS_WdgM_00307]⌈The Watchdog Manager configuration generator shall reject configurations where Supervised Entity ID is not unique and the configurations where SW-C Supervised Entities use as a Supervised Entity ID a value that is equal to the Module ID of any BSW module.⌋

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 → WdgMSupervised Entity 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 → WdgMSupervised Entity AND
2. No mode-dependent settings of Supervision Functions shall be set for the given mode (Alive, Deadline, Logical for external graphs)

Because the Logical supervision for internal graphs is a property of a Supervised Entity, the configuration of Logical supervision for internal graphs do not impact the deactivation/activation status of Supervised Entity.⌋()

7.9.1.2 OS Application

Supervised Entities can reside in trusted or non-trusted partitions. Each non-trusted partition has its memory access isolated so that its failure does not corrupt the memory of other partitions of the MCU. The partition can be terminated and restarted independently. Each partition corresponds one-to-one to and OS-Application, managed by AUTOSAR OS.

If a *Supervised Entity* has an OS-Application configured, the Watchdog Manager module requests a restart of the corresponding partition when the *Local Supervision Status* `WDGM_LOCAL_STATUS_FAILED` for that *Supervised Entity* is reached. (see chapter 7.2.3).

To enable partition restart, the Supervised Entity need to refer to an OS Application (see WdgMOsApplicationRef). The OS Application must be non-trusted.

[SWS_WdgM_00311]⌈The Watchdog Manager configuration generator shall reject the configurations where WdgMOsApplicationRef points to a trusted OS-Application (i.e. where OsTrusted the of OsApplication is `true`).⌋()

7.9.1.3 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]). One of the *Checkpoints* is marked as the initial one (see [ECUC_WdgM_00323]).

[SWS_WdgM_00212]⌈The Watchdog Manager configuration generator shall reject configurations where Internal Transitions (see WdgMInternalTransition) in a *Supervised Entity* connect *Checkpoints* that do not both belong to the same *Supervised Entity*.⌋()

To switch on and off the Logical monitoring 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 only zero, one or more Internal Graphs per Supervised Entity. Moreover, not all Checkpoints of a Supervised Entity need to be monitored. However, no checkpoint may belong to more than one Graph. This is because it is assumed that each Graph can be executed concurrently and in case of overlaps, there are no means to differentiate to which Graph a given Checkpoint would belong.

[SWS_WdgM_00308]⌈The Watchdog manager shall reject configurations where a Checkpoint belongs to more than one *Internal Graphs*.⌋()

[SWS_WdgM_00309]⌈The Watchdog manager shall reject configurations where in any mode there is a Checkpoint that belongs to an External Graph and to an 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 11 shows a configuration of two independently *Supervised Entities*, with independently configured Internal Graphs.

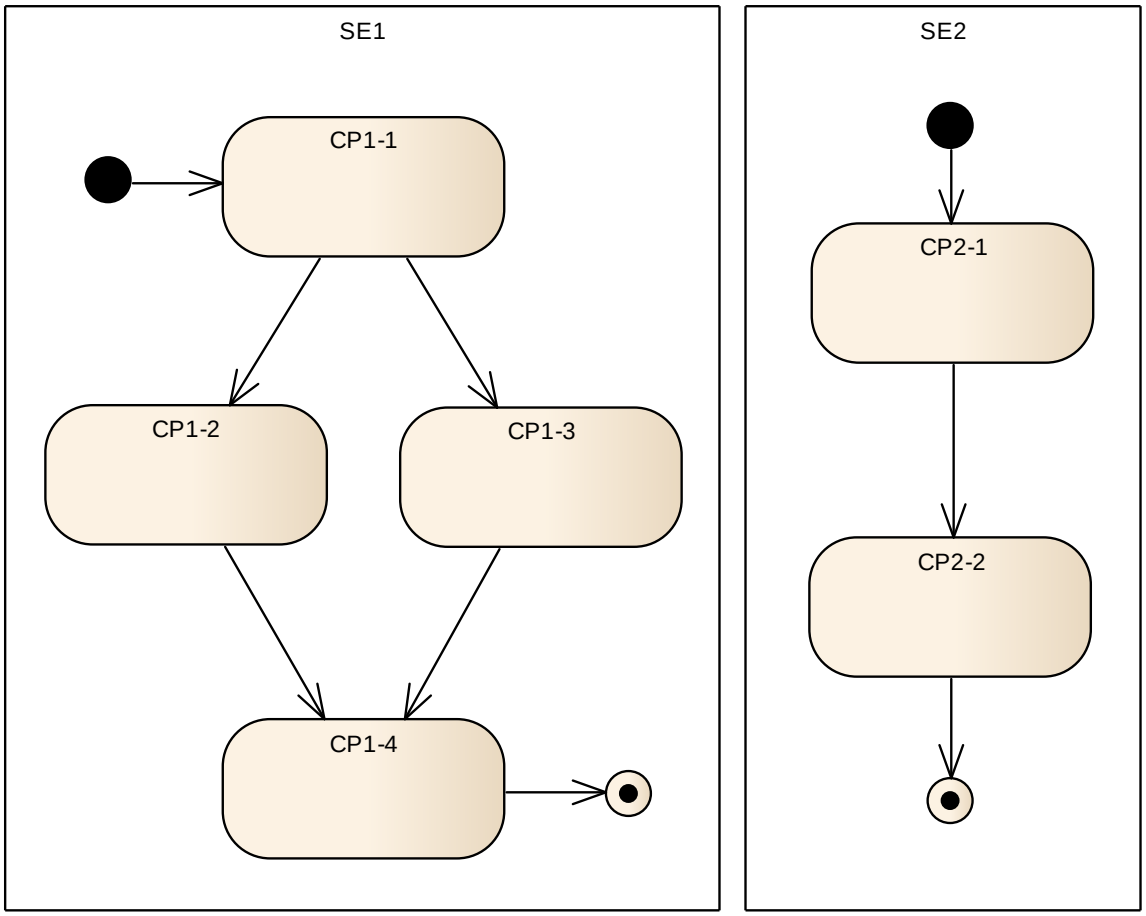

Figure 11: Two Supervised Entities with their Checkpoints and Internal Transitions

7.9.2 Mode-Dependent Parameters

7.9.2.1 Mode

Changing the mode of the Watchdog Manager module 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.

7.9.2.2 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 11, 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.

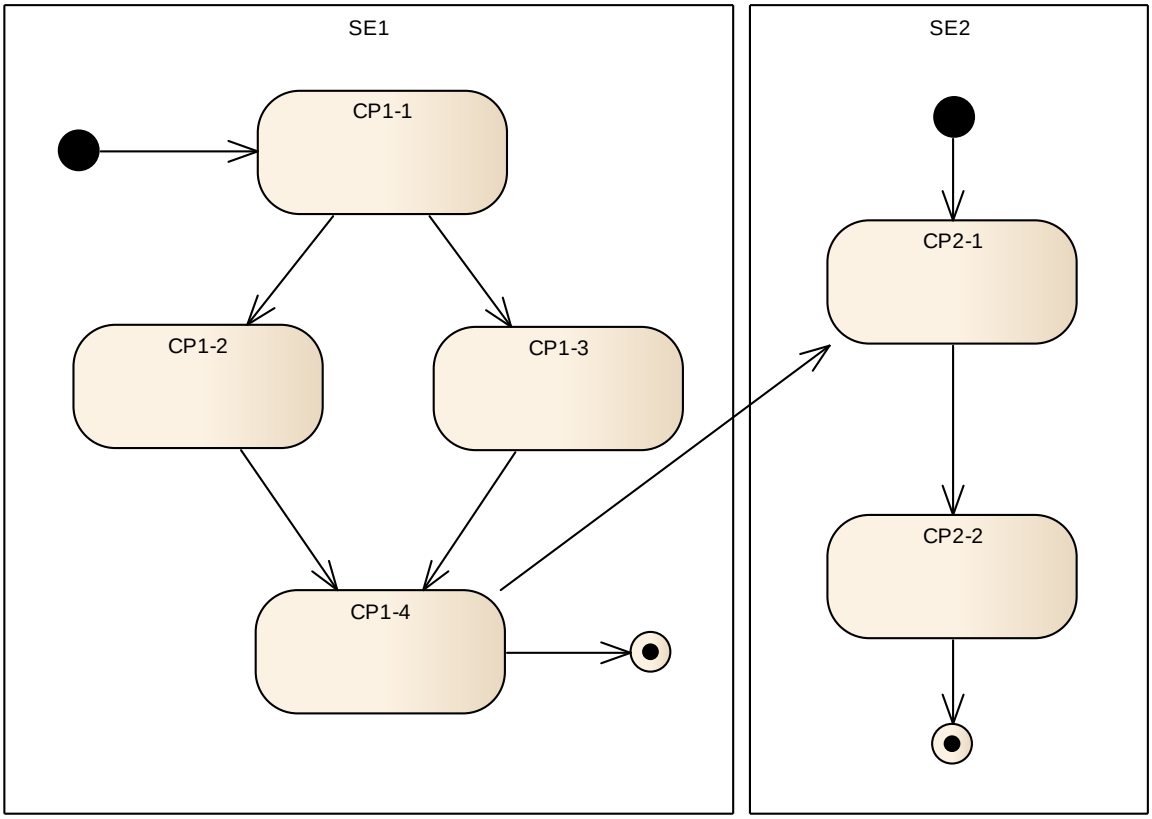

Figure 12: Two Supervised Entities with a External Transition

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 `WdgMMInitialMode` [ECUC_WdgM_00336] which shall be activated when it is initialized.⌋()

The external Graphs cannot overlap.

[SWS_WdgM_00310]⌈The Watchdog manager shall reject configurations where in the same mode a Checkpoint belongs to more than one *External Graph*s.⌋()

7.9.2.3 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 Monitoring 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 `WdgMFailedSupervisionRefCycleTol` [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]).

7.9.2.4 Deadline Supervision

[SWS_WdgM_00313]⌈The Watchdog Manager configuration generator shall reject configurations where the Deadline Supervision (WdgMDeadlineSupervision) of a Supervised Entity refer to Checkpoints (WdgMDeadlineStartRef, WdgMDeadlineEndRef) that does not both belong to that Supervised Entity (i.e. both referred Checkpoints shall belong to the Supervised Entity).⌋()

[SWS_WdgM_00314]⌈The Watchdog Manager configuration generator shall reject configurations where for an ordered set of two Checkpoints there are more than one Deadline Supervision (WdgMDeadlineSupervision) defined.⌋()

7.10 Switching Modes

7.10.1 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.⌋()

7.10.2 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]).

7.10.3 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.

7.11 Specification of the Ports and Port 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 should call the API functions directly and taking into account the mapping by RTE.

All the following interface definitions are interpreted to be in:

```
ARPackage AUTOSAR/Services/WdgM
```

7.11.1 Ports and Port Interface for Alive Supervision

7.11.1.1 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*.

7.11.1.2 Data Types

The parameters passed between the application and the service are:

1. ID to identify a *Supervised Entity* and
2. ID to indentify 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_00371]⌜

```
ImplementationDataType WdgM_SupervisedEntityIdType {
     CATEGORY = TYPE_REFERENCE;
implementationDataType = uint16;
};⌟()
```

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_00372]⌜

```
ImplementationDataType WdgM_CheckpointIdType {
     CATEGORY = TYPE_REFERENCE;
implementationDataType = uint16;
};⌟()
```

7.11.1.3 Port Interface for Alive Supervision

All operations are put into one single interface, in order to minimize the number of ports and names needed in the XML description.

Thus we will have the following operations which match the APIs defined within this specification:

[SWS_WdgM_00333]⌜

```
ClientServerInterface WdgM_AliveSupervision {
   isService = true;

   PossibleErrors {
             E_OK = 0
             E_NOT_OK
   };


   /* Functions for a specific SEID. */
   CheckpointReached(IN WdgM_CheckpointIdType CheckpointID, ERR{E_OK, E_NOT_OK});
   UpdateAliveCounter(ERR{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.」()

7.11.1.4 Service Ports

Figure 13 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.

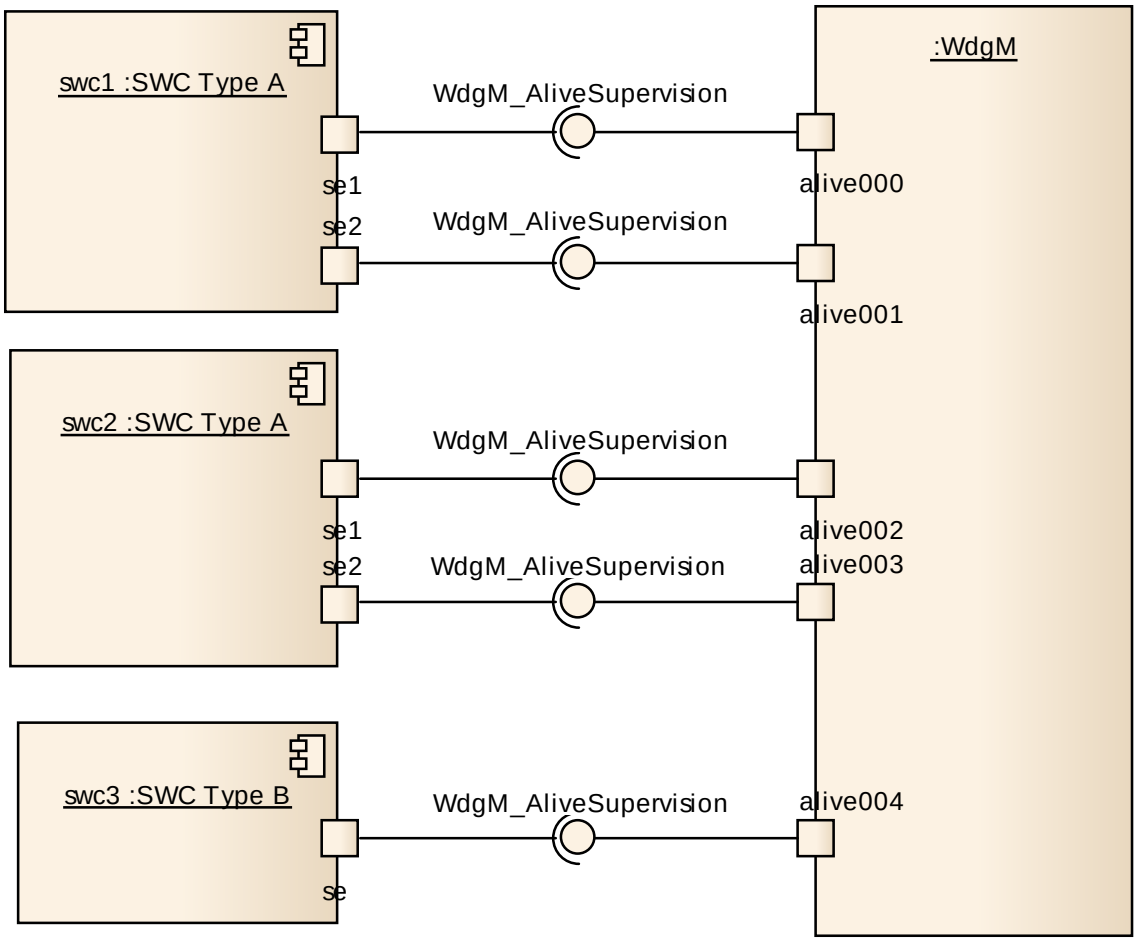

Figure 13: Example of SW-Cs connected to the Watchdog Manager via service ports

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 *Alive Supervision* for each *Supervised Entity* that is configured.

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

[SWS_WdgM_00147]⌈The *Alive Supervision* ports of the Watchdog Manager module shall be named `alive000`, `alive001`, to `alive<#SE-1>`. The numbers shall start with 0 and be consecutive until the number of configured *Supervised Entities* is reached.⌋()

7.11.1.5 Error Codes

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

7.11.2 Ports and Port Interface for Status Reporting

7.11.2.1 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.

[SWS_WdgM_00195]⌈The mode ports of the Watchdog Manager module shall declare the following modes:

```
STATUS_OK
STATUS_FAILED
STATUS_EXPIRED
STATUS_STOPPED
STATUS_DEACTIVATED
```

This definition corresponds to the type WdgM_LocalStatusType.⌋()

[SWS_WdgM_00196]⌈The Watchdog Manager module shall notify SW-Cs / CDDs through the RTE mode ports when the state change occurs.

It is an implementation choice whether to use the Direct or the Indirect RTE API for this notification.

Via the Direct API the implementation must invoke the generated API for individual *Supervised Entities*

```
Rte_StatusType Rte_Switch_mode<SEID>_currentMode(
Rte_ModeType_WdgMMode mode)
```

and for the global state

```
Rte_StatusType Rte_Switch_globalMode_currentMode(
Rte_ModeType_WdgMMode mode)
```

⌋()

where `mode` is the new mode to be notified. The value range is specified by the previous requirement. The return value can be ignored.

Using the indirect port API as shown in chapter *7.11.2.3* Port Interfaces can result in less code when reporting the state to individual *Supervised Entities* and can therefore be used alternatively to the above API.

[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.⌋()

[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 individual 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 individual 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*.⌋()

7.11.2.2 Data Types

The mode declaration group `WdgMMode` represents the modes of the Watchdog Manager module that will be notified to the SW-Cs / CDDs and the RTE. The definition of this mode corresponds to the type WdgM_LocalStatusType.

[SWS_WdgM_00334]⌈

```
ModeDeclarationGroup WdgMMode {
     { SUPERVISION_OK,
       SUPERVISION_FAILED,
       SUPERVISION_EXPIRED,
       SUPERVISION_STOPPED,
       SUPERVISION_DEACTIVATED
     }
     initialMode = SUPERVISION_OK
};⌋()
```

7.11.2.3 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_IndividualMode` is used to signal the *Local Supervision Status* of a single *Supervised Entity*.

[SWS_WdgM_00335]⌈

```
ModeSwitchInterface WdgM_IndividualMode {
     isService = true;
     WdgMMode currentMode;
};
```

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

[SWS_WdgM_00336]⌈

```
ModeSwitchInterface WdgM_GlobalMode {
     isService = true;
     WdgMMode currentMode;
};
```

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_IndividualMode 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_IndividualMode_P ph =
                          Rte_Ports_WdgM_IndividualMode_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_IndividualMode.」()

7.11.2.4 Mode Ports

Figure 14 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*.

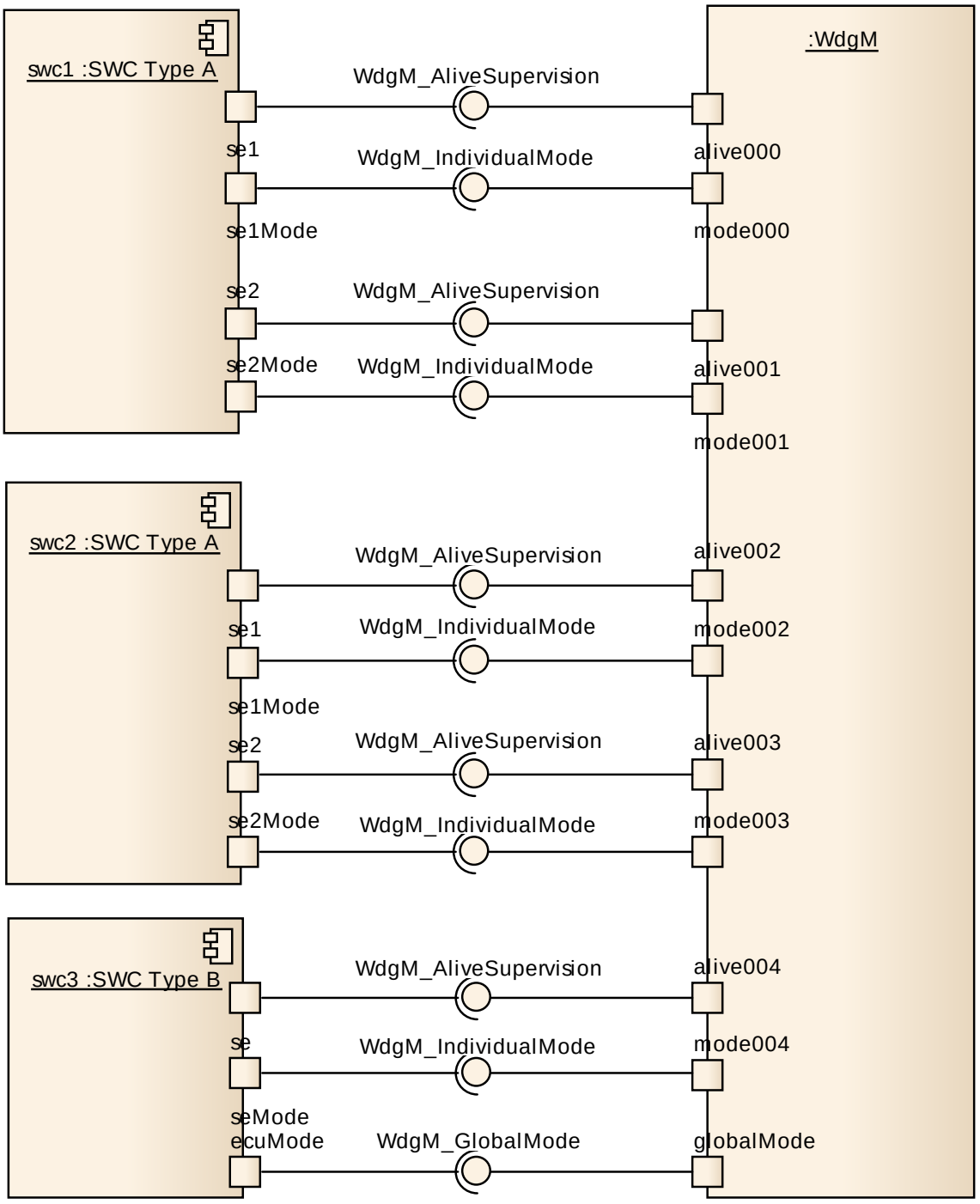

Figure 14: Example of SW-Cs connected to the Watchdog Manager via service ports and mode ports

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 an Alive Supervision port with its corresponding mode port for Status Reporting, a naming convention is necessary.」(SRS_ModeMgm_09160, SRS_ModeMgm_09225)

[SWS_WdgM_00149]「The Watchdog Manager module's single mode ports for reporting the Supervision Status of each *Supervised Entity* shall be named `mode000`, `mode001`, to `mode<#SE-1>`. The numbers shall start with 0 and be consecutive until the number of configured *Supervised Entities* is reached.

Furthermore, the Watchdog Manager module must be able to report the *Global Supervision Status*.」()

[SWS_WdgM_00150]「The Watchdog Manager module shall provide one mode port for reporting the *Global Supervision Status*.」(SRS_ModeMgm_09160, SRS_ModeMgm_09225, SRS_ModeMgm_09162)

7.11.2.5 Error Codes

Mode ports are not able to signal any errors.
`Internal Behavior`
First of all, the runnable entities of a service shall be specified within the "Internal Behavior" description. Runnable entities relevant for the service description are API's of a basic software module realizing the service which are accessed by application software components. The following description results out of that:

```
// Runnable entities of the Watchdog Manager
RunnableEntity SetMode
     symbol "WdgM_SetMode"
     canbeInvokedConcurrently = FALSE

RunnableEntity GetMode
     symbol "WdgM_GetMode"
     canbeInvokedConcurrently = FALSE


RunnableEntity CheckpointReached
     symbol "WdgM_CheckpointReached"
     canbeInvokedConcurrently = TRUE

RunnableEntity UpdateAliveCounter
     symbol "WdgM_UpdateAliveCounter"
     canbeInvokedConcurrently = TRUE
RunnableEntity GetLocalStatus
     symbol "WdgM_GetLocalStatus"
     canbeInvokedConcurrently = TRUE

RunnableEntity GetGlobalStatus
```

```
     symbol "WdgM_GetGlobalStatus"
     canbeInvokedConcurrently = TRUE

RunnableEntity PerformReset
     symbol "WdgM_PerformReset"
     canbeInvokedConcurrently = TRUE

RunnableEntity GetFirstExpiredSEID
     symbol "WdgM_GetFirstExpiredSEID"
     canbeInvokedConcurrently = TRUE
```

Then the Internal Behavior defines the port-defined argument values for the Alive Supervision ports:

```
PortArgument{port= alive000, value.type=
WdgM_SupervisedEntityIdType, value.value=0};

PortArgument{port= alive001, value.type=
WdgM_SupervisedEntityIdType, value.value=1};

…

PortArgument{port= alive<#SE-1>, value.type=
WdgM_SupervisedEntityIdType, value.value=<#SE-1>};
```

And finally the Internal Behavior instructs the RTE to generate additional APIs to indirectly access the mode ports for Status Reporting:

```
IndirectAPI{port= mode000};

IndirectAPI{port= mode001};

…

IndirectAPI{port= mode<#SE-1>};
```

7.11.3 Definition of the Watchdog Manager Service

This section shows the an example of a complete definition of the Watchdog Manager Service. Note that these definitions can only be completed during ECU configuration (because it depends on certain configuration parameters of the Watchdog Manager module which determine the number of ports provided by the Watchdog Manager Service). Also note that the implementation of a SW-C/CDD does *not* depend on these definitions.
There are ports on both sides of the RTE: This description of the Watchdog Manager Service defines the ports below the RTE. Each SW-C/CDD that uses the Service, must contain "service ports" in its own SW-C/CDD description which will be connected to the ports of the Watchdog Manager module, so that the RTE can be generated.

[SWS_WdgM_00338]⌜

```
/** This is the definition of the Watchdog Manager as a
  * service. This is the outside view of the Watchdog Manager,
  * which must be visible to the SW-Cs/CDDs / ECU integrator
  **/
Service WdgM {
// For each supervised entity the Watchdog Manager
// provides a port to update the alive counter
ProvidePort WdgM_AliveSupervision alive000;
…
ProvidePort WdgM_AliveSupervision alive<#SE-1>;

// For each supervised entity the Watchdog Manager
// provides a mode port to signal the Local
// Supervision Status to interested SW-Cs/CDDs and the RTE
ProvidePort WdgM_IndividualMode mode000;
…
ProvidePort WdgM_IndividualMode mode<#SE-1>;

// The Watchdog Manager also provides a single mode port
// to signal the Global Supervision Status to
//      interested      SW-Cs/CDDs      and      the      RTE
ProvidePort WdgM_GlobalMode globalMode;

InternalBehavior
{
     // Runnable entities of the Watchdog Manager
          RunnableEntity UpdateAliveCounter
               symbol “WdgM_UpdateAliveCounter”
canbeInvokedConcurrently = TRUE

          RunnableEntity CheckpointReached
               symbol “WdgM_CheckpointReached”
canbeInvokedConcurrently = TRUE



     PortArgument{port=          alive000,          value.type=
WdgM_SupervisedEntityIdType, value.value=0};

     PortArgument{port=          alive001,          value.type=
WdgM_SupervisedEntityIdType, value.value=1};

…

PortArgument{port=          alive<#SE-1>,          value.type=
WdgM_SupervisedEntityIdType, value.value=<#SE-1>};
```

```
IndirectAPI{port= mode000};

IndirectAPI{port= mode001};

…

IndirectAPI{port= mode<#SE-1>};
};
};
```

」()

8 API Specification

8.1 Imported Types

[SWS_WdgM_00011]⌈The Watchdog Manager module shall use only the following imported types of other modules:

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
Os	ApplicationType
	CounterType
	ISRType
	RestartType
	StatusType
	TaskType
	TickRefType
Std_Types	Std_ReturnType
	Std_VersionInfoType
WdgIf	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]WdgM_ConfigType

⌈

Name:	`WdgM_ConfigType`	
Type:	`Structure`	
Range:	–	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.	

⌋()

[SWS_WdgM_00042]⌈The structure `WdgM_ConfigType` shall contain all post-build configurable parameters of the Watchdog Manager module. The exact content of this structure depends on the selected configuration variant.

See Chapter 10.2 for information on configuration parameters.」(SRS_ModeMgm_09106)

8.2.2 WdgM_SupervisedEntityIdType

[SWS_WdgM_00356]WdgM_SupervisedEntityIdType

⌜

Name:	WdgM_SupervisedEntityIdType		
Type:	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.		

⌟()

8.2.3 WdgM_CheckpointIdType

[SWS_WdgM_00357]WdgM_CheckpointIdType

⌜

Name:	WdgM_CheckpointIdType		
Type:	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.		

⌟()

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

8.2.4 WdgM_ModeType

[SWS_WdgM_00358]WdgM_ModeType

⌜

Name:	WdgM_ModeType		
Type:	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.		

⌟()

8.2.5 WdgM_LocalStatusType

[SWS_WdgM_00359]WdgM_LocalStatusType

⌜

Name:	WdgM_LocalStatusType		
Type:	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.		

⌋()

8.2.6 WdgM_GlobalStatusType

[SWS_WdgM_00360]WdgM_GlobalStatusType

⌈

Name:	WdgM_GlobalStatusType		
Type:	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.		

⌋()

8.3 Function Definitions

8.3.1 WdgM_Init

[**SWS_WdgM_00151]**WdgM_Init

⌜

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.	

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.⌟(SRS_BSW_00310, SRS_BSW_00358, SRS_ModeMgm_09107)

[SWS_WdgM_00018]⌜The function `WdgM_Init` shall initialize all module variables (global and static) of the Watchdog Manager module.⌟(SRS_ModeMgm_09107)

[SWS_WdgM_00135]⌜The function `WdgM_Init` shall establish the initial mode of the Watchdog Manager module.

The behavior in case the initial mode cannot be established is described in SWS_WdgM_00139.

There are optional checks that are executed if and only if `WdgMDevErrorDetect` is enabled.⌟(SRS_ModeMgm_09107)

[SWS_WdgM_00255]⌜If the `WdgMDevErrorDetect [ECUC_WdgM_00301]` switch is enabled and the configuration variant is VARIANT-POST-BUILD, the function `WdgM_Init` shall check if a NULL pointer is passed for the `ConfigPtr` parameter. In case of an error the remaining function `WdgM_Init` shall not be executed and the function `WdgM_Init` shall report development error code `WDGM_E_INV_POINTER` to the `Det_ReportError` service of the Development Error Tracer.⌟(SRS_BSW_00323)

[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 development error code WDGM_E_PARAM_CONFIG to the Det_ReportError service of the Development Error Tracer.⌋(SRS_BSW_00323, SRS_BSW_00338)

[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 not execute the initialization routine and if the WdgMDevErrorDetect switch is enabled, the function WdgM_Init shall report development error code WDGM_E_DISABLE_NOT_ALLOWED to the Det_ReportError service of the Development Error Tracer.⌋(SRS_BSW_00323, SRS_BSW_00338, SRS_ModeMgm_09109)

[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 *8.3.11* for more information.⌋()

8.3.2 WdgM_DeInit

[SWS_WdgM_00261]WdgM_DeInit
⌈

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.

⌋(SRS_BSW_00310, SRS_BSW_00336)

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 check if the Watchdog Manager is initialized. In case of an error, the function `WdgM_DeInit` shall return without any effect and shall report the error to the Development Error Tracer with the error code `WDGM_E_NO_INIT`.⌋(SRS_BSW_00323)

8.3.3 WdgM_GetVersionInfo

[SWS_WdgM_00153]WdgM_GetVersionInfo
⌈

Service name:	WdgM_GetVersionInfo
Syntax:	`void WdgM_GetVersionInfo( Std_VersionInfoType* VersionInfo )`
Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	VersionInfo: Pointer to where to store the version information of the module WdgM.
Return value:	None
Description:	Returns the version information of this module.

⌋(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 Development Error Tracer.⌋(SRS_BSW_00323)

8.3.4 WdgM_SetMode

[SWS_WdgM_00154]WdgM_SetMode

⌜

Service name:	WdgM_SetMode	
Syntax:	`Std_ReturnType WdgM_SetMode( WdgM_ModeType Mode, uint16 CallerID )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Mode	One of the configured Watchdog Manager modes.
	CallerID	Module ID of the calling module.
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.	

⌟(SRS_BSW_00310, SRS_ModeMgm_09110)

The behavior of this service and the corresponding functional requirements are described in chapter 7.10.

[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 and the error is not a defined development error [SWS_WdgM_00004], the Watchdog Manager shall report to the Diagnostic Event Manager an error with the value `WDGM_E_SET_MODE.`⌟(SRS_BSW_00339)

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, the error shall be reported to the Development Error Tracer with the value `WDGM_E_PARAM_MODE` and the routine shall return the value `E_NOT_OK.`⌟(SRS_BSW_00323, SRS_BSW_00338)

[SWS_WdgM_00021]⌈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 mode switch shall not be executed, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_NO_INIT` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323, SRS_BSW_00338, SRS_BSW_00406)

[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, the mode switch shall not be executed, and if If the configuration parameter `WdgMDevErrorDetect` is enabled, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_DISABLE_NOT_ALLOWED` and the routine shall return the value `E_NOT_OK`. ⌋(SRS_BSW_00323, SRS_BSW_00338, SRS_ModeMgm_09109)

[SWS_WdgM_00245]⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00352] switch is enabled, the Watchdog Manager shall check if the given CallerID is in the list of allowed CallerIDs [ECUC_WdgM_00358]. If it is not, then the service shall return without any effect, it shall return the value E_NOT_OK, and shall report the error status `WDGM_E_IMPROPER_CALLER` to the DEM.⌋()

8.3.5 WdgM_GetMode

[SWS_WdgM_00168]WdgM_GetMode

⌈

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.	

⌋(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 routine shall check if the Watchdog Manager is initialized. In case of an error the service shall not be executed, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_NO_INIT` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323)

[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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_INV_POINTER` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323)

8.3.6 WdgM_CheckpointReached

[SWS_WdgM_00263]WdgM_CheckpointReached
⌈

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_ReturnType	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.	

⌋(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 for the reported Supervised Entity using the reported Checkpoint. The output shall be an updated result of Deadline Supervision for the Supervised Entity.⌋()

[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.⌋()

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

[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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_PARAM_SEID` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323)

[SWS_WdgM_00279]⌈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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_NO_INIT` and the routine shall return the value `E_NOT_OK`⌋(SRS_BSW_00323)

[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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_CPID` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323)

[SWS_WdgM_00319]⌈If the configuration parameter `WdgMDevErrorDetect` [ECUC_WdgM_00301] is enabled, 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 not be executed, the error shall be reported to the Development Error Tracer

with the error code WDGM_E_SEDEACTIVATED and the routine shall return the value E_NOT_OK.⌋()

8.3.7 WdgM_UpdateAliveCounter

[SWS_WdgM_00155]WdgM_UpdateAliveCounter

⌈

Service name:	WdgM_UpdateAliveCounter	
Syntax:	`Std_ReturnType WdgM_UpdateAliveCounter( WdgM_SupervisedEntityIdType SEID )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SEID	Identifier of the entity under control of the WdgM whose alive counter shall be updated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Successfully updated alive counter E_NOT_OK: Update failed
Description:	BEWARE, this API is deprecated. Gives alive indications to the Watchdog Manager.	

This function is deprecated and should not be used anymore. It is only provided for backward compatibility. Use WdgM_CheckpointReached instead!

If the function WdgM_UpdateAliveCounter is used, then there shall be at most one Alive Supervision configured for a Supervised Entity for a given mode. By this means, at runtime the function WdgM_CheckpointReached is able to identify the Checkpoint from the Supervised Entity ID. If more than one are configured, the function returns an error code to DET.⌋(SRS_BSW_00310, SRS_ModeMgm_09125)

[SWS_WdgM_00318]⌈This function shall call the function WdgM_CheckpointReached and shall provide as parameter the Checkpoint of the Alive Supervision of the current mode.⌋()

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

[SWS_WdgM_00320]⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled, the routine shall check if in the current mode there are more than one Alive Supervisions (WdgMAliveSupervision) configured. If so, then

the routine shall report the error code WDGM_E_AMBIGIOUS to the Development Error Tracer.⌋()

[SWS_WdgM_00027]⌈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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_PARAM_SEID` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323, SRS_BSW_00338)

[SWS_WdgM_00028]⌈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, the error shall be reported to the Development Error Tracer with the error code WDGM_E_NO_INIT and the routine shall return the value E_NOT_OK.⌋(SRS_BSW_00323, SRS_BSW_00338, SRS_BSW_00406)

[SWS_WdgM_00290]⌈If the configuration parameter WdgMDevErrorDetect [ECUC_WdgM_00301] is enabled, the function shall report the error code WDGM_E_DEPRECATED to the Development Error Tracer.⌋()

8.3.8 WdgM_GetLocalStatus

[SWS_WdgM_00169]WdgM_GetLocalStatus
⌈

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_ReturnType	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.	

⌋(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, the error shall be reported to the Development Error Tracer with the error code WDGM_E_PARAM_SEID and the routine shall return the value E_NOT_OK.⌋(SRS_BSW_00323)

[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, the error shall be reported to the Development Error Tracer with the error code WDGM_E_INV_POINTER and the routine shall return the value E_NOT_OK.⌋(SRS_BSW_00323)

[SWS_WdgM_00173]⌈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, the error shall be reported to the Development Error Tracer with the error code WDGM_E_NO_INIT and the routine shall return the value E_NOT_OK.⌋(SRS_BSW_00323)

8.3.9 WdgM_GetGlobalStatus

[SWS_WdgM_00175]WdgM_GetGlobalStatus

⌈

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 monitoring 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.	

⌋(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) and return.⌋()

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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_INV_POINTER` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323)

[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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_NO_INIT` and the routine shall return the value `E_NOT_OK`.⌋(SRS_BSW_00323)

8.3.10 WdgM_PerformReset

[SWS_WdgM_00264]WdgM_PerformReset

⌈

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.

⌋(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 routine shall check if the Watchdog Manager is initialized. In case of an error the service shall not be executed, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_NO_INIT`.⌋(SRS_BSW_00323)

8.3.11 WdgM_GetFirstExpiredSEID

[SWS_WdgM_00346]WdgM_GetFirstExpiredSEID

⌈

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.	

⌋()

[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) and return.⌋()

[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]WdgM_MainFunction

⌈

Service name:	WdgM_MainFunction

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

Service ID[hex]:	0x08
Description:	Performs the processing of the cyclic Watchdog Manager jobs.

⌋(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_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 first 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_00327]⌈Based on the Local Supervision status of each Supervision Status and the Global Supervision Status, the function `WdgM_MainFunction()` shall manage the corresponding error handling.⌋()

[SWS_WdgM_00328]⌈Based on the Global Supervision Status, the function `WdgM_MainFunction()` shall call set correspondingly the trigger condition of Watchdog Interface modules.⌋()

[SWS_WdgM_00063]⌈If the *Global Supervision Status* is not in the state WDGM_GLOBAL_STATUS_DEACTIVATED, then the `WdgM_MainFunction()` shall be executed according to the configured *Supervision Cycle* (see WdgMSupervisionCycle [ECUC_WdgM_00330]).⌋(SRS_ModeMgm_09112)

If a *Supervised Entity* finishes in a deadlock and does not exit, it could be that the watchdog manager main function is not called and therefore they do not detect the failed supervised entity. Therefore the tasks containing the main function shall be separated from the tasks containing *Supervised Entities* that are supervised by the Watchdog Manager Module.

[SWS_WdgM_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.⌋()

[SWS_WdgM_00039]⌈If the configuration parameter `WdgMDefensiveBehavior` [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, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_NO_INIT`.⌋(SRS_BSW_00323, SRS_BSW_00338, SRS_BSW_00406)

8.6 Expected Interfaces

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

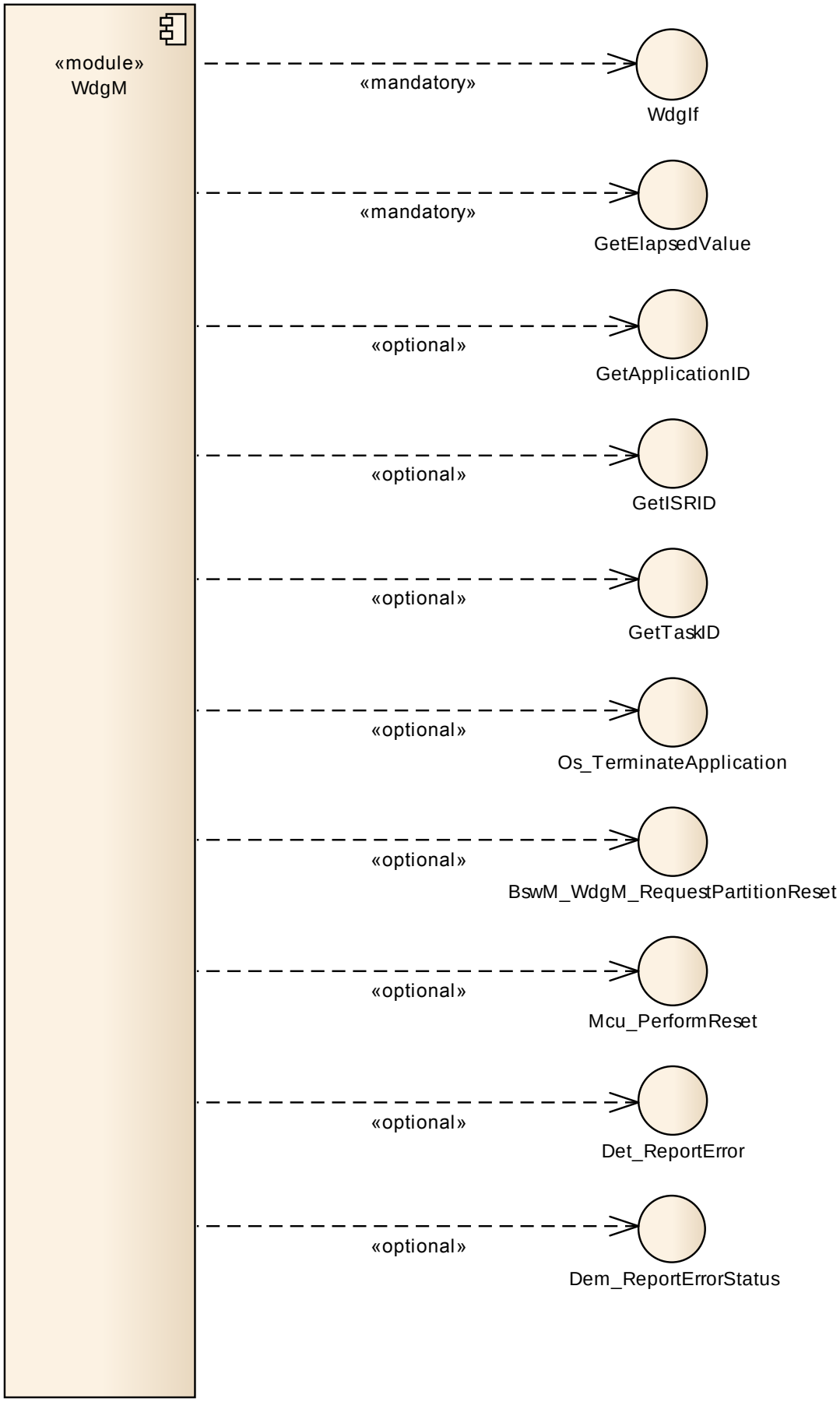
«module»
WdgM
«mandatory»
WdgIf
«mandatory»
GetElapsedValue
«optional»
GetApplicationID
«optional»
GetISRID
«optional»
GetTaskID
«optional»
Os_TerminateApplication
«optional»
BswM_WdgM_RequestPartitionReset
«optional»
Mcu_PerformReset
«optional»
Det_ReportError
«optional»
Dem_ReportErrorStatus

Figure 15: Expected Interfaces

8.6.1 Mandatory Interfaces

This chapter defines a superset of interfaces which are required to fulfill the core functionality of the module.

[SWS_WdgM_00161]

⌈

API function	*Description*
GetElapsedValue	This service gets the number of ticks between the current tick value and a previously read tick value.
WdgIf_SetMode	Map the service WdgIf_SetMode to the service Wdg_SetMode of the corresponding Watchdog Driver.
WdgIf_SetTriggerCondition	Map the service WdgIf_SetTriggerCondition to the service Wdg_SetTriggerCondition 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	*Description*
BswM_WdgM_RequestPartitionReset	Function called by WdgM to request a partition reset.
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function. OBD Events Suppression shall be ignored for this computation.
Det_ReportError	Service to report development errors.
GetApplicationID	This service determines the currently running OS-Application (a unique identifier has to be allocated to each application).
GetISRID	This service returns the identifier of the currently executing ISR.
GetTaskID	GetTaskID returns the information about the TaskID of the task which is currently running.
Mcu_PerformReset	The service performs a microcontroller reset.
TerminateApplication	This service terminates the OS-Application to which the calling Task/Category 2 ISR/application specific error hook belongs.

⌋()

8.6.3 Configurable Interfaces

Not Applicable

8.6.4 Job End Notification

Not Applicable

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.

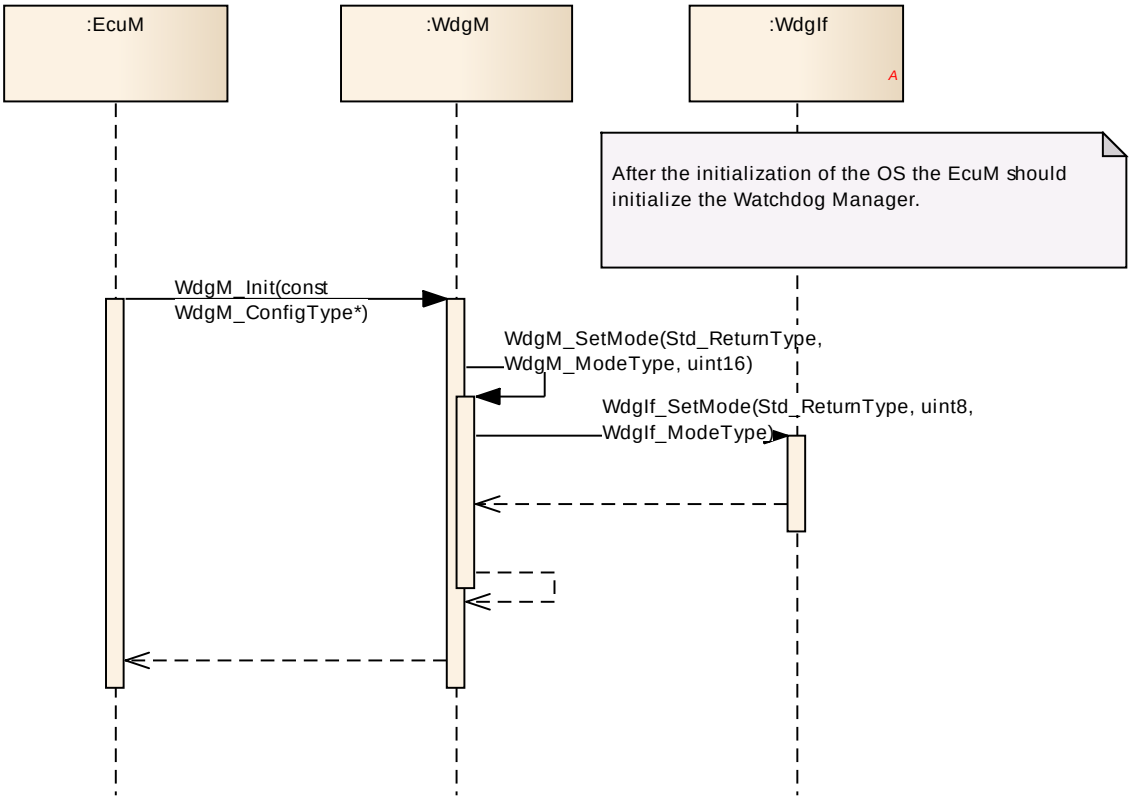

Figure 16: Initialization of the Watchdog Manager module

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) shall be located in the module’s configuration header file WdgM_Cfg.h.⌋(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 shall be located in an external data structure of type WdgM_ConfigType. The type declaration shall be located in the file WdgM.h．⌋()

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 sourcecode-file of the Watchdog Manager module according to settings of static configuration. They should be located at the module's configuration header file WdgM_Cfg.h⌋(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*
Module Description	Configuration of the WdgM (Watchdog Manager) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMConfigSet	1	This container describes one of multiple configuration sets of WdgM. This is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
WdgMGeneral	1	Container defines all general configuration parameters of the Watchdog Manager.

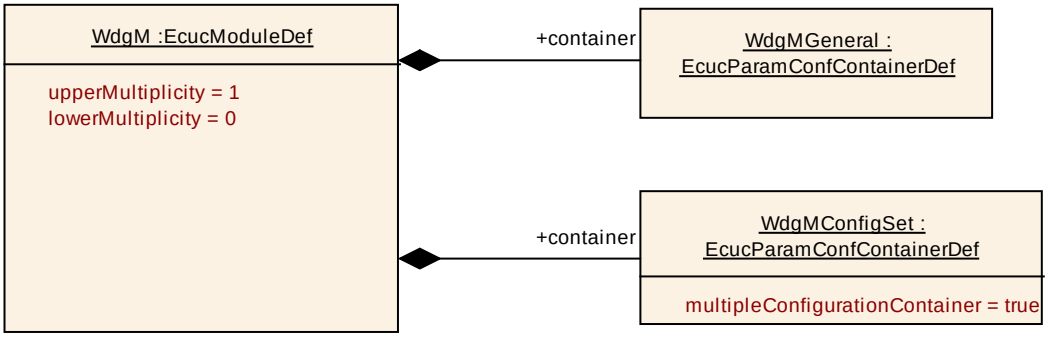

Figure 17: Configuration Module WdgM

10.2.3 WdgMGeneral

SWS Item	**ECUC_WdgM_00300 :**
Container Name	WdgMGeneral
Description	Container defines all general configuration parameters of the Watchdog Manager.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00352 :**		
Name	WdgMDefensiveBehavior {WDGM_DEFENSIVE_BEHAVIOR}		
Description	Preprocessor switch to enable/disable the defensive behavior of the Watchdog Manager module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00338 :**		
Name	WdgMDemStoppedSupervisionReport {WDGM_DEM_ALIVE_SUPERVISION_REPORT}		
Description	Parameter to enable/disable the error reporting to DEM. true: A notification to DEM is sent if the Watchdog Manager reaches the state WDGM_GLOBAL_STATUS_STOPPED. false: The notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00301 :**		
Name	WdgMDevErrorDetect {WDGM_DEV_ERROR_DETECT}		
Description	Preprocessor switch to enable/disable development error detection and reporting. Shall be used to remove unneeded code segments regarding DET features true: Development error detection is enabled false: Development error detection is disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00339 :**		
Name	WdgMImmediateReset {WDGM_IMMEDIATE_RESET}		
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	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00340 :**		
Name	WdgMOffModeEnabled {WDGM_OFF_MODE_ENABLED}		
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	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	**ECUC_WdgM_00302 :**		
Name	WdgMVersionInfoApi {WDGM_VERSION_INFO_API}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMCallerIds	0..1	Contains the definition of valid CallerIds for the callers who have permission to call the function WdgM_SetMode.
WdgMSupervisedEntity	0..65535	This container collects all common (mode-independent) parameters of a Supervised Entity to be supervised by the Watchdog Manager.
WdgMWatchdog	0..255	This container collects all common (mode-independent) parameters of a Watchdog to be triggered by the Watchdog Manager.

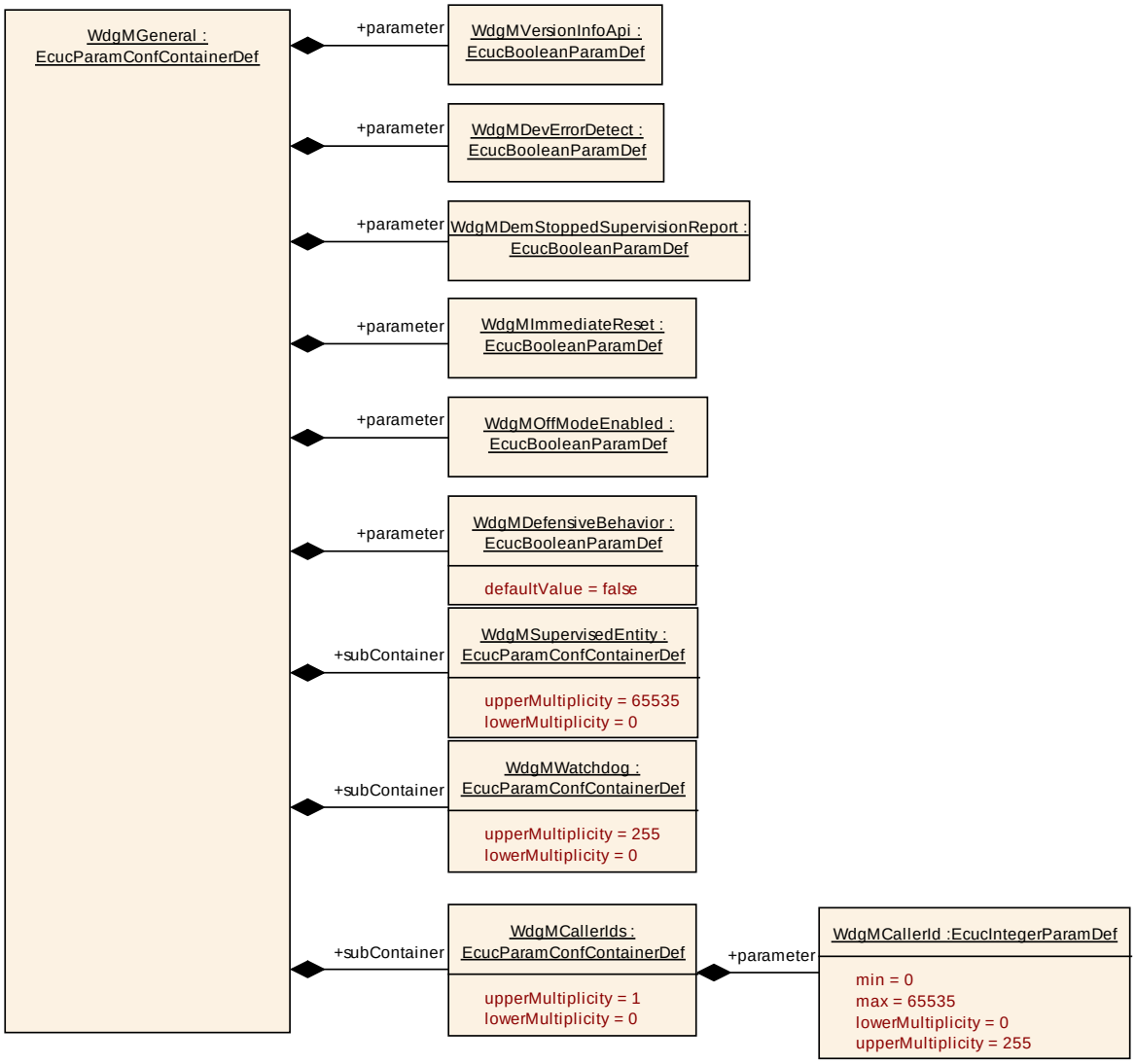

Figure 18: Configuration Container WdgMGeneral

10.2.4 WdgMSupervisedEntity

SWS Item	**ECUC_WdgM_00303 :**
Container Name	WdgMSupervisedEntity{WdgMSupervisedEntity}
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 :**		
Name	WdgMSupervisedEntityId {WDGM_SUPERVISED_ENTITY_ID}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

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

SWS Item	**ECUC_WdgM_00360 :**		
Name	WdgMEcucPartitionRef		
Description	Denotes in which "EcucPartition" the supervised entity is executed. When the partition is stopped, the supervised entity shall be de-activated in the WdgM to avoid an ECU reset.		
Multiplicity	0..1		
Type	Reference to [EcucPartition]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00343 :**		
Name	WdgMInternalCheckpointInitialRef		
Description	This is the reference to the initial Checkpoint for this Supervised Entity.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00344 :**		
Name	WdgMInternallCheckpointFinalRef		
Description	This is the reference to the final Checkpoint(s) for this Supervised Entity.		
Multiplicity	1..65535		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00361 :**		
Name	WdgMOSCounter		
Description	OS counter used by Watchdog Manager to perform the deadline monitoring of the Supervised Entity.		
Multiplicity	0..1		
Type	Reference to [OsCounter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUC_WdgM_00346 :**		
Name	WdgMOsApplicationRef		
Description	Optional reference to an OS Application. Beware, the Watchdog Manager module will trigger a partition restart of this OS Application when the corresponding Supervised Entity reaches WDGM_LOCAL_STATUS_FAILED.		
Multiplicity	0..1		
Type	Reference to [OsApplication]		
ConfigurationClass	***Pre-compile time***	X	All Variants

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

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.
WdgMInternalTransition	0..65535	This container defines the graph of Internal Transitions within this Supervised Entity.

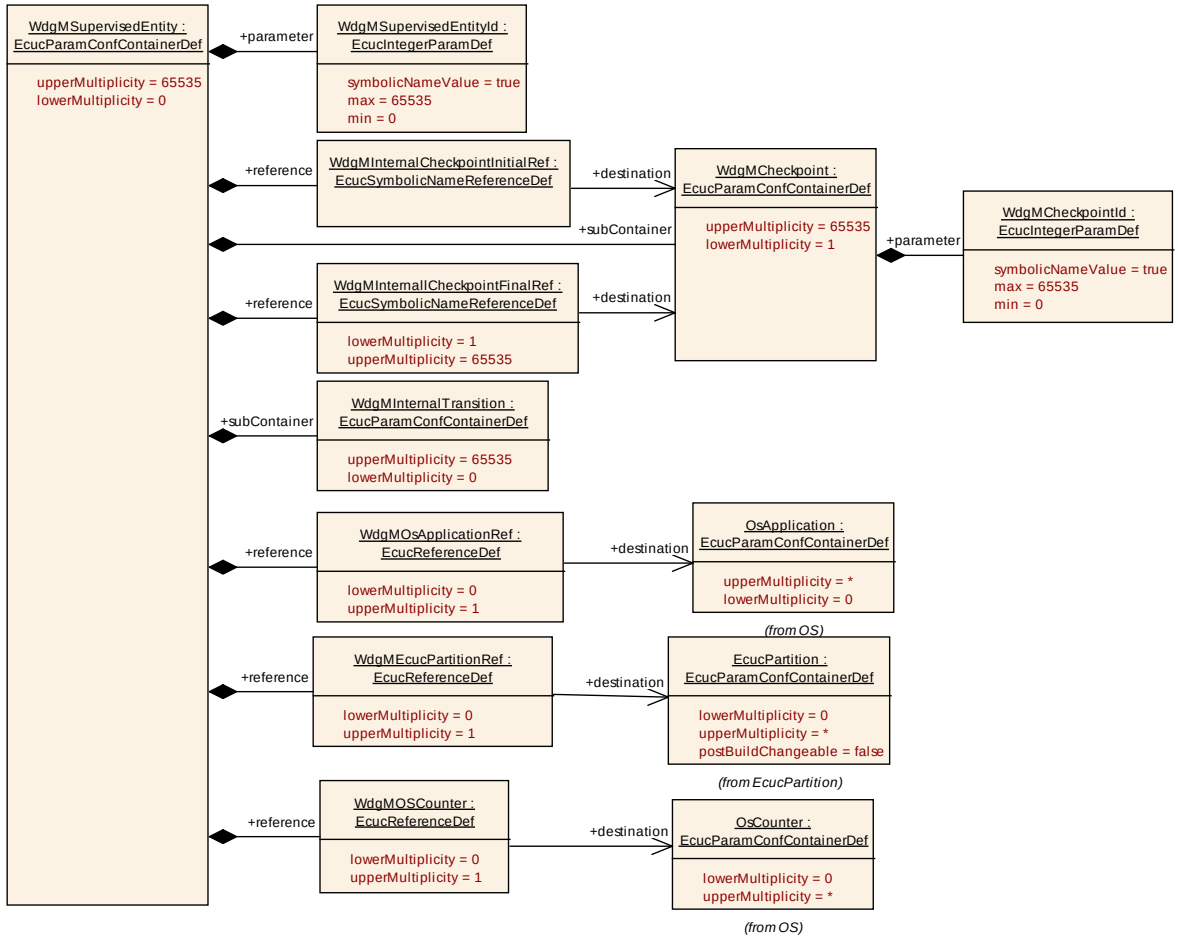

Figure 19: Configuration Container WdgMSupervisedEntity

10.2.5 WdgMCheckpoint

SWS Item	**ECUC_WdgM_00305 :**
Container Name	WdgMCheckpoint{WdgMCheckpoint}
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 :**
Name	WdgMCheckpointId {WdgMCheckPointId}

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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

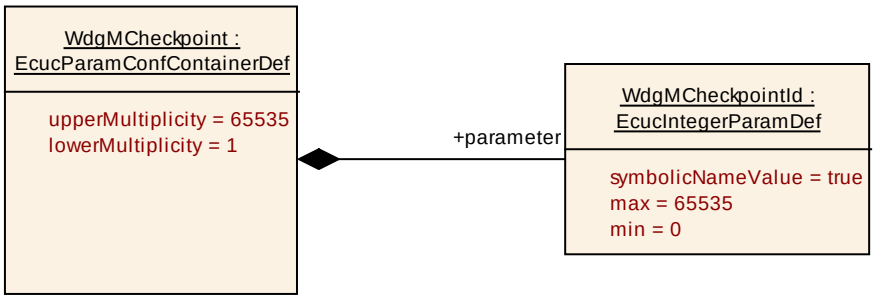

Figure 20: Configuration Container WdgMCheckpoint

10.2.6 WdgMInternalTransition

SWS Item	**ECUC_WdgM_00345 :**
Container Name	WdgMInternalTransition
Description	This container defines the graph of Internal Transitions within this Supervised Entity.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00351 :**		
Name	WdgMInternalTransitionDestRef		
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]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00350 :**
Name	WdgMInternalTransitionSourceRef
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]

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

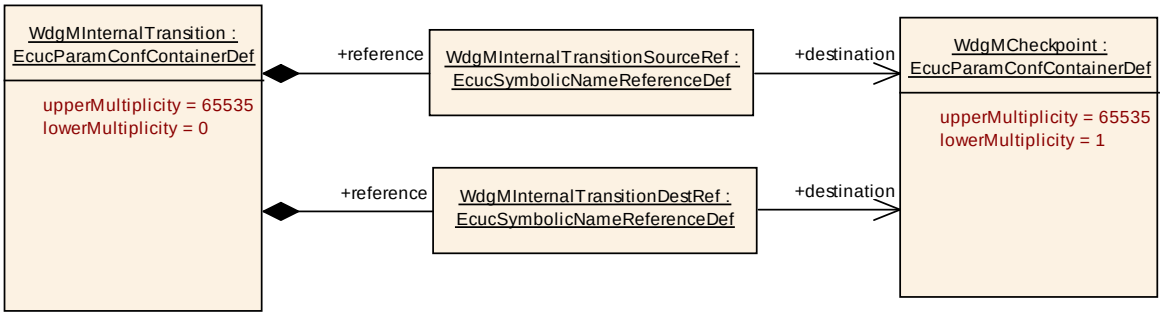

Figure 21: Configuration Container WdgMInternalTransition

10.2.7 WdgMWatchdog

SWS Item	**ECUC_WdgM_00347 :**
Container Name	WdgMWatchdog
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 :**		
Name	WdgMWatchdogName {WDGM_WATCHDOG_INSTANCE_ID}		
Description	This parameter shall contain the symbolic name of the watchdog instance.		
Multiplicity	1		
Type	EcucStringParamDef (Symbolic Name generated for this parameter)		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00349 :**		
Name	WdgMWatchdogDeviceRef		
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]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

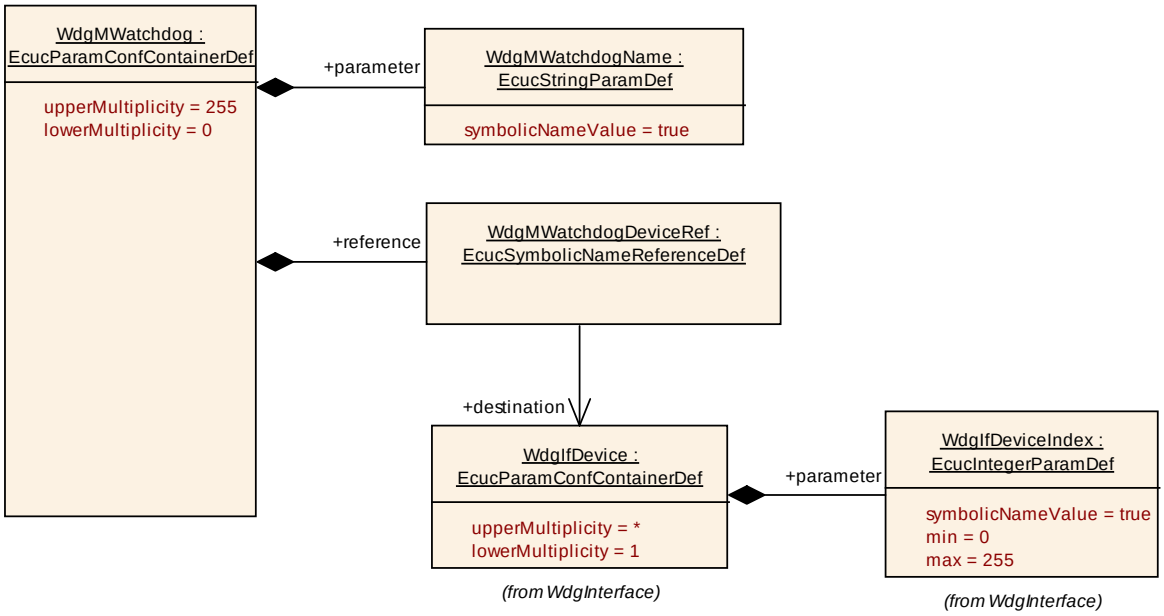

Figure 22: Configuration Container WdgMWatchdog

10.2.8 WdgMConfigSet

SWS Item	**ECUC_WdgM_00337 :**
Container Name	WdgMConfigSet [Multi Config Container]
Description	This container describes one of multiple configuration sets of WdgM. This is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00336 :**		
Name	WdgMInitialMode		
Description	The mode that the Watchdog Manager is in after it has been initialized.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMMode]		
ConfigurationClass	***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***
WdgMDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The

		standardized errors are provided in the container and can be extended by vendor specific error references.
WdgMMode	1..255	The container describes one of several modes of the Watchdog Manager.

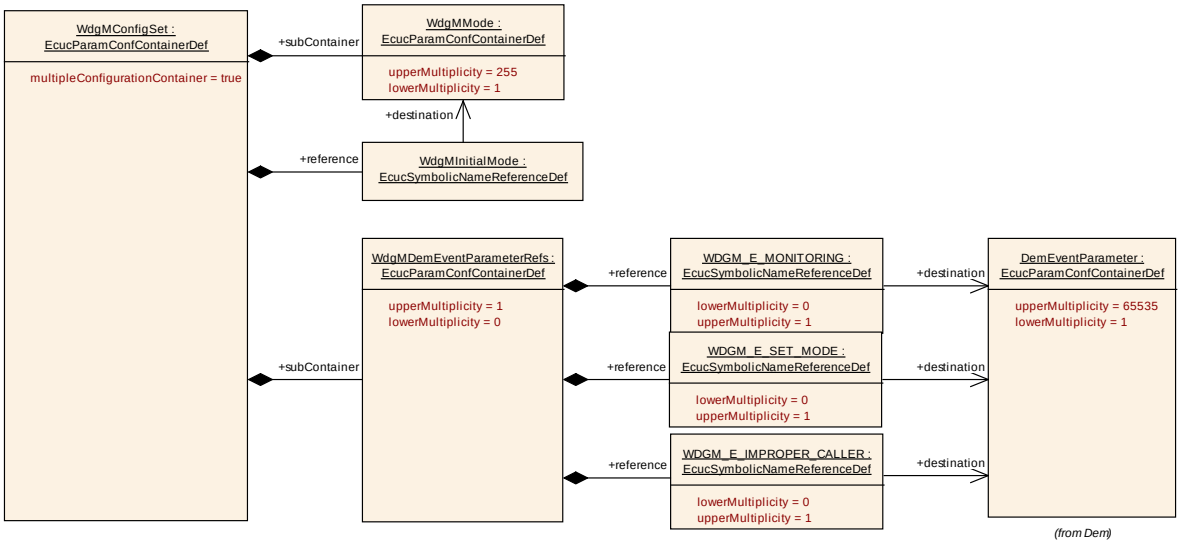

Figure 23: Configuration Container WdgMConfigSet

10.2.9 WdgMDemEventParameterRefs

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

SWS Item	**ECUC_WdgM_00357 :**		
Name	WDGM_E_IMPROPER_CALLER		
Description	Reference to the DemEventParameter which shall be issued when the defensive behavior checks have detected an improper caller.		
Multiplicity	0..1		
Type	Symbolic name reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00354 :**
Name	WDGM_E_MONITORING
Description	Reference to the DemEventParameter which shall be issued when the error "Monitoring has failed and a watchdog reset will occur" has occurred.
Multiplicity	0..1

Type	Symbolic name reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00355 :**		
Name	WDGM_E_SET_MODE		
Description	Reference to the DemEventParameter which shall be issued when the error "Watchdog drivers' mode switch has failed" has occurred.		
Multiplicity	0..1		
Type	Symbolic name reference to [DemEventParameter]		
ConfigurationClass	***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{WDGM_MODE}
Description	The container describes one of several modes of the Watchdog Manager.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00329 :**		
Name	WdgMExpiredSupervisionCycleTol {WDGM_EXPIRED_SUPERVISION_CYCLE_TOLERANCE}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_WdgM_00307 :**		
Name	WdgMModeId		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_WdgM_00330 :**		
Name	WdgMSupervisionCycle {WDGM_SUPERVISION_CYCLE}		
Description	This parameter defines the schedule period of the main function WdgM_MainFunction. Unit: [s]		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMAliveSupervision	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.
WdgMDeadlineSupervision	0..65535	This container collects all configuration parameters for Deadline Supervision for a Supervised Entity.
WdgMExternalLogicalSupervision	0..65535	This container collects all configuration parameters for Logical Supervision for one external graph.
WdgMLocalStatusParams	0..65535	This container collects all configuration parameters for the Local Status of a Supervised Entity.
WdgMTrigger	0..255	This container collects all configuration parameters for the triggering of hardware watchdogs.

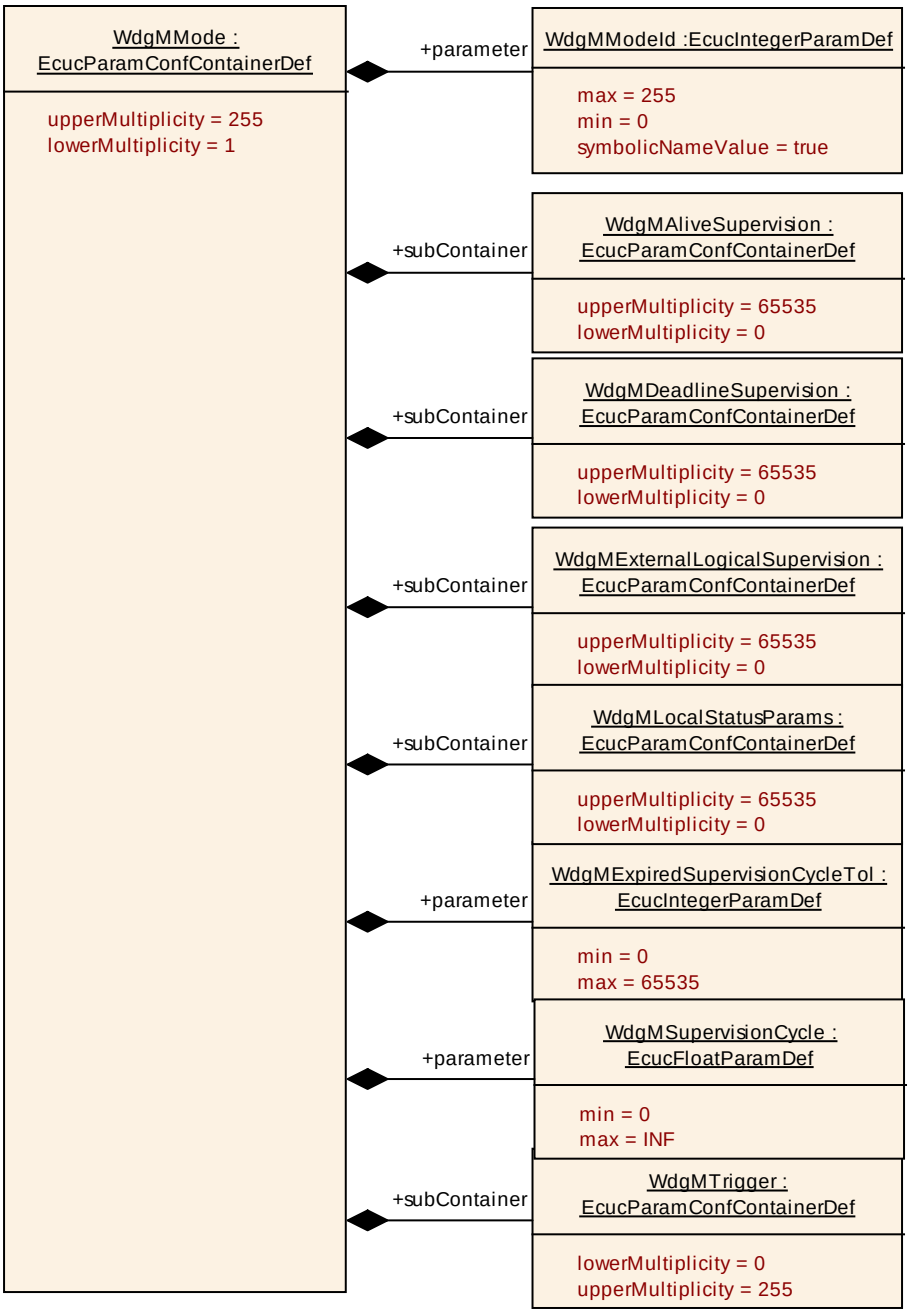

Figure 24: Configuration Container WdgMMode

10.2.11 WdgMAliveSupervision

SWS Item	**ECUC_WdgM_00308 :**
Container Name	WdgMAliveSupervision{WdgMAliveSupervision}
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 :**		
Name	WdgMExpectedAliveIndications {WDGM_EXPECTED_ALIVE_INDICATIONS}		
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	--		
ConfigurationClass	***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 :**		
Name	WdgMMaxMargin {WDGM_MAX_MARGIN}		
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	--		
ConfigurationClass	***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 :**		
Name	WdgMMinMargin {WDGM_MIN_MARGIN}		
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	--		
ConfigurationClass	***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 :**		
Name	WdgMSupervisionReferenceCycle {WDGM_SUPERVISION_REFERENCE_CYCLE}		
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	--		
ConfigurationClass	***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 :**		
Name	WdgMAliveSupervisionCheckpointRef		
Description	Reference to Checkpoint within a Supervised Entity that shall be supervised.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

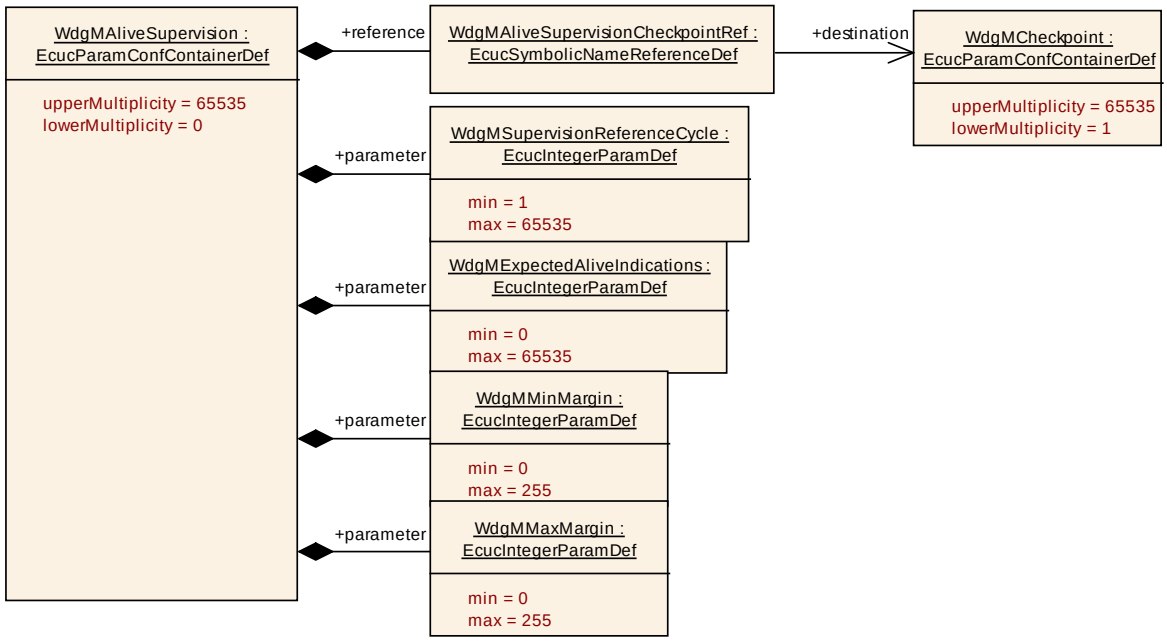

Figure 25: Configuration Container WdgMAliveSupervision

10.2.12 WdgMDeadlineSupervision

SWS Item	**ECUC_WdgM_00314 :**
Container Name	WdgMDeadlineSupervision
Description	This container collects all configuration parameters for Deadline Supervision for a Supervised Entity.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00318 :**
Name	WdgMDeadlineMax
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	--		
ConfigurationClass	***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 :**		
Name	WdgMDeadlineMin		
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	--		
ConfigurationClass	***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 :**		
Name	WdgMDeadlineStartRef		
Description	This is the reference to the start Checkpoint for Deadline Supervision.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***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 :**		
Name	WdgMDeadlineStopRef		
Description	This is the reference to the stop Checkpoint for Deadline Supervision.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

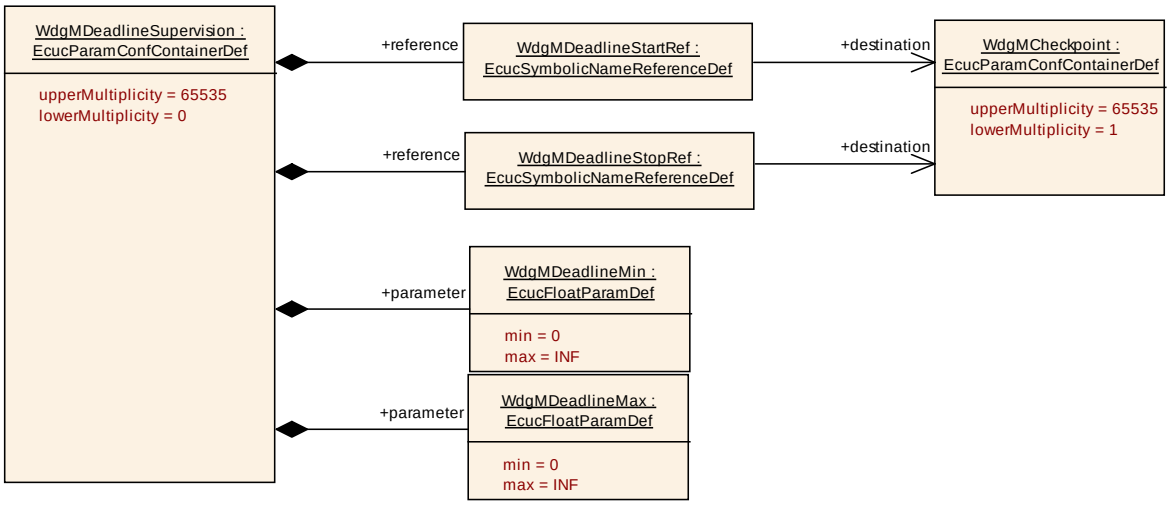

Figure 26: Configuration Container WdgMDeadlineSupervision

10.2.13 WdgMExternalLogicalSupervision

SWS Item	**ECUC_WdgM_00319 :**
Container Name	WdgMExternalLogicalSupervision
Description	This container collects all configuration parameters for Logical Supervision for one external graph.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00324 :**		
Name	WdgMExternalCheckpointFinalRef		
Description	This is the reference to the final Checkpoint(s) for this External Graph.		
Multiplicity	1..65535		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***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 :**		
Name	WdgMExternalCheckpointInitialRef		
Description	This is the reference to the initial Checkpoint(s) for this External Graph.		
Multiplicity	1..65535		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***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***
WdgMExternalTransition	0..65535	This container collects the Checkpoints for an External Transition across Supervised Entities.

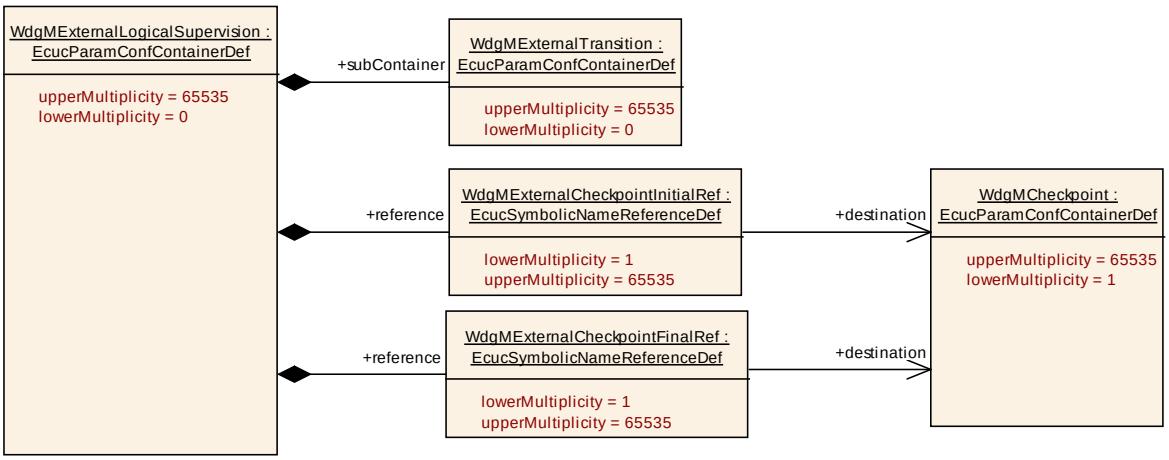

Figure 27: Configuration Container WdgMExternalLogicalSupervision

10.2.14 WdgMExternalTransition

SWS Item	**ECUC_WdgM_00320 :**
Container Name	WdgMExternalTransition
Description	This container collects the Checkpoints for an External Transition across Supervised Entities.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00322 :**		
Name	WdgMExternalTransitionDestRef		
Description	This is the reference to the destination Checkpoint of an External Transition.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***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 :**		
Name	WdgMExternalTransitionSourceRef		
Description	This is the reference to the source Checkpoint of an External Transition.		
Multiplicity	1		
Type	Symbolic name reference to [WdgMCheckpoint]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

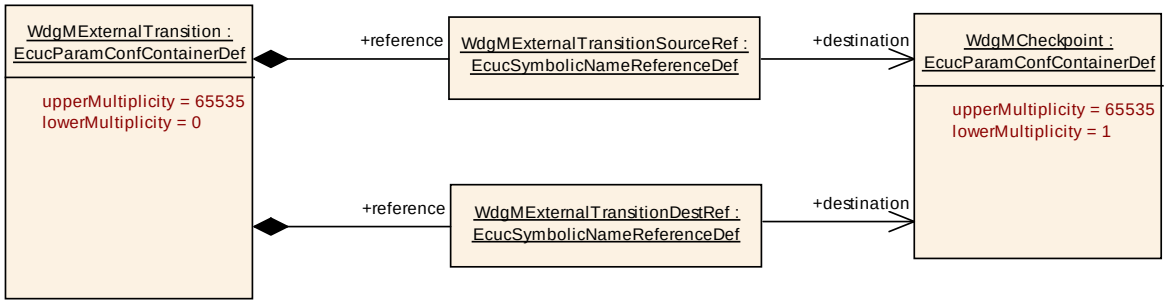

Figure 28: Configuration Container WdgMExternalTransition

10.2.15 WdgMTrigger

SWS Item	**ECUC_WdgM_00331 :**
Container Name	WdgMTrigger{WdgMTrigger}
Description	This container collects all configuration parameters for the triggering of hardware watchdogs.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00333 :**		
Name	WdgMTriggerConditionValue		
Description	This parameter shall contain the value that is passed to WdgIf_SetTriggerCondition for this watchdog.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
ConfigurationClass	***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 :**		
Name	WdgMWatchdogMode		
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	--	
ConfigurationClass	***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 :**
Name	WdgMTriggerWatchdogRef
Description	This parameter is a reference to the configured watchdog.
Multiplicity	1

Type	Symbolic name reference to [WdgMWatchdog]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

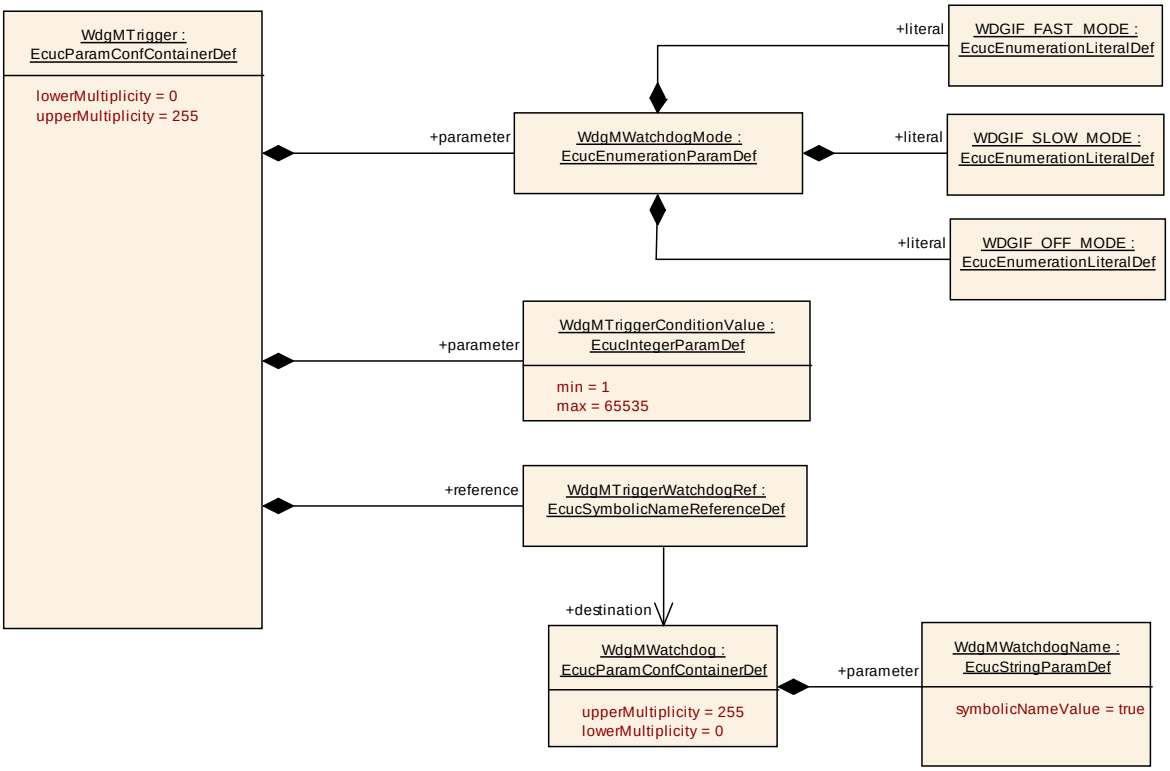

Figure 29: Configuration Container WdgMTrigger

10.2.16 WdgMLocalStatusParams

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

SWS Item	**ECUC_WdgM_00327 :**		
Name	WdgMFailedAliveSupervisionRefCycleTol {WDGM_FAILED_SUPERVISION_REFERENCE_CYCLE_TOLERANCE}		
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	--		
ConfigurationClass	***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 :**		
Name	WdgMLocalStatusSupervisedEntityRef		
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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

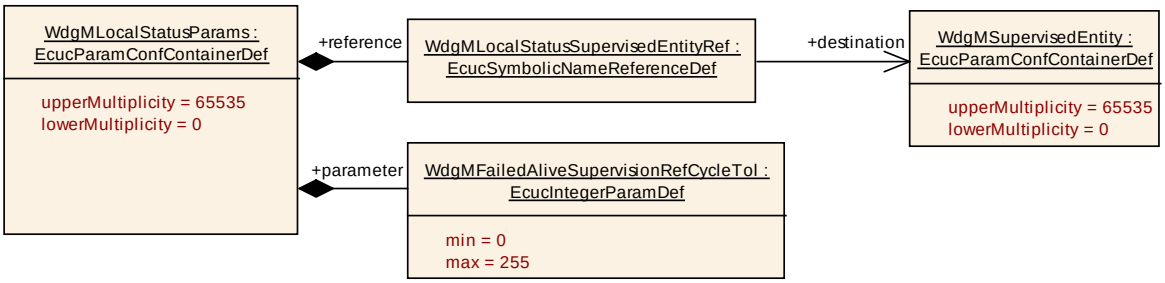

Figure 30: Configuration Container WdgMLocalStatusParams

10.2.17 WdgMCallerIds

SWS Item	**ECUC_WdgM_00358 :**
Container Name	WdgMCallerIds
Description	Contains the definition of valid CallerIds for the callers who have permission to call the function WdgM_SetMode.
Configuration Parameters	

SWS Item	**ECUC_WdgM_00359 :**		
Name	WdgMCallerId		
Description	This parameter defines one valid CallerId for the callers who have permission to call the function WdgM_SetMode.		
Multiplicity	0..255		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.3 Published Information

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

10.4 Callback Routines

The Watchdog Manager module follows the standardized AUTOSAR concept to report development errors. The provided callback routines are specified in the Development Error Tracer (DET) specification.

The Watchdog Manager module follows the standardized AUTOSAR concept to report production errors. The provided callback routines are specified in the Diagnostic Event Manager (DEM) specification.

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

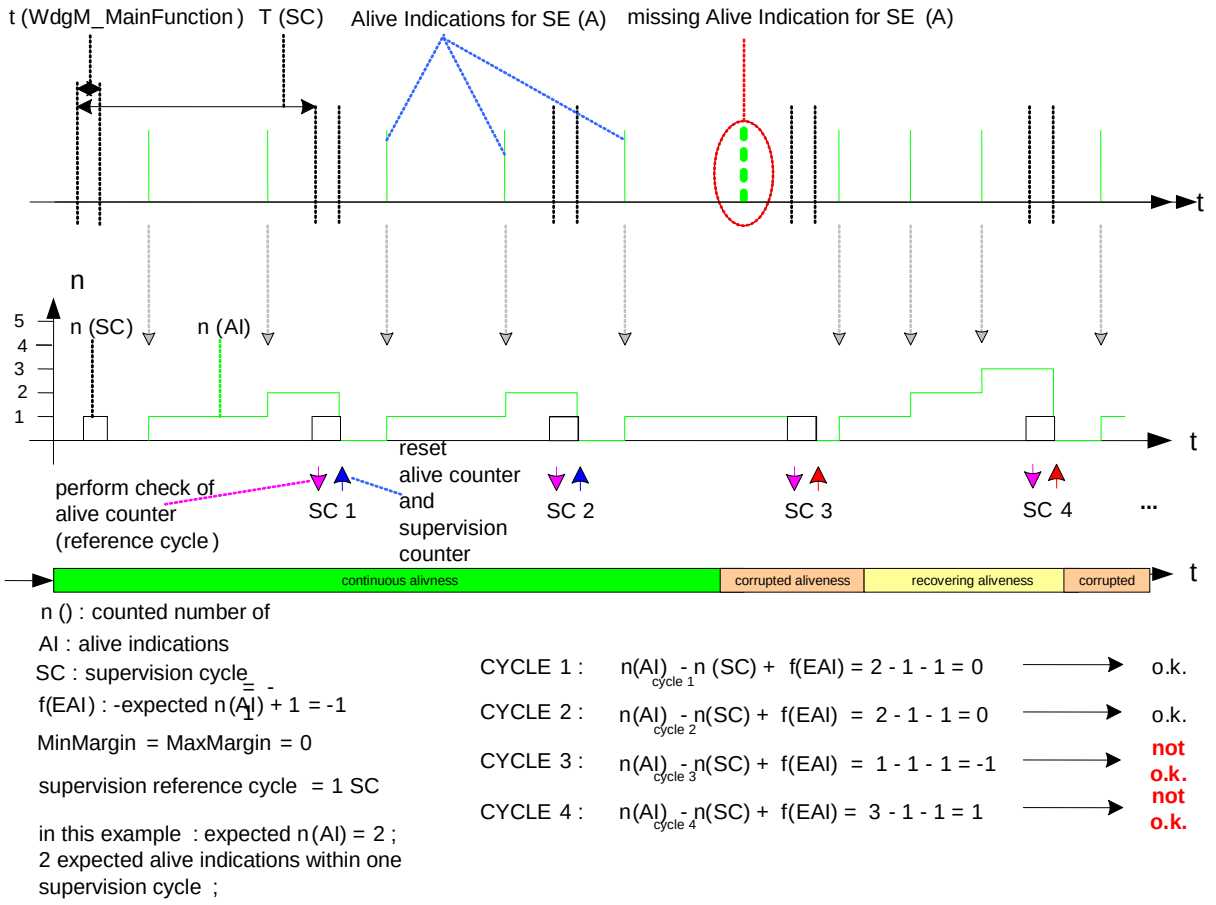

Figure 31: Alive-supervision algorithm – Scenario A

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

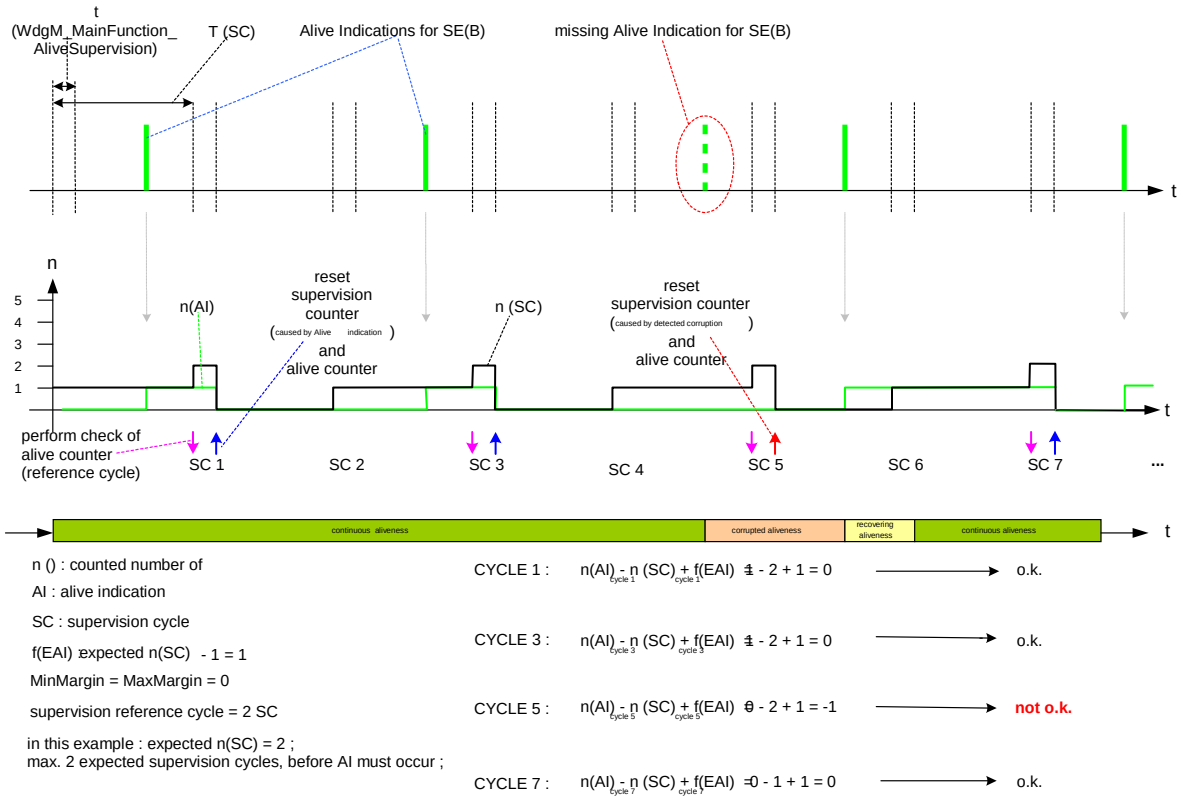

Figure 32: Alive Supervision algorithm – Scenario B

12 Not applicable requirements

[SWS_WdgM_00345]⌈These requirements are not applicable to this specification.⌋(SRS_BSW_00300, SRS_BSW_00304, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00326, SRS_BSW_00328, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00422, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00344, SRS_BSW_00347, SRS_BSW_00355, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00440, SRS_BSW_00370, SRS_BSW_00371, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00386, SRS_BSW_00387, SRS_BSW_00398, SRS_BSW_00405, SRS_BSW_00413, SRS_BSW_00416, SRS_BSW_00437, SRS_BSW_00417, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, BSW00431, SRS_BSW_00432, SRS_BSW_00433, BSW00434, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00439, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00160, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172)