Document Title	Specification of Watchdog Manager
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	080
Document Classification	Standard

Document Version	1.2.0
Document Status	Final
Part of Release	3.0
Revision	0001

Document Change History

Date	Version	Changed by	Change Description
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 Mode • Document meta information extended • Small layout adaptations made
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

Page left intentionally blank

Disclaimer

Advice to users of AUTOSAR Specification Documents:

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.
It is intended to supervise the reliability of applications execution in consideration of periodicity and maximum timing constraints of periodicity. Derived from the approach of the layered architecture, a decoupling of application timing constraints from hardware watchdog timing constraints becomes possible. Based on this decoupling the Watchdog Manager provides alive-supervision of application, abstracted from the triggering of hardware watchdog entities.

The Watchdog Manager specification provides a flexible approach to serve different patterns of ensuring execution reliability. The adaptation of usage to individual project demands is controlled by configurable parameters of the Watchdog Manager.

1.1 Concepts of Watchdog Manager Usage

The Watchdog Manager is always based on the decoupled alive-supervision of application and the triggering of hardware watchdog via the Watchdog Interface and Watchdog Driver(s). A difference in concept to proof the execution reliability could be achieved by different usage of the alive-supervision mechanism of the Watchdog Manager. The main use cases that represent a differentiated perspective on execution reliability will be described in the following paragraphs of this section. This overview shall support system-designers to make up decision about usage or to find the interoperable concept to re-use application software, which have been designed in cooperation of previous watchdog services.

1.1.1 Alive-supervision of multiple entities

The execution reliability of multiple entities could be supervised individually by the Watchdog Manager. Therefore the Watchdog Manager provides an individual port for each supervised application entity, where the entities have to indicate their proof of aliveness (update an alive-counter).
Within the cyclic scheduled main-service of the Watchdog Manager, the alive-counters of all supervised entities are checked against their own independent timing constraints and a "global supervision status" is derived. The Watchdog Manager will initiate the triggering of the hardware watchdog according to this global supervision status.

The set of supervised entities and their individual timing constraints will be defined by configurable parameters of the Watchdog Manager.

Note: Consider that the Watchdog Manager does not care about the placement of supervised entities, so it could be possible that SW-Cs contain none, one or multiple supervised entities. It is assumed, that the Software Component Templates will detail this information for the respective SW-Cs.

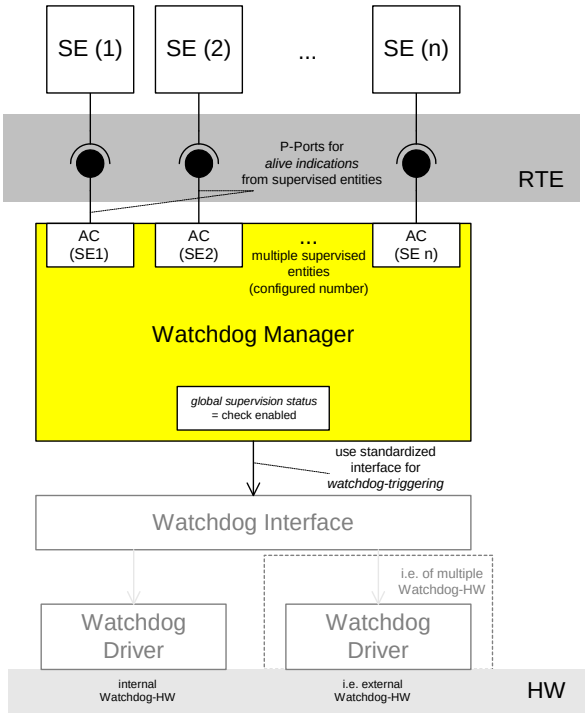

Figure 1: Alive-supervision of multiple entities

1.1.2 Alive-supervision of a single entity

As a subset of alive-supervision of multiple entities, the Watchdog Manager could be used to supervise only one entity. In this case the Watchdog Manager provides an individual port for one entity, where the entity has to indicate its proof of aliveness (update an alive-counter).
Within the cyclic scheduled main-service of the Watchdog Manager, the alive-counter of this supervised entity is checked against its timing constraints and the "global supervision status" is derived as usual. The Watchdog Manager will initiate the triggering of the hardware watchdog according to this global supervision status.

The supervised entity and its individual timing constraints will be defined by configurable parameters of the Watchdog Manager.

Note: From an abstract point of view, this concept handles the supervision of the whole application software as one logical entity. This may support compatibility of existing application software solutions, where the application software has to initiate the watchdog service.

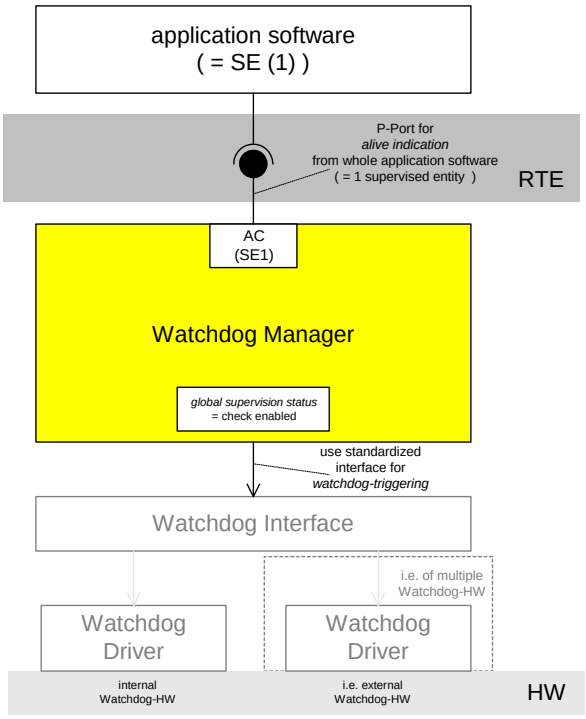

Figure 2: Alive-supervision of a single entity

1.1.3 No alive-supervision

This concept does not proof the reliability of execution by alive-indications from any application entities, but by scheduling the Watchdog Manager cyclically.
Within the cyclic scheduled main-service the Watchdog Manager will initiate the triggering of the hardware watchdog.
This scenario is included in each of the previous mentioned concepts in case that the alive-supervision of all assigned supervised-entities is temporarily disabled. (For details of temporarily disabled supervision refer to chapter 7.1.2.2)

If in general no alive-supervision is required, the alive-supervision mechanism could be completely disabled by a configurable parameter of the Watchdog Manager.

Note: From an abstract point of view, this concept handles the proof of reliable execution by the cyclic execution of the Watchdog Manager, which has to initiate the triggering of the hardware watchdog. If the Watchdog Manager is not scheduled according to the timing constraints of the hardware watchdog, the hardware watchdog will not be treated sufficiently. This may support compatibility of existing software solutions, where an encapsulated trigger-entity was scheduled cyclically with lowest execution priority to service the hardware watchdog.

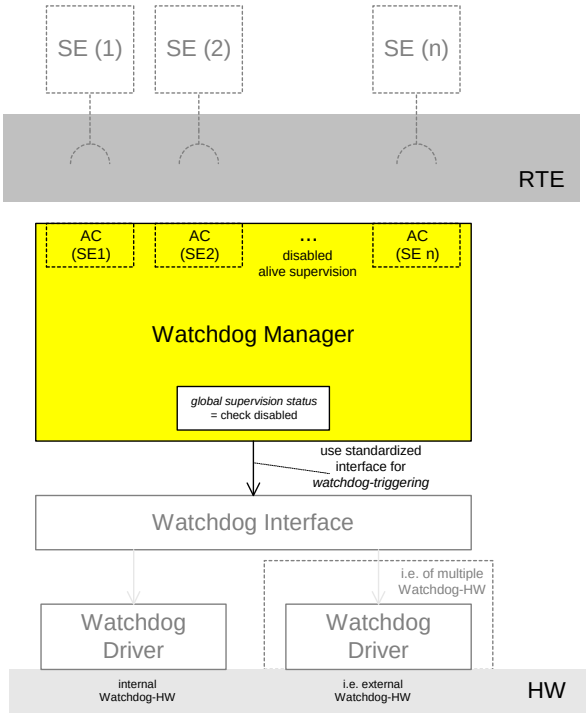

Figure 3: No Alive-supervision

1.1.4 Advantages in scalability of supervision concepts

The scalability of the “supervision concept” inserts flexibility to the Watchdog Manager to service a configurable amount of entities with different timing constraints that need to be supervised in the context of a reliable cyclic execution.
The reliability is especially in focus of well-defined applications, likely safety relevant applications, and therefore the demand for supervision and its timing constraints are associated to the application itself. In focus of AUTOSAR, the basic software architecture shall support portability of applications across platforms, which means services to perform supervision need to be provided by the basic software on each platform. The configurable and therefore scalable concept of the Watchdog Manager supports this portability of such SW-C from one platform to another.

Another advantage is the backward-compatibility with existing application software solutions. The concept could be adapted easily by configurable parameters to the inherited circumstances of the existing solutions.

One precondition for flexibility is given by the layered architecture and the decoupled mechanism of alive-supervision and initiation of triggering, which are performed by the Watchdog Manager.

Note: The Watchdog Manager does not distinguish between different kinds of "users" (e.g. entities within SW-C's or BSW-modules). Therefore all entities share the same supervision context and need a unique ID within this common context. It is recommended to support the declaration and configuration of these unique IDs by a tool chain that shares the whole local ECU context.

1.2 Reliability of Watchdog Manager Mechanisms

Services of Watchdog Manager are designed to ensure an application’s reliability by supervising its cyclic execution. The Watchdog Manager will support mechanisms to restart a "hanging" execution or to trigger a watchdog reset and thus bring the ECU back to normal execution through a complete restart.

2 Acronyms, Abbreviations and Terms

Abbreviation / Acronym	*Description*
AI	Alive Indication
BSW	Basic Software
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
VFB	Virtual Functional Bus
WdgM	Watchdog Manager

Term	*Description*
Activation Status	Individual status of a *Supervised Entity* that indicates whether the alive-supervision of the corresponding entity is activated or deactivated.
Alive Counter	An independent data resource in the Watchdog Manager in context of a *Supervised Entity* to track and handle its amount of *Alive Indications*.
Alive Indication	An indication provided by a *Supervised Entity* to signal its aliveness to the Watchdog Manager.
Alive Indication Point	A certain point within the *Supervised Entity*, where the *Alive Indication* shall be forwarded to the Watchdog Manager, when this point is passed. There may be one or more *Alive Indication Points* per *Supervised Entity*, but the Watchdog Manager cannot distinguish them.
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*.
Global Supervision Status	Status that represent the current global result of alive-supervision computed by the Watchdog Manager from the *Individual Supervision Status* of all *Supervised Entities*.
Individual Supervision Status	Status that represents the current result of alive-supervision of a single *Supervised Entity*.
Supervised Entity	A software entity which is included in the alive-supervision algorithm of the Watchdog Manager. Each *Supervised Entity* has exactly one identifier. A *Supervised Entity* denotes a checkpoint within a Software Component or Basic Software Module. There

Term	Description
	may be zero, one or more *Supervised Entities* in a Software Component or Basic Software Module. Note that due to the design of the alive-supervision algorithm, each Supervised Entity must exhibit a cyclic timing behavior.
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 algorithm is performed.
Supervision Reference Cycle	The amount of *Supervision Cycles* to be used as reference by the alive-supervision mechanism to perform the check of counted *Alive Indications* (individually for each *Supervised Entity*).
Trigger Cycle	The time period of the Watchdog Manager, where the triggering of watchdog instances is performed.
Trigger Period	The time period of a watchdog instance, when it shall be triggered cyclically.
Trigger Reference Cycle	The amount of *Trigger Cycles* to be used as reference by the Watchdog Manager to perform the triggering of watchdog instances (individually for each watchdog instance).

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_General.pdf

[3] Requirements on Mode Management
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SRS_ModeManagement.pdf

[4] Specification of Platform Types
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_PlatformTypes.pdf

[5] Specification of RTE
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_RTE.pdf

[6] Specification of ECU State Manager
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_ECU_StateManager.pdf

[7] AUTOSAR Basic Software Module Description Template,
https://svn2.autosar.org/repos2/22_Releases
AUTOSAR_BSW_Module_Description.pdf

4 Constraints and assumptions

4.1 Limitations

- The Watchdog Manager doesn't encapsulate the watchdog driver initialization. The watchdog driver initialization will be performed by the ECU State Manager early in the startup process.
- The Watchdog Manager should be initialized after the OS has been started. Hence, it cannot be responsible for triggering the hardware watchdog earlier in the startup process. If watchdog-triggering is needed during early startup phase (before the OS has been started), implementer should use ECU State Manager facilities (callouts).
- The Watchdog Manager should be de-initialized before the OS shutdown. Hence, it cannot be responsible for triggering the hardware watchdog later in the shutdown process. If watchdog-triggering is needed during shutdown phase (after the OS has been shutdown), implementer should use ECU State Manager facilities (callouts).
- For ECU's which implement low consumption modes, if the hardware watchdog remains active, its triggering shall also be handled by the ECU State Manager.
- The alive-supervision of the Watchdog Manager shall only apply to entities with cyclic timing constraints.

4.2 Applicability to car domains

No restriction

5 Dependencies to other modules

- Watchdog Interface (WdgIf)
  The Watchdog Manager is responsible for changing the mode of the watchdog driver and for triggering the hardware watchdog via the watchdog driver. The services of the watchdog driver are accessed via the watchdog interface which enables to address multiple watchdog instances.

- ECU State Manager (EcuM)
  The ECU State Manager is responsible for initializing, de-initializing and changing the mode of the Watchdog Manager.

- Micro Controller Unit Driver (Mcu)
  The Watchdog Manager enables, as an optional feature, to perform an immediate reset of the ECU in case of alive-supervision failure. This reset service is provided by the MCU driver.

- Development Error Tracer (Det)
  If development error detection is enabled, the Watchdog Manager shall inform the Development Error Tracer about detected development errors.

- Diagnostic Event Manager (Dem)
  The Watchdog Manager shall inform 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. The Watchdog Manager shall use the services of the BSW Scheduler to implement critical sections.

- General Purpose Timer (Gpt)
  The General Purpose Timer is used to activate the Watchdog Manager when the BSW Scheduler is not active, i.e. during startup, shutdown and sleep.

- Runtime Environment (Rte)
  The Runtime Environment is responsible for propagating updates of alive counters from supervised entities in SW-Cs to the Watchdog Manager. The Watchdog Manager shall use the services of the Runtime Environment to inform SW-Cs about changes in the alive supervision status.

5.1 File structure

5.1.1 Code file structure

WDGM127: The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- WdgM_PBcfg.c – for post build time configurable parameters.

This file shall contain all post-build time configurable parameters.

5.1.2 Header file structure

WDGM126: The module shall include the Dem.h file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

WDGM014: The file include structure shall be as follows:

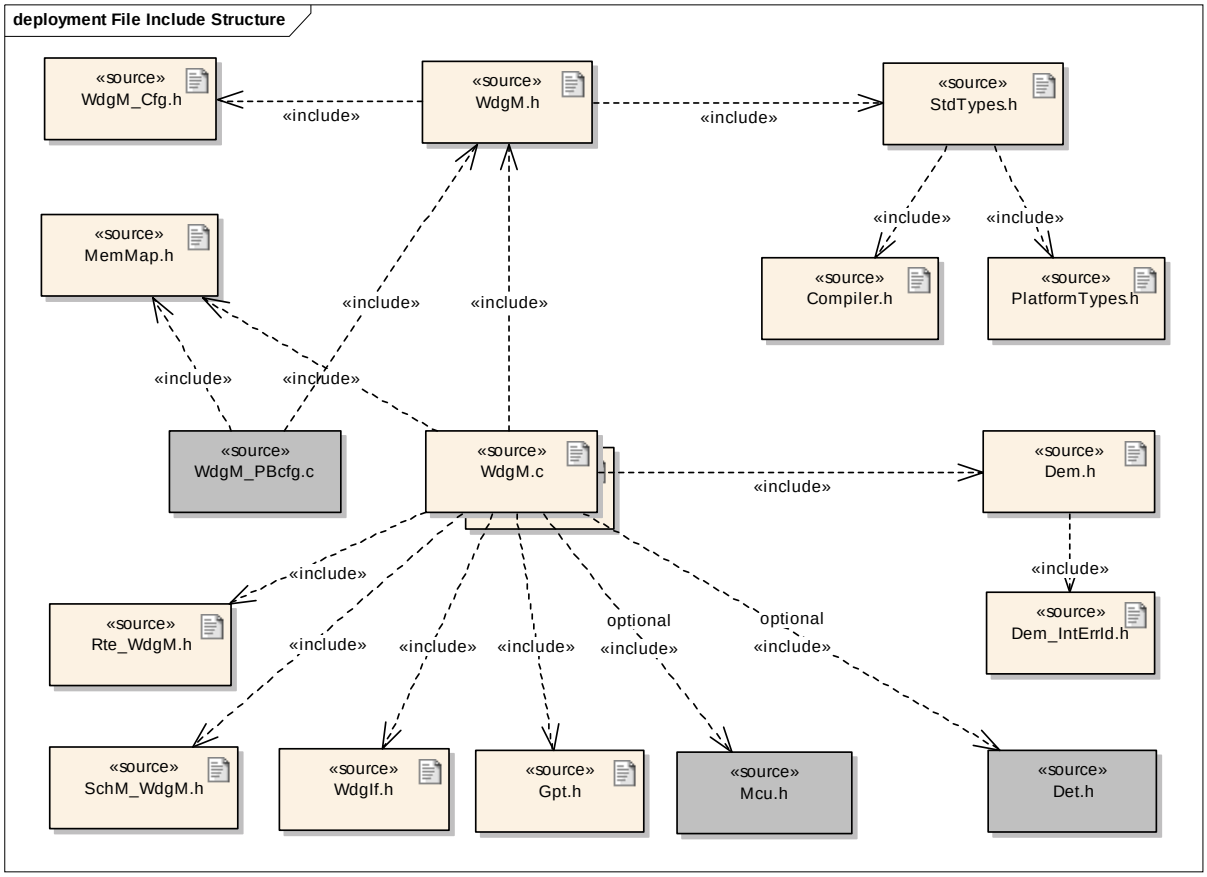

Figure 4: File include structure for the Watchdog Manager

Gray boxes are optional.

6 Requirements Traceability

Document: AUTOSAR General Requirements on Basic Software Modules [2]

Requirement	***Satisfied by***
[BSW003] Version identification	[WDGM012]
[BSW00300] Module naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00301] Limit imported information	[WDGM014]
[BSW00302] Limit exported information	[WDGM012]
[BSW00304] AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00305] Self-defined data types naming convention	[WDGM038]
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not for specification)
[BSW00307] Global variables naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not for specification)
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not for specification)
[BSW00310] API naming convention	[WDGM044]
[BSW00312] Shared code shall be re-entrant	Not applicable (requirement on implementation, not for specification)
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module does not implement any interrupt service routines)
[BSW00318] Format of module version numbers	[WDGM012]
[BSW00321] Enumeration of module version numbers	Not applicable (requirement on implementation, not for specification)
[BSW00323] API parameter checking	[WDGM010], [WDGM030][WDGM020], [WDGM031], [WDGM027], [WDGM055], [WDGM057]
[BSW00325] Runtime of interrupt service routines	Not applicable (this module does not implement any interrupt service routines)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (this module does not implement any interrupt service routines)
[BSW00327] Error values naming convention	[WDGM004]
[BSW00328] Avoid duplication of code	Not applicable (requirement on implementation, not for specification)
[BSW00329] Avoidance of generic interfaces	[WDGM050]
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not for specification)
[BSW00331] Separation of error and status	[WDGM004]

values	
[BSW00333] Documentation of callback function context	Not applicable (this module does not provide any callback routines)
[BSW00334] Provision of XML file	Not applicable (requirement on documentation, not on specification)
[BSW00335] Status values naming convention	Not applicable (no status type)
[BSW00336] Shutdown interface	[WDGM084], [WDGM045]
[BSW00337] Classification of errors	[WDGM004]
[BSW00338] Detection and Reporting of development errors	[WDGM048], WDGM049[WDGM010], [WDGM030], [WDGM020], [WDGM021], [WDGM031], [WDGM027], [WDGM028], [WDGM055], [WDGM056], [WDGM108], [WDGM057], [WDGM058], [WDGM039], [WDGM068], [WDGM088]
[BSW00339] Reporting of production relevant error status	[WDGM006], [WDGM015], [WDGM022], [WDGM129], [WDGM137], [WDGM138], [WDGM141], [WDGM142]
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation, not on specification)
[BSW00342] Usage of source code and object code	Not applicable (requirement on software architecture, not for a single module)
[BSW00343] Specification and configuration of time	Not applicable (timing constraints for alive-supervision are defined in number of executions)
[BSW00344] Reference to link-time configuration	Not applicable (this module does not provide link-time parameters)
[BSW00345] Pre-compile-time configuration	[WDGM025], [WDGM052], [WDGM047], [WDGM048], [WDGM104], WDGM009[WDGM111], [WDGM032], [WDGM092]WDGM109WDGM114
[BSW00346] Basic set of module files	[WDGM014]
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (requirement on implementation, not on specification)
[BSW00348] Standard type header	[WDGM014]
[BSW00350] Development error detection keyword	[WDGM048]WDGM049
[BSW00353] Platform specific type header	[WDGM014]
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00357] Standard API return type	[WDGM011]
[BSW00358] Return type of init() functions	[WDGM001]
[BSW00359] Return type of callback functions	Not applicable (this module does not provide any callback routines)
[BSW00360] Parameters of callback functions	Not applicable (this module does not provide any callback routines)
[BSW00361] Compiler specific language extension header	[WDGM014]
[BSW00369] Do not return development error codes via API	[WDGM048]
[BSW00370] Separation of callback interface from API	Not applicable (this module does not provide any callback routines)
[BSW00371] Do not pass function pointers via API	Not applicable (no function pointers in this specification)

[BSW00373] Main processing function naming convention	[WDGM049]
[BSW00374] Module vendor identification	[WDGM012]
[BSW00375] Notification of wake-up reason	Not applicable (this module does not implement wake-up interrupts)
[BSW00376] Return type and parameters of main processing functions	[WDGM043]
[BSW00377] Module specific API return types	Not applicable (no module specific return types)
[BSW00378] AUTOSAR boolean type	Not applicable (requirement on implementation, not for specification)
[BSW00379] Module identification	[WDGM012]
[BSW00380] Separate C-Files for configuration parameters	[WDGM127]
[BSW00381] Separate configuration header file for pre-compile time parameters	[WDGM014]
[BSW00383] List dependencies of configuration files	[WDGM014]
[BSW00384] List dependencies to other modules	[WDGM008], [WDGM009]
[BSW00385] List possible error notifications	[WDGM004]
[BSW00386] Configuration for detecting an error	Not applicable (requirement on implementation, not on specification)
[BSW00387] Specify the configuration class of callback function	Not applicable (this module does not provide any callback routines)
[BSW00388] Introduce containers	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00389] Containers shall have names	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00390] Parameter content shall be unique within the module	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00391] Parameter shall have unique names	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00392] Parameters shall have a type	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00393] Parameters shall have a range	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00394] Specify the scope of the parameters	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00395] List the required parameters (per parameter)	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00396] Configuration classes	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00397] Pre-compile-time parameters	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00398] Link-time parameters	Not applicable (this module does not provide link-time parameters)
[BSW00399] Loadable Post-build time parameters	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW004] Version check	[WDGM013]
[BSW00400] Selectable Post-build time parameters	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00401] Documentation of multiple instances of configuration parameters	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00402] Published information	[WDGM012]
[BSW00404] Reference to post build time configuration	[WDGM001], [WDGM016], [WDGM029], [WDGM032], [WDGM035], [WDGM037],

	[WDGM092], [WDGM094]
[BSW00405] Reference to multiple configuration sets	Not applicable (this module does not provide multiple configuration sets)
[BSW00406] Check module initialization	[WDGM021], [WDGM028], [WDGM056], [WDGM058], [WDGM039], [WDGM068], [WDGM088]
[BSW00407] Function to read out published parameters	[WDGM110]
[BSW00408] Configuration parameter naming convention	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW00409] Header files for production code error IDs	[WDGM126], [WDGM128], [WDGM014]
[BSW00410] Compiler switches shall have defined values	WDGM051[WDGM052], [WDGM047], WDGM049 [WDGM111]WDGM109WDGM114
[BSW00411] Get version info keyword	[WDGM111]WDGM109WDGM114
[BSW00412] Separate H-File for configuration parameters	[WDGM014]
[BSW00413] Accessing instances of BSW modules	Not applicable (requirement on implementation, not on specification)
[BSW00414] Parameter of init function	[WDGM001]
[BSW00415] User dependent include files	[WDGM014]
[BSW00416] Sequence of Initialization	Not applicable (this module is not responsible for initialization sequence)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (requirement for SW-Cs)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	[WDGM127]
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (requirement on documentation, not on specification)
[BSW00424] BSW main processing function task allocation	Not applicable (requirement on implementation, not on specification)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (requirement on documentation, not on specification)
[BSW00426] Exclusive areas in BSW modules	Not applicable (requirement on documentation, not on specification)
[BSW00427] ISR description for BSW modules	Not applicable (this module does not implement any interrupt service routines)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (requirement on implementation, not on specification)
[BSW00429] Restricted BSW OS functionality access	Not applicable (requirement on implementation, not on specification)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (requirement on implementation, not on specification)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (this module does not receive and transmit data path)

[BSW00433] Calling of main processing functions	Not applicable (requirement on implementation, not on specification)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (requirement on implementation, not on specification)
[BSW00435] Module Header File Structure for the Basic Software Scheduler	[WDGM014]
[BSW00436] Module Header File Structure for the Basic Memory Mapping	[WDGM014]
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on implementation, not on specification)
[BSW006] Platform independency	Not applicable (requirement on implementation, not for specification)
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW009] Module User Documentation	Not applicable (requirement on documentation, not on specification)
[BSW010] Memory resource documentation	Not applicable (requirement on documentation, not on specification)
[BSW101] Initialization interface	[WDGM001]
[BSW158] Separation of configuration from implementation	[WDGM014]
[BSW159] Tool-based configuration	[WDGM032], [WDGM035], [WDGM037], [WDGM092], [WDGM094]
[BSW160] Human-readable configuration data	Not applicable (requirement on documentation, not on specification)
[BSW161] Microcontroller abstraction	Not applicable (requirement on software architecture, not for a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on software architecture, not for a single module)
[BSW164] Implementation of interrupt service routines	Not applicable (this module does not implement any interrupt service routines)
[BSW167] Static configuration checking	Not applicable (requirement on configuration tool)
[BSW168] Diagnostic Interface of SW components	Not applicable (the module does not support a special diagnostic interface)
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (requirement for SW-Cs)
[BSW171] Configurability of optional functionality	[WDGM052], [WDGM047], [WDGM104], WDGM051 WDGM049[WDGM111]WDGM109WDGM114
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (requirement on documentation, not on specification)

Document: AUTOSAR Requirements on Mode Management [3]

Requirement	*Satisfied by*
[BSW09028] Support multiple watchdog instances	[WDGM002], [WDGM051], [WDGM103], [WDGM109], [WDGM066]
[BSW09106] Configuration of Watchdog Manager	[WDGM042], [WDGM046], WDGM084[WDGM090], [WDGM091], [WDGM095], [WDGM118], [WDGM096], [WDGM093], [WDGM003], [WDGM085], [WDGM087], [WDGM116]
[BSW09107] Watchdog Manager initialization	[WDGM151], [WDGM001], [WDGM018], [WDGM135]
[BSW09109] Prohibit disabling of watchdog	[WDGM030], [WDGM031] WDGM050
[BSW09110] Watchdog Manager mode selection service	[WDGM154], [WDGM062], [WDGM139]
[BSW09111] Watchdog driver triggering	[WDGM065], [WDGM066], [WDGM067], [WDGM119], [WDGM120], [WDGM121], [WDGM122], [WDGM040], [WDGM041]
[BSW09112] Check aliveness of application	[WDGM060], [WDGM061], [WDGM063], [WDGM074], [WDGM115], [WDGM097], [WDGM124], [WDGM073], [WDGM083], [WDGM098], [WDGM075], [WDGM100], [WDGM101], [WDGM078], [WDGM076], [WDGM077], [WDGM113], [WDGM117], [WDGM024], [WDGM130]
[BSW09125] Update alive-supervision	[WDGM155], [WDGM064], [WDGM026]
[BSW09142] Enabling / Disabling alive-supervision	[WDGM156], [WDGM157], [WDGM059], [WDGM079], [WDGM053], [WDGM054], [WDGM099], [WDGM083], [WDGM080], [WDGM144]
[BSW09143] Behavior of the Watchdog Manager during inactive alive-supervision	[WDGM083WDGM069]
[BSW09153] Access protection for disabling alive-supervision	WDGM080[WDGM082], [WDGM099]
[BSW09154] Parameter for access protection for disabling alive-supervision	[WDGM082], [WDGM093]
[BSW09158] Post build time configuration sets for the Watchdog Manager	[WDGM177], [WDGM105WDGM113], [WDGM180], [WDGM145]
[BSW09159] Reporting failure of the alive-supervision to DEM	[WDGM022], [WDGM129]
[BSW09160] Indication of failed alive-supervision	[WDGM113], WDGM123[WDGM024], [WDGM130], [WDGM148], [WDGM150]
[BSW09161] Condition to stop triggering the watchdog driver	[WDGM114]
[BSW09162] Indication of an upcoming watchdog reset	[WDGM150]
[BSW09163] Delay before provoking a watchdog reset	[WDGM125]
[BSW09167] Independent timing-constraints for watchdog-instances	[WDGM051], [WDGM103], [WDGM109]
[BSW09169] Immediate reset of the Watchdog Manager	[WDGM131], [WDGM132], [WDGM133], [WDGM134]
[BSW09171] Check aliveness of EcuM	[WDGM191], [WDGM193], [WDGM194]

7 Functional specification

This chapter presents the specification details of the internal functional behavior of the Watchdog Manager. Therefore the topics of the functional behavior focus on the following subjects: alive-supervision, initiation of watchdog triggering, modes of watchdog.

7.1 Alive-supervision

The alive-supervision of the Watchdog Manager offers a mechanism to periodically check the execution reliability of one or several supervised entities. This mechanism supports a checkup of cyclic timing constraints of independent supervised entities, tracks the checkup result as their *individual supervision status* and provides a *global supervision status* to support decision for triggering the hardware watchdog or not.
In general the alive-supervision supports three different states represented within the *global* and *individual supervision status* as the current result of its investigations.
First state represents no timing deviation detected.
The remaining two states differentiate escalation steps of detected timing deviations by the alive-supervision.
Second state is a first and optional escalation step to support error recovery within a certain amount of *failed supervision reference cycles*, while triggering of the watchdog still goes on. If a recovery occurs, alive-supervision goes on and gets back to state where timing constraints are fulfilled.
Third state, representing the second escalation step, does not support recovery of alive-supervision any more and therefore will end up with an ECU reset. But it offers a configurable amount of *expired supervision cycles* to support preparations of SW-Cs to get ready for upcoming reset, as far as it is possible.

7.1.1 Resources of alive-supervision

According to scalability and portability issues, mentioned in the introduction, and taking the individual timing constraints of supervised entities into account, the alive-supervision has to provide several configurable parameters to detail the behavior according to specific project demands.
In consideration to simplify usage and to achieve minimized effort of resources the following general mechanism is defined:

WDGM052: For ECUs which do not need this mechanism to supervise applications, the alive-supervision shall be configurable (on/off) at pre-compile time with the preprocessor switch `WdgMAliveSupervisionEnabled` (refer to chapter 10.2.3).

The alive-supervision of the Watchdog Manager shall not monitor absolute time periods of consecutive *alive indications* from supervised entities.

WDGM087: The timing constraints of alive-supervision shall be represented by a ratio of an expected amount of *alive indications* over a defined amount of *supervision cycles* of the Watchdog Manager.

In general the functional mechanism of the Watchdog Manager has to support a configurable and therefore flexible amount of independent supervised entities. As a consequence the following general issue has to be considered:

WDGM085: The required number of independent data resources to perform the alive-supervision within the Watchdog Manager should be derived from the number of assigned supervised entities (refer to chapter 7.1.3.1). Examples of independent data resources in context of the Watchdog Manager are: *alive counters*, *supervision cycles* counters, *failed supervision reference cycles* counters, *expired supervision cycles* counters, *individual supervision status*.

7.1.2 Functional mechanism

Within this subchapter the functional details of alive-supervision in the meaning of internal activities, provided services with its reactions and the periodicity of required processing service are described.

7.1.2.1 Supervision cycle

To guarantee a continuous alive-supervision during runtime, a cyclic checkup of asynchronous counted *alive indications* has to be performed. The time period for repeating examinations on supervised entities defines a logical *supervision cycle* in context of the Watchdog Manager.

WDGM063: The Watchdog Manager shall provide a processing service *"WdgM_MainFunction_AliveSupervision"* which is intended for a cyclic scheduled usage according to the *supervision cycle* of Watchdog Manager's alive-supervision.

Note: According to decoupled alive-supervision from watchdog triggering, the cycle period of the alive-supervision is independent from *trigger cycle*. Separated services are provided for each context (for details refer to chapter 8.5).

7.1.2.2 Activation status

According to application context of supervised entities, the alive-supervision may be affected by some events (e.g. inhibited execution of a supervised entity by the FiM, ModeDisablingDependencies of RTE (refer to [5]) or internal branches). Therefore it is necessary to track an *activation status* of the individual supervised entities, whether supervision needs to be performed on the corresponding entity or not.
The access to disable alive-supervision shall be limited according to configuration settings. Therefore the access to change the activation status shall be encapsulated by services.

WDGM099: The Watchdog Manager shall track the *activation status* of each assigned supervised entity independently according to *WdgM_ActivationStatusType* definition.

WDGM059: The Watchdog Manager shall provide the service *"WdgM_ActivateAliveSupervision"* to enable the alive-supervision of a selectable, single supervised entity.

WDGM079: The Watchdog Manager shall provide the service *"WdgM_DeactivateAliveSupervision"* to disable the alive-supervision of a selectable, single supervised entity.

WDGM082: The activation status of a supervised entity shall not be affected by the service *"WdgM_DeactivateAliveSupervision"* if the corresponding supervised entity is protected against deactivation access by setting of its static configurable parameter.
(for details of involved parameter refer to chapter 7.1.3.6; WDGM093).

WDGM144: The activation status of a supervised entity shall not be affected by the service *"WdgM_DeactivateAliveSupervision"* if the corresponding supervised entity has been detected as faulty i.e. its *individual supervision status* is equal to WDGM_ALIVE_EXPIRED.

WDGM080: When the alive-supervision of a supervised entity is disabled, its *individual supervision status* shall be set to WDGM_ALIVE_DEACTIVATED according to *WdgM_AliveSupervisionStatusType* definition.

7.1.2.3 Examination of individual alive counters

WDGM064: During the intermediate time of *supervision cycles* the *alive counters* may have been updated by synchronous usage of the *"WdgM_UpdateAliveCounter"* service according to passed *alive indication points* at the corresponding supervised entities. The cyclic examinations of these counters define the baseline strategy to check the reliability of cyclic execution on these corresponding supervised entities.

WDGM098: The examination of the *individual alive counters* shall only be performed in periods of the corresponding *supervision reference cycle* for each supervised entity. During the intermediate *supervision cycles* the *individual supervision status* shall remain unaffected on its current value.
(for details of involved parameters refer to chapter 7.1.3.2; WDGM090)

WDGM074: A first step of examination shall check, whether the counted *alive indications* (stored at the *alive counters*) matches to the expected amount of *alive indications* within tolerable margins and according to the referenced amount of *supervision cycles.*
(for details of involved parameters refer to chapter 7.1.3.2; WDGM090, WDGM091)

WDGM115: If this first examination step detects a deviation between the counted *alive indications* and the expected amount of *alive indications* (including tolerance margins), the alive-supervision has failed at this *supervision reference cycle* for this supervised entity and then the amount of *failed supervision reference cycles* shall be tracked.

WDGM097: A second step of examination shall check, whether the amount of *failed supervision reference cycles* of the corresponding supervised entity exceeds its configurable limit or not.
(for details of involved parameters refer to chapter 7.1.3.3; WDGM095)

WDGM124: If this second examination step detects an exceeding of the allowed amount of *failed supervision reference cycles*, the alive-supervision has expired for this supervised entity and then the amount of *expired supervision cycles* shall be tracked.

WDGM073: The current result of the completed examination shall be represented by an *individual supervision status* of each supervised entity. The result shall contain a value according to *WdgM_AliveSupervisionStatusType* definition:

WDGM_ALIVE_OK	if the *alive counter* value matches to the expected amount of *alive indications* (including the tolerance margins).
WDGM_ALIVE_FAILED	if the *alive counter* value doesn't match to the expected amount of *alive indications* (including the tolerance margins), but the acceptable amount of *failed supervision reference cycles* has not been exceeded.
WDGM_ALIVE_EXPIRED	if the *alive counter* value doesn't match to the expected amount of *alive indications* (including the tolerance margins) for more often than the acceptable amount of *failed supervision reference cycles.*

WDGM083: The examination of the *individual alive counters* shall not be performed if the alive-supervision of the corresponding supervised entity is disabled by its *activation status*. In this case the *individual supervision status* shall remain unaffected on status WDGM_ALIVE_DEACTIVATED.

WDGM075: The examination to calculate the *individual supervision status* of supervised entities shall be performed before the update of the *global supervision status* is calculated.

7.1.2.4 Recovery strategy of alive-supervision

The second step of alive-supervision examination (refer to WDGM097) does implicitly introduce the optional escalation strategy: If the second examination step does not detect an exceeding of the configured amount of *failed supervision reference cycles* tolerance, the corresponding *individual supervision status* shall be set to WDGM_ALIVE_FAILED (refer to WDGM095, WDGM097, WDGM073).

This state is intended to support an optional amount of *failed supervision reference cycles*, which offers a defined time in multiplicity of the corresponding *supervision*

reference cycle, where the triggering of the watchdog shall not be stopped and a recovery of the *individual supervision status* is possible. After this time has expired, recovery is obsolete.

WDGM113: If the *alive counter* returns to an expected amount of *alive indications* within its *supervision reference cycle*, while the corresponding *individual supervision status* is equal to WDGM_ALIVE_FAILED, the *individual supervision status* shall return to WDGM_ALIVE_OK.

WDGM130: If the *alive counter* does not return to an expected amount of *alive indications* within its *failed supervision reference cycles* tolerance, while the corresponding *individual supervision status* is equal to WDGM_ALIVE_FAILED, the *individual supervision status* shall be set to WDGM_ALIVE_EXPIRED (refer to WDGM073).

WDGM114: If any *individual supervision status* has reached the state WDGM_ALIVE_EXPIRED, this *individual supervision status* shall not be affected by software control any more, besides a reset.

Note: If the tolerance of *failed supervision reference cycles* is configured to 0 (zero), this should result into a direct transition of *the individual supervision status* from WDGM_ALIVE_OK to WDGM_ALIVE_EXPIRED, if the corresponding alive-supervision check fails the first time. In this case, no recovery-strategy will be performed (for details of the involved parameter refer to WDGM095).

For features to support individual reactions at SW-Cs derived from the context of a changed *individual supervision status* refer to chapter 7.4.2.

7.1.2.5 Individual Supervision Status State Machine

Figure 5 shows the resulting state machine for the Individual Supervision Status of each Supervised Entity. One such state machine exists for each Supervised Entity.

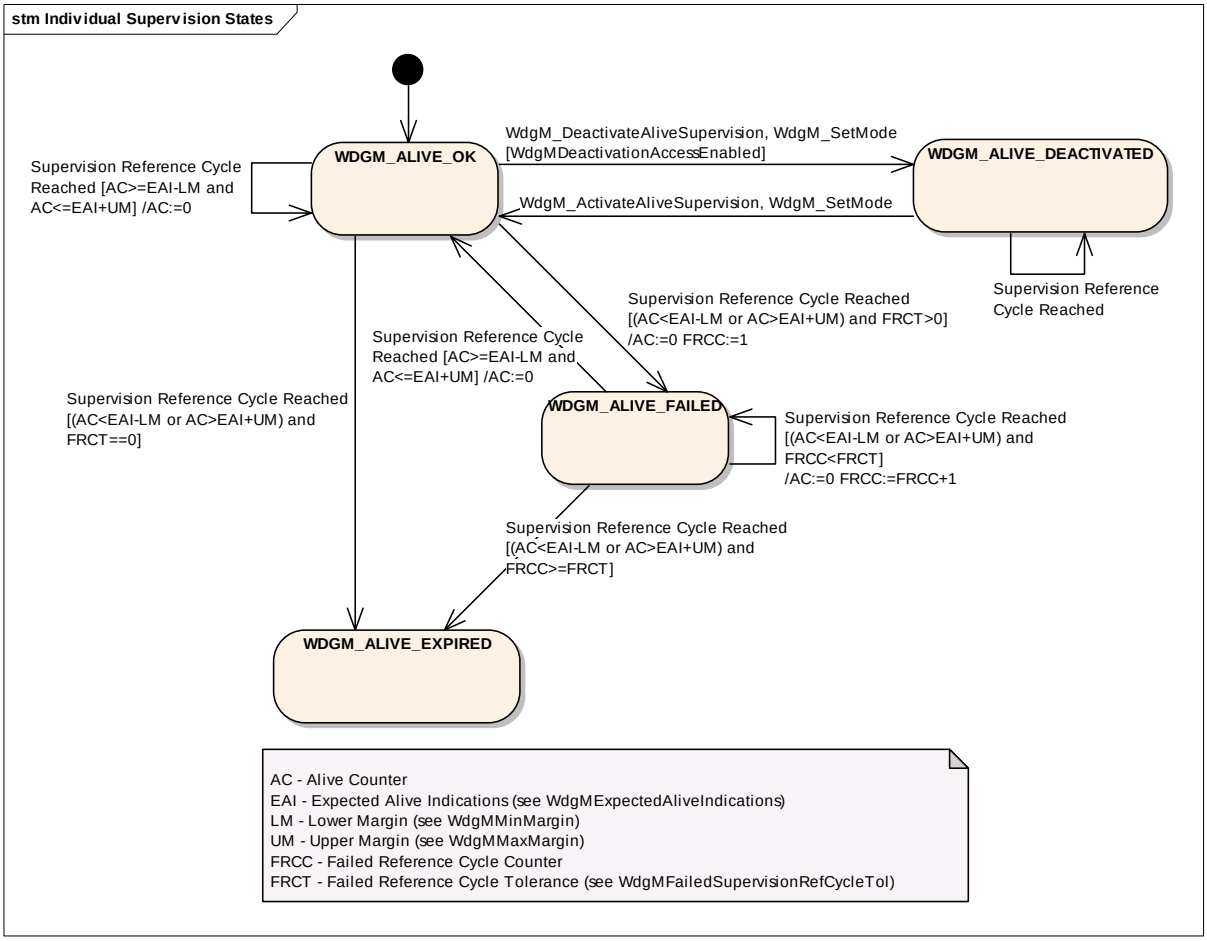

Figure 5: Individual Supervision Status State Machine

7.1.2.6 Calculation of Global Supervision Status

After checkup and update of *individual supervision status* at the corresponding *supervision cycles (supervision reference cycles),* the Watchdog Manager has to derive a merged result to support decision whether further watchdog triggering shall be blocked or not.

WDGM100: The calculated merged result shall be represented by a *global supervision status* and shall contain a unique value according to *WdgM_AliveSupervisionStatusType* definition.

WDGM101: The *global supervision status* shall be calculated and updated on every *supervision cycle*.

Following rules shall be considered to calculate the *global supervision status*:

WDGM078: The *global supervision status* shall be set to WDGM_ALIVE_OK if the *individual supervision statuses* of all “activated” supervised entities are equal to WDGM_ALIVE_OK.
Note: a supervised entity is interpreted as “activated” if its *activation status* is equal to WDGM_SUPERVISION_ENABLED.
(for details of *activation status*, refer to chapter 7.1.2.2)

WDGM076: If the *individual supervision status* of at least one supervised entity is equal to WDGM_ALIVE_FAILED, but no *individual supervision status* is equal to WDGM_ALIVE_EXPIRED, the *global supervision status* shall be set to WDGM_ALIVE_FAILED.

WDGM125: The Watchdog Manager shall support a feature to postpone blocking of watchdog triggering for a configurable amount of time measured in multiplicity of *supervision cycles* (*expired supervision cycles*). This could be used to support further preparation activities of SW-Cs before watchdog triggering will be stopped within a defined time-limit.

WDGM077: If the *individual supervision status* of at least one supervised entity is equal to WDGM_ALIVE_EXPIRED and the amount of *expired supervision cycles* to postpone the blocking of watchdog triggering is not exceeded, the *global supervision status* shall be set to WDGM_ALIVE_EXPIRED.

WDGM117: If the *global supervision status* has reached the state WDGM_ALIVE_EXPIRED and the amount of consecutive *expired supervision cycles* exceeds the configured limit to postpone the blocking of watchdog triggering, the *global supervision status* shall be set to WDGM_ALIVE_STOPPED.

WDGM131: For applications which need a microcontroller reset as soon as an unrecoverable alive-supervision failure is detected, the Watchdog Manager shall perform an immediate reset by calling the MCU service *Mcu_PerformReset*.

WDGM132: This feature shall be configurable (on/off) at pre-compile time with the preprocessor switch `WdgMImmediateReset` (refer to chapter 10.2.3).

WDGM133: If this feature is activated, when the *global supervision status* has reached the state WDGM_ALIVE_STOPPED, the Watchdog Manager shall call the MCU service *Mcu_PerformReset*.

WDGM134: In this case, the Watchdog Manager will not provide notification to the application via RTE mechanism.

Note: If the tolerance of *expired supervision cycles* is configured to 0 (zero), this should result into a direct transition of the *global supervision status* from WDGM_ALIVE_EXPIRED to WDGM_ALIVE_STOPPED. In this case, the direct call to *Mcu_PerformReset* will not be delayed.

For features to support reactions at SW-Cs derived from the context of a changed *global supervision status* refer to chapter 7.4.2.

For details of affecting the watchdog triggering refer to chapter 7.2.

7.1.2.7 Global Supervision Status State Machine

Figure 6 shows the resulting state machine for the Global Supervision Status. Only one such state machine exists on each ECU.

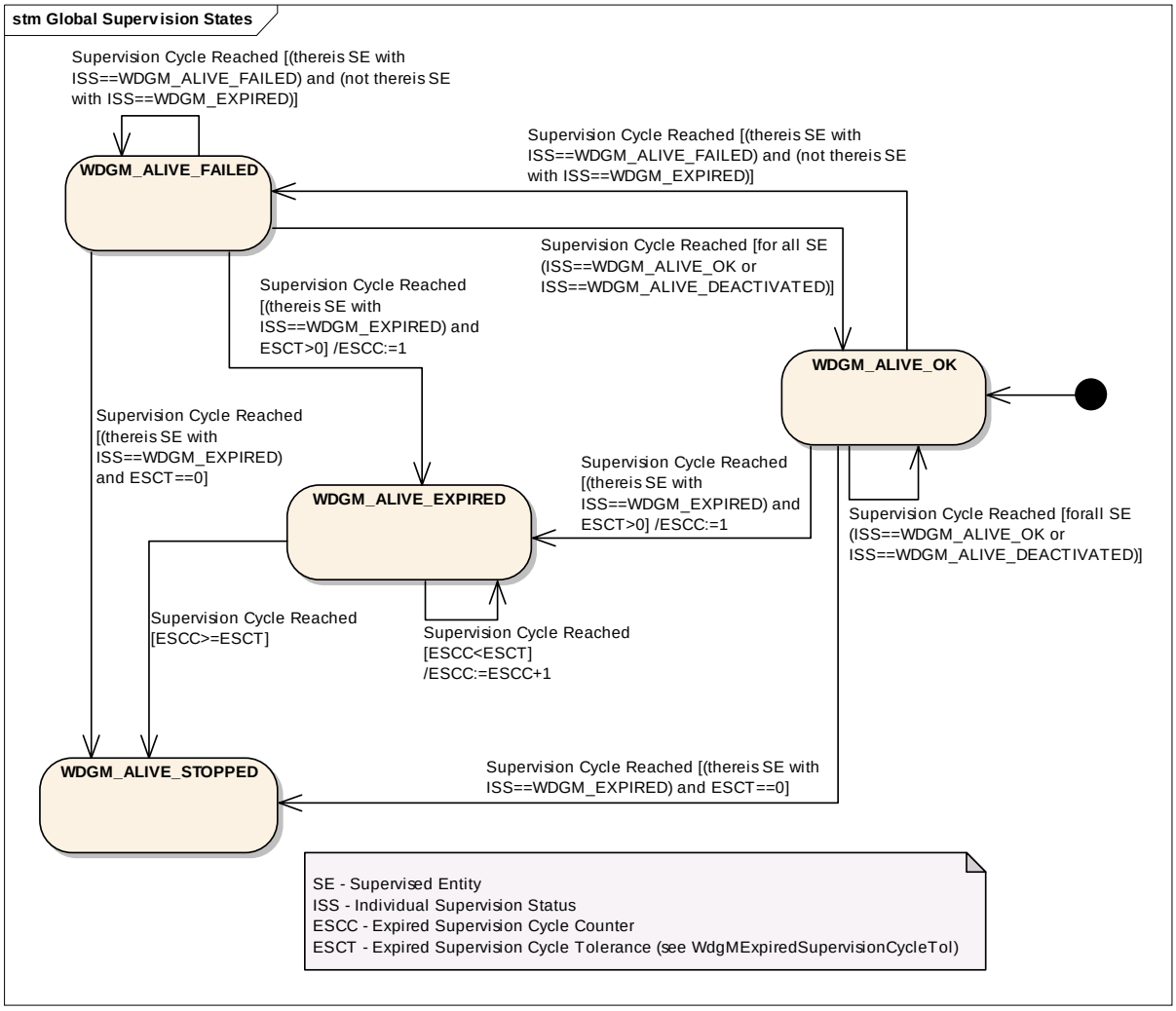

Figure 6: Global- Supervision Status State Machine

7.1.2.8 Error Entry to DEM

WDGM129: When the *global supervision status* has reached the state WDGM_ALIVE_STOPPED which means that the alive-supervision has failed and a reset will occur, an error status shall be reported to the DEM if the pre-processor switch WdgMDemAliveSupervisionReport is set (for details see chapters 7.5 and 7.7).

7.1.3 Configurable Parameters

To perform the alive-supervision, some preparations must have taken place. The SW-Cs which should be under alive-supervision and their individual timing constraints have to be assigned by configuration. Also the access to change the *activation status* of supervised entities may depend on their application context, which brings this feature to the list of configurable parameters.

7.1.3.1 Assigning supervised entities

According to portability support of SW-Cs across platforms the Watchdog Manager needs to be adapted to the amount of supervised entities located on the respective ECU. It is obvious that the amount of supervised entities has a “one-to-one” relationship on the amount of independent alive-supervisions which have to be performed by the Watchdog Manager. According to the static definition of SW-Cs location on platforms, the following issues have to be supported by configuration of the Watchdog Manager:

WDGM046: The Watchdog Manager shall support the assignment of supervised entity identifiers by pre-compile configuration.

7.1.3.2 Access of activation status

For a subset of supervised entities, it may not be acceptable to be inhibited for further execution derived from their application context (entities of non conditional execution reliability). As a corresponding conclusion, it is not acceptable to disable the alive-supervision of these entities during runtime under any condition.
Therefore the access of the *activation status* by the provided service *“WdgM_DeactivateAliveSupervision”* must not be granted.

WDGM093: The following configurable parameter shall be provided for each assigned supervised entity, to allow the deactivation of its alive-supervision:

- WdgMDeactivateAliveSupervisionAllowed

This parameter will be checked by the *“WdgM_DeactivateAliveSupervision”* service to know if deactivation of alive-supervision for this supervised entity is allowed or not.

7.1.3.3 Further configurable parameters

Further parameters of the alive-supervision shall be configurable but depend on the current mode of the Watchdog Manager. These parameters are described in chapter 7.3.1.

7.1.4 Example of alive-supervision algorithm

For the alive-supervision, an algorithm to detect mismatching timing constraints of the supervised entities is provided in order to clearly fix the parameters needed for the alive-supervision.
Doing this with incremental *alive counters* for the supervised entities 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 supervised entity 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 supervised entity is expected to occur less often than the *supervision cycle*.
The number of *supervision cycles* (SC) between two *alive indications* (AI) shall be counted.

To cope with these two scenarios, it is necessary to count both AI and SC.
We also need the parameter “WdgMExpectedAliveIndications” (EAI) which represents the expected amount of *alive indications* of the supervised entity within the referenced amount of *supervision cycles (supervision reference cycle)*. The value of this parameter should have been determined during the design phase and defined by configuration.

The alive-supervision is checked with the following algorithm:

n (AI) – n (SC) + EAI = 0

To avoid the detection of too many supervision errors for the supervised entities, we shall use the parameters “WdgMMinMargin” and “WdgMMaxMargin” to define tolerances on the timing constraints. For some non critical supervised entities, it may be allowed to have more or less executions than expected without impact on the reset of the application.
“WdgMMinMargin” represents the allowed number of missing executions of the supervised entity.
“WdgMMaxMargin” represents the allowed number of additional executions of the supervised entity.
Therefore the algorithm becomes:

(n (AI) – n (SC) + EAI <= WdgMMaxMargin) and
(n (AI) – n (SC) + EAI >= - WdgMMinMargin)

Scenario A:
To check, if the right amount of *alive indications* (n (AI)) was proceeded, EAI value is preloaded with the negative value of the expected *alive indications* + 1.

Example: 2 *alive indications* are expected in one *supervision cycle* which represents the *supervision reference cycle*:
EAI = -2 + 1 = -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) + EAI = 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*.

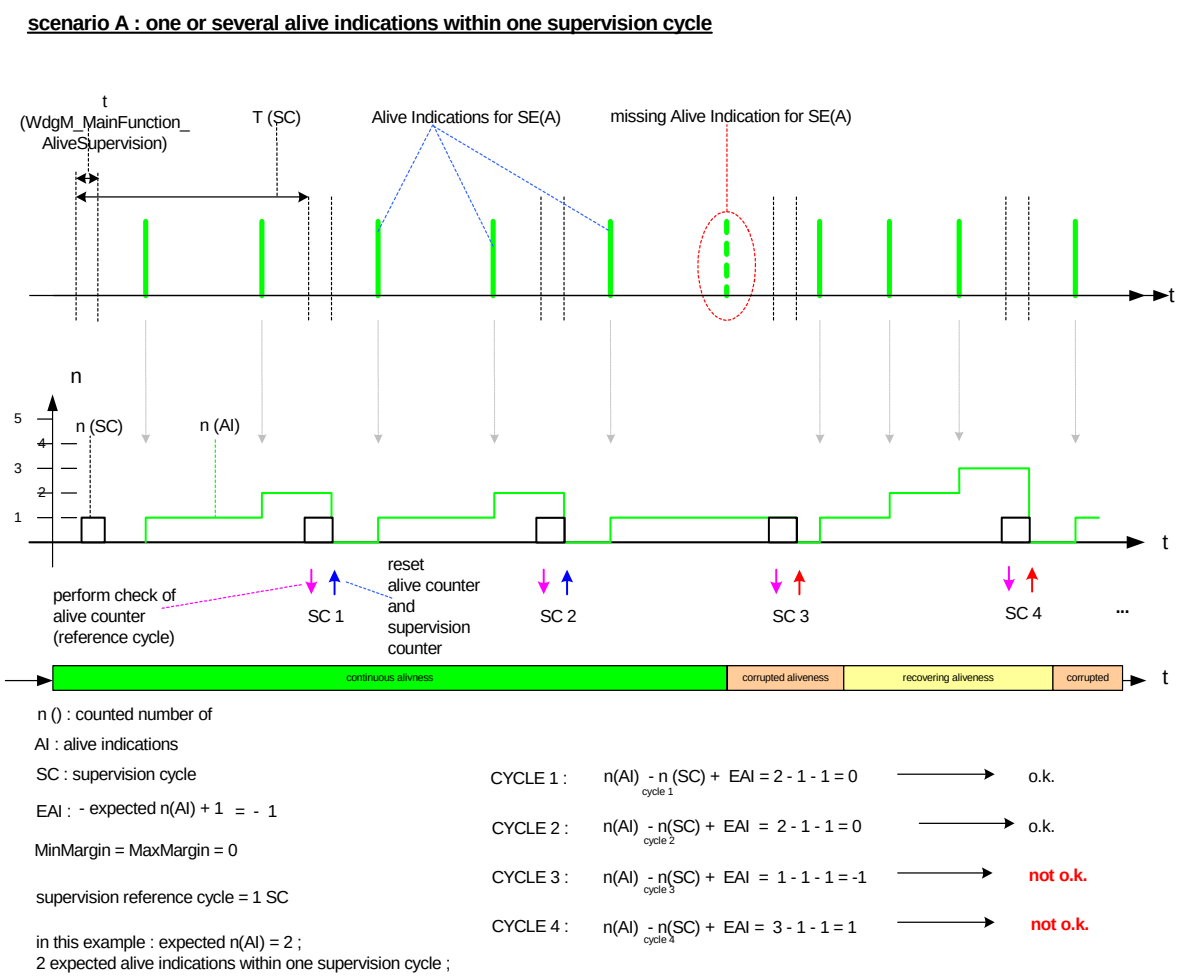

Figure 7: Alive-supervision algorithm – Scenario A

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) + EAI = 0
The *alive indication* must occur at least within a predefined number of *supervision cycles* which represent the *supervision reference cycle*.
The EAI value is pre-configured with the positive value of the expected *supervision cycles* before the next *alive indication* – 1.

Example: one *alive indication* is expected within 2 *supervision cycles* (*supervision reference cycle* = 2 *supervision cycles*):
EAI = +2 – 1 = +1
The *alive counter* shall 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* shall also invoke a reset of the *alive counter* and *supervision counter* after this compare-operation.
If a mismatch of the alive-supervision is detected and recovering of supervision shall be possible for further *alive indications*, the *alive counter* must be reset also. The reset must only be performed, for those supervised entities that were configured with a “WdgMFailedSupervisionRefCycleTol” different from 0, because this brings back alive-supervision in suitable condition for the next *supervision cycle*.

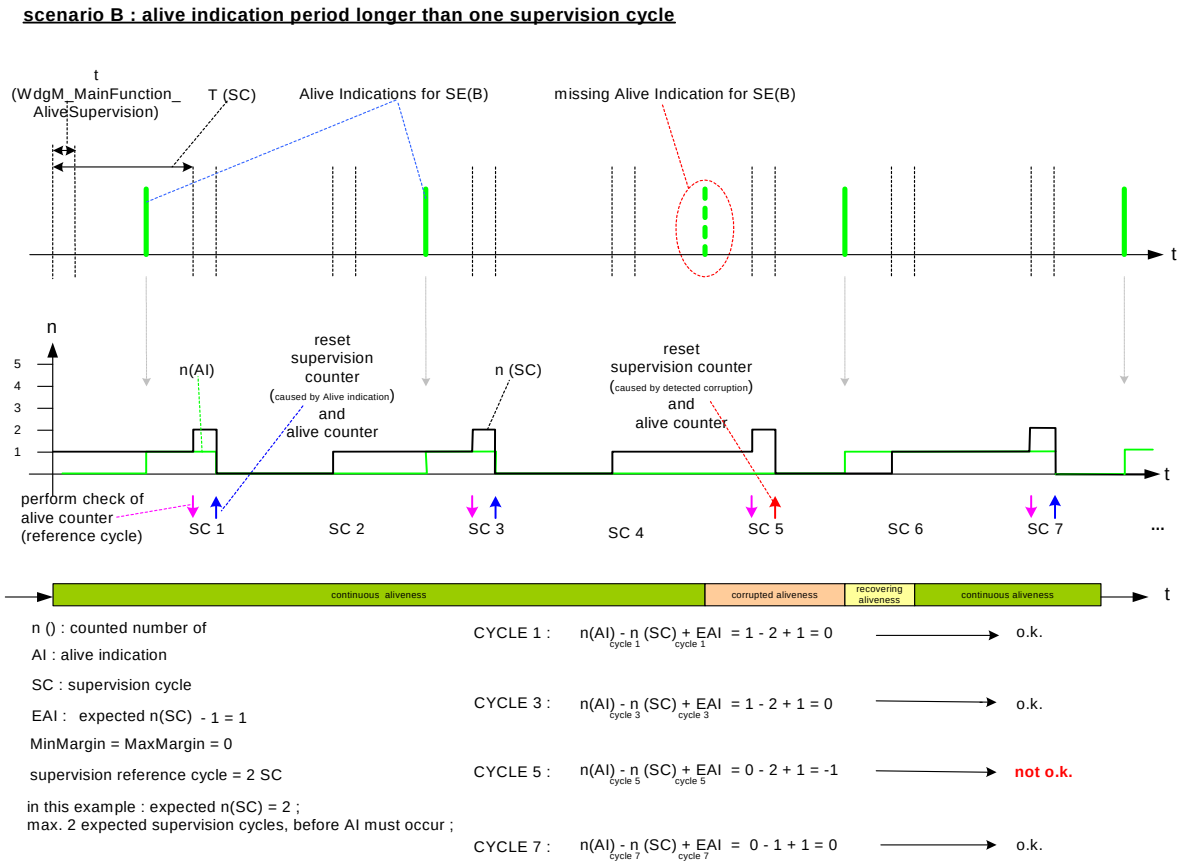

Figure 8: Alive-supervision algorithm – Scenario B

7.2 Triggering the Watchdog

The initiation of triggering the watchdog is the most important feature of the Watchdog Manager to prevent the ECU from resets by expired hardware watchdog instances while program execution is running properly.

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

Usually hardware watchdogs have their own timing constraints and the trigger-signal for each watchdog instance must be performed cyclically within a maximum time-period or within a defined time-window according to the timing constraints of the corresponding watchdog instance. If it doesn't occur, the corresponding hardware watchdog instance will invoke the reset.

The condition to trigger the watchdog instance(s) is based on the current *global supervision status*, which is managed by the alive-supervision mechanism (for details of alive-supervision refer to chapter 7.1)

7.2.1 Functional mechanism

Within this subchapter the functional details to perform watchdog triggering and the periodicity of required processing service are described.

7.2.1.1 Support multiple watchdog instances

WDGM002: The Watchdog Manager shall support the parallel usage of multiple watchdogs.

7.2.1.2 Trigger Cycle

According to cyclic context of watchdog triggering, a mechanism to support suitable cyclic triggering of watchdog instances shall be managed by the Watchdog Manager. Usually, timeout periods of individual hardware watchdog instances are different, which may require individual *trigger periods* for these watchdog instances.

WDGM065: For performing the triggering of watchdog instances, the Watchdog Manager shall provide a processing service *"WdgM_MainFunction_Trigger"*, which shall be scheduled cyclically.

The schedule period of the *"WdgM_MainFunction_Trigger"* defines a logical *trigger cycle* in context of the Watchdog Manager.

WDGM103: The individual *trigger period* of a watchdog instance shall be derived by a prescaled multiplicity of the *trigger cycle* which will represent the *trigger reference cycle* for this watchdog instance.

7.2.1.3 Example of a trigger schedule design

Usually a suitable schedule period for *"WdgM_MainFunction_Trigger"* service could be derived by GCR (greatest common ratio) of required *trigger periods* from all watchdog instances.

To avoid additional delays between detecting a supervision failure and stopping the watchdog triggering, the schedule period of WdgM_MainFunctionTrigger should also be a ratio of the supervision cycle.

It is recommended to system designer to harmonize *trigger periods* of hardware watchdog instances if possible, to ease up or avoid the prescaling of triggering.

Embedded into OS context, the cyclic task, which schedules the *"WdgM_MainFunction_Trigger"* service, might be delayed by some interrupts. To avoid the watchdog expiring meanwhile and in case of non window-watchdog, the *"WdgM_MainFunction_Trigger"* service schedule period should be configured much shorter than the hardware watchdog timeout (it could likely occur in event driven systems that timer-interrupts are issued at the same time especially if some alarms share the same counter).

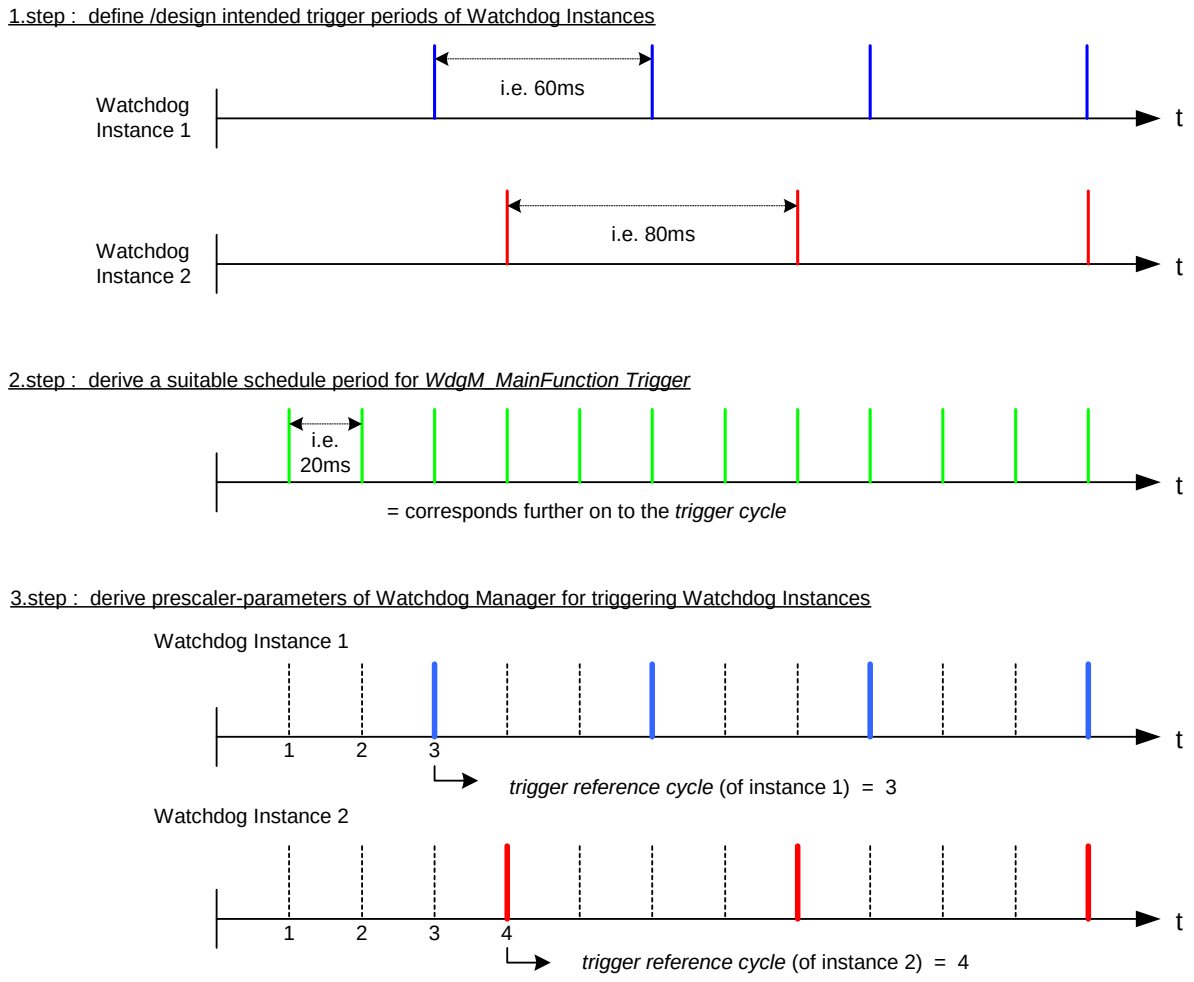

Figure 9: Trigger schedule for 2 watchdog instances

7.2.1.4 Conditions to trigger watchdog instances

The triggering of the watchdog is decoupled from the alive-supervision so far. Within the *"WdgM_MainFunction_AliveSupervision"*, the *global supervision status* will be updated and used by the *"WdgM_MainFunction_Trigger"* service to make a decision if the triggering should be performed or if it should be inhibited.

Following rules shall be considered to derive the decision, whether the watchdog triggering needs to be performed or not:

WDGM119: If the *global supervision status* is equal to WDGM_ALIVE_OK, the watchdog shall be triggered.

WDGM120: If the *global supervision status* is equal to WDGM_ALIVE_FAILED, the watchdog shall be triggered.

WDGM121: If the *global supervision status* is equal to WDGM_ALIVE_EXPIRED, the watchdog shall be triggered.

WDGM122: If the *global supervision status* is equal to WDGM_ALIVE_STOPPED, the watchdog shall not be triggered.

7.2.1.5 Perform triggering

WDGM051: The triggering of a dedicated watchdog instance shall be invoked by the *"WdgM_MainFunction_Trigger"* service if conditions of *global supervision status* are granted.

WDGM109: The triggering of a dedicated watchdog instance shall be performed when the *trigger cycle* matches the *trigger reference cycle* of the watchdog instance.

WDGM066: The *"WdgM_MainFunction_Trigger"* service shall call the *"WdgIf_Trigger"* service of the watchdog interface with a "Device Index" of the corresponding watchdog instance as parameter.

Note: The *"WdgIf_Trigger"* service will call synchronously the *"Wdg_Trigger"* service of the corresponding watchdog driver to trigger the hardware watchdog instance.

7.2.2 Configurable Parameters

7.2.2.1 Watchdog Device Index

WDGM003: The “Device Index” which is necessary to address different watchdog drivers shall be statically configured and represented by the following parameter:

- WdgMDeviceRef

7.2.2.2 Further configurable parameters

Further parameters of the watchdog triggering shall be configurable but depend on the current mode of the Watchdog Manager. These parameters are described in chapter 7.3.2.

7.3 Modes of the Watchdog Manager

The Watchdog Manager supports a number of different modes of operation. Each mode is defined by:

- the set of active supervised entities and their alive supervision parameters,
- the set of watchdogs to be triggered and their triggering parameters, and
- the source of activation for cyclic activities of the Watchdog Manager (i.e. activation by the Basic Software Scheduler (SchM) or the General Purpose Timer (GPT))

WDGM177: The Watchdog Manager shall support a configurable number of modes.

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 may be needed: when multiple applications run on the same ECU, one application may be shutdown already and require no supervision, while another application still runs and needs supervision.

WDGM178: Each mode of the Watchdog Manager shall have a unique identifier.

WDGM179: The Watchdog Manager shall have one initial mode which shall be activated when it is initialized.

Since the initial mode is activated before OS is running, the initial mode must be configured to use GPT activation.

7.3.1 Mode-dependent Alive-supervision Parameters

Changing the mode of the Watchdog Manager leads to changed conditions of the alive-supervision such as different entities to supervise or different timing constraints of supervised entities.

WDGM105: The Watchdog Manager shall provide for each configured mode and for each supervised entity a number of statically configured alive-supervision parameters.

7.3.1.1 Initial activation status

The initial *activation status* could differ for each supervised entity according to its application context. Therefore the initial activation status needs to be a configurable issue.

WDGM096: The following configurable parameter shall be provided for each assigned supervised entity:

- WDGMACTIVATIONACTIVATED

7.3.1.2 Assigning timing constraints

The alive-supervision of independent supervised entities is based on their individual periodicity and time-tolerances according to their application context.
Therefore the timing constraints of each supervised entity need to be represented by configurable parameters of the Watchdog Manager.

WDGM090: According to the simplified mechanism of alive-supervision, the expected periodicity of supervised entities shall be represented by the following two parameters:

- WdgMExpectedAliveIndications
- WdgMSupervisionReferenceCycle

WDGM091: The absolute time tolerances of alive-supervision shall be transferred into countable margins as deviations regarding the expected amount of *alive indications* by the following two parameters:

- WdgMMinMargin
- WdgMMaxMargin

7.3.1.3 Tolerance of failed supervision reference cycles

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.

WDGM095: The acceptable amount of *failed supervision reference cycles* shall be represented by the following parameter:

- WdgMFailedSupervisionRefCycleTol

7.3.1.4 Tolerance of expired supervision cycles

When the alive-supervision has reached expired conditions by any *individual 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*. If this feature is required and how many consecutive *supervision cycles* are needed does have an application context. Therefore the amount of consecutive *supervision cycles* needs to be a configurable parameter.

WDGM118: The amount of consecutive *expired supervision cycles*, which are used to postpone the blocking of watchdog triggering, shall be represented by the following parameter:

- WdgMExpiredSupervisionCycleTol

7.3.2 Mode-dependent Watchdog Settings

Changing the mode of the Watchdog Manager also leads to changed conditions for triggering the watchdogs such as different watchdog modes and different timing constraints of supervised entities.

WDGM180: The Watchdog Manager shall provide for each configured mode and for each watchdog a number of statically configured triggering parameters.

7.3.2.1 Watchdog Mode

WDGM181: For each watchdog instance, the watchdog mode shall be statically configured and represented by the following parameter:

- WdgMWatchdogMode

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

7.3.2.2 Trigger reference cycle

WDGM116: For each watchdog instance, the *trigger reference cycle* which defines the prescaler value for the corresponding *trigger period* shall be statically configured and represented by the following parameter:

- WdgMTriggerReferenceCycle

7.3.3 Mode-dependent activation source parameters

For each mode of the Watchdog Manager the designer can choose between two activation sources:

- Main Function activation is achieved via the BSW Scheduler (SchM) which cyclically calls *"WdgM_MainFunction_AliveSupervison"* and *"WdgM_MainFunction_Trigger".*
- GPT activation is achieved by configuring the General Purpose Timer (Gpt) which cyclically calls *"WdgM_Cbk_GptNotification"*.

GPT activation is necessary in startup, shutdown, and sleep phase, where no OS and no scheduler are running. In all other cases it is recommended to use Main Function activation only.

Beware that with GPT activation, the alive supervision algorithm and the watchdog triggering are executed in a GPT callback which in turn runs in an ISR context. This changes the semantics of watchdog triggering significantly, especially if no supervised entities are actively supervised. This may not be acceptable for safety reasons!

WDGM191: For each mode it shall be possible to choose between Main Function and GPT activation by configuration.

7.3.3.1 Main Function activation parameters

WDGM192: For each mode that uses Main Function activation, the periods for calls to *"WdgM_MainFunction_AliveSupervison"* and *"WdgM_MainFunction_Trigger"* shall be statically configured and represented by the following parameters:

- WdgMSupervisionCycle
- WdgMTriggerCycle

7.3.3.2 GPT activation parameters

WDGM193: For each mode that uses GPT activation, the GPT channel shall be statically configured and represented by the following parameter:

- WdgMGptChannelRef

WDGM194: For each mode that uses GPT activation, the periods for calls to *"WdgM_Cbk_GptNotification"* shall be statically configured and represented by the following parameter:

- WdgMGptCycle

7.3.4 Switching Modes

WDGM016: The Watchdog Manager can be switched between different modes that are statically configured and contained in the Watchdog Manager configuration set.

7.3.4.1 Preconditions

From a safety point of view, executing a mode switch is not allowed when the Watchdog Manager has already detected an alive supervision failure. In this case a mode switch has no effect.

WDGM145: The service "*WdgM_SetMode*" shall only be executed if the *global supervision status* is equal to WDGM_ALIVE_OK.

7.3.4.2 Effect on supervision status

A mode switch changes the alive-supervision parameters of the supervised entities. Therefore, alive-supervision status has to be adapted to the new mode.

WDGM182: When executing a mode switch, the Watchdog Manager shall retain the current individual supervision status of each supervised entity that is activated in the old and in the new mode. It shall only start a new alive-supervision cycle with the parameters for the new mode.

WDGM183: If the supervised entity is deactivated in the new mode, the Watchdog Manager shall change the individual supervision status to WDGM_ALIVE_DEACTIVATED.

WDGM184: If the supervised entity was deactivated in the old mode and is activated in the new mode, the Watchdog Manager shall assume the initial supervision status of WDGM_ALIVE_OK.

WDGM185: After adapting the individual supervision status of each supervised entity to the new mode, the Watchdog Manager shall compute a new global supervision status.

7.3.4.3 Effect on watchdogs

A mode switch also changes the parameters for watchdog triggering.

WDGM186: When executing a mode switch, the Watchdog Manager shall apply the configured watchdog mode to each watchdog by calling the *"WdgIf_SetMode"* service.

WDGM139: If one *"WdgIf_SetMode"* service fails, the Watchdog Manager shall assume a global alive-supervision failure and set the global supervision status to WDGM_ALIVE_STOPPED.

Note that this failure will cause a reset, either when the first watchdog expires or immediately, if an immediate reset of the Watchdog Manager is configured.

7.3.4.4 Effect on activation source

Switching the mode of the Watchdog Manager can also change its source of activation. While activation by the BSW Scheduler (SchM) is enabled as long as the scheduler runs, activation by the General Purpose Timer (GPT) has to be enabled and disabled.

WDGM187: If the old mode uses GPT activation, the Watchdog Manager shall deactivate the configured GPT when switching modes.

WDGM188: If the new mode uses GPT activation, the Watchdog Manager shall activate the configured GPT when switching modes.

WDGM189: If the current mode uses GPT activation, the Watchdog Manager shall ignore calls to *"WdgM_MainFunction_AliveSupervison"* and *"WdgM_MainFunction_Trigger".*

WDGM190: If the current mode uses Main Function activation, the Watchdog Manager shall ignore calls to *"WdgM_Cbk_GptNotification"*.

Note that switching activation sources may require re-synchronization between the activation source and the watchdogs. For watchdogs that support oversampling, this can easily be achieved by triggering the watchdog during the switch.

7.3.4.5 Error Entry to DEM

WDGM141: In case of switch mode failure, the Watchdog Manager shall report an error status to the DEM (for details see chapters 7.5 and 7.7).

7.4 Watchdog Handling during Sleep

While the ECU State Manager is in SLEEP state, the normal execution of code and therefore also of the Watchdog Manager 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.

To cater for this Use Case the ECU State Manager activates a Watchdog Manager mode in each sleep mode (see [6] configuration container EcuMSleepMode). The Integrator must configure each of these Watchdog Manager modes to use GPT Activation (Main Functions will not be executed). In addition, the Integrator must configure the used GPT channel as a Wakeup Source in the ECU State Manager (see [6] configuration container EcuMWakeupSource) that is active in the used sleep modes.

When the ECU State Manager enters SLEEP state it activates the sleep mode and sets up the GPT channel as a wakeup source. It also activates the Watchdog Manager mode, which sets up the timing and notification of the GPT channel. Then the ECU goes to sleep.

When the GPT counter has reached the configured value, it will cause a wakeup interrupt or the wakeup event will be detected by polling (depends on type of sleep mode and configuration of wakeup source). The ECU State Manager enters WAKEUP I state and activates the configured Watchdog Manager mode to be used during wakeup. Then it checks and if necessary validates other wakeup sources and afterwards executes EcuM_OnWakeupReaction. In this callout, the Integrator has to check if there are other valid wakeup sources except the used GPT channel (using EcuM_GetValidatedWakeupEvents). If the GPT channel is the only valid wakeup source, the callout should return ECUM_WKACT_SHUTDOWN, so the ECU State Manager will afterwards re-enter the previous sleep mode, as described in the previous paragraph.

7.5 Specification of the Ports and Port Interfaces

This chapter specifies the AUTOSAR Interfaces which are provided by the Watchdog Manager. Note that ports on both sides of the RTE are required: The SW-C description of the Watchdog Manager Service will define the ports below the RTE. Each AUTOSAR SW-C, which uses the service, must contain service ports in its own SW-C description. These ports are typed with the same interfaces and have to be connected to the ports of the Watchdog Manager, so that the RTE can generate the appropriate IDs and the required symbols.
The statuses of individual supervised entities and the global supervision status of the Watchdog Manager are reported to SW-Cs through mode ports. An SW-C can define its own mode port with the same interface as the mode ports of the Watchdog Manager. Afterwards the SW-C 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.

All the following interface definitions are interpreted to be in:

`ARPackage AUTOSAR/Services/WdgM`

7.5.1 Ports and Port Interface for Alive Supervision

7.5.1.1 General Approach

To reduce the number of ports provided by the Watchdog Manager all interfaces between SW-Cs and the service are modeled as Client/Server communication. To update the alive counter the sender-receiver paradigm seems 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.
There is one type of APIs defined within the watchdog manager:

- A supervised entitiy (SW-C) needs the Watchdog Manager for services dealing with individual supervised entities.

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 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. 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 both within the application (SW-C) as well as within the service (Watchdog Manager).

7.5.1.2 Data Types

For the port interface of the watchdog service no additional data types are required. The only parameter passed between the application and the service is the ID to identify the supervised entity. The type for this identifier shall be based on the type

WdgM_SupervisedEntityIdType. This type is currently defined as `uint8/uint16`. Therefore the following type description is required:

```
Uint16 WdgM_SupervisedEntityIdType {
          LOWER-LIMIT = 0;
          UPPER-LIMIT = <#SE-1>;
     };
```

Thus, all Watchdog Manager APIs are using the *WdgM_SupervisedEntityIdType* type instead of the `uint16` type. It is very likely that due to efficiency or other reasons the integer type might vary between ECUs[1]. Therefore the ECU specific type definition has to be provided by the Watchdog Manager instead of providing an "AUTOSAR type".

The number of supervised entities is defined in the configuration parameter WdgMNumberOfSupervisedEntities.

7.5.1.3 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 to the APIs defined within this specification:

```
ClientServerInterface WdgM_AliveSupervision {
     isService = true;
     PossibleErrors {
          E_NOT_OK
     };
     UpdateAliveCounter (ERR{E_NOT_OK});
     ActivateAliveSupervision (ERR{E_NOT_OK});
     DeactivateAliveSupervision (ERR{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.

7.5.1.4 Service Ports

Figure 10 shows how AUTOSAR Software components (single or multiple instances) are connected via service ports to the Watchdog Manager. 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`.

[1] Actually, supervised entity identifier are only used indirectly via port-defined argument values (see chapter *7.5.4 Internal Behavior*). Thus, an SW-C will never use `WdgM_SupervisedEntityIdType` directly in any of its interfaces.

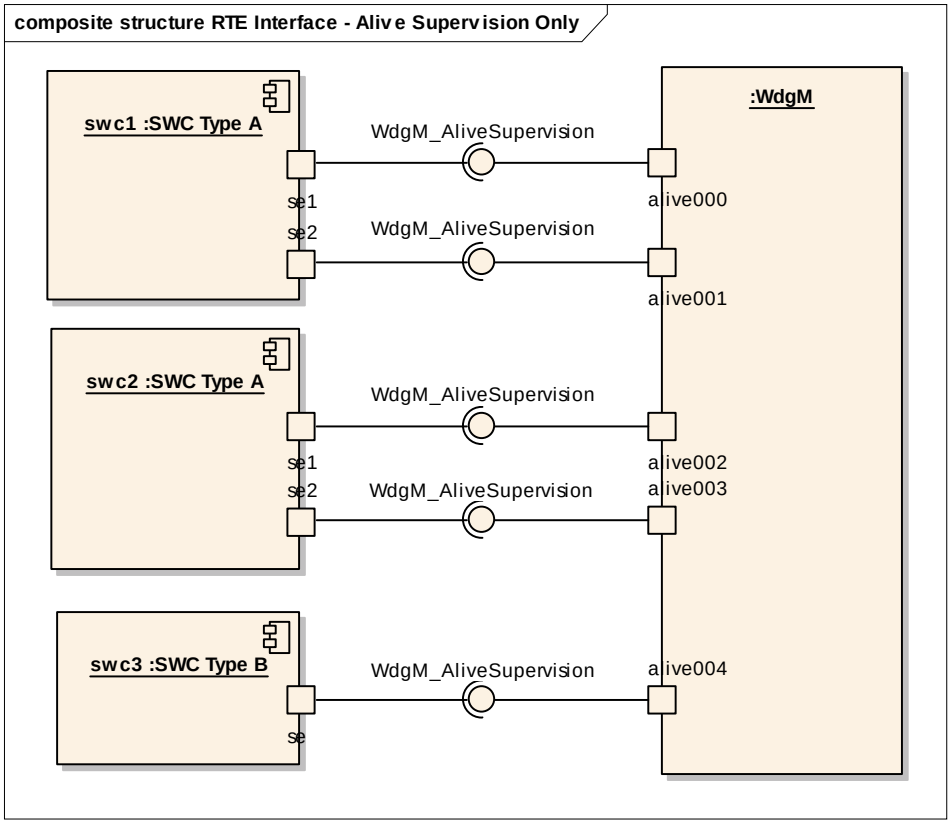

Figure 10: Example of SW-Cs connected to the Watchdog Manager via service ports

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.

WDGM146: The Watchdog Manager 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.

WDGM147: The Alive Supervision ports 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.

The number of supervised entities is defined in the configuration parameter WDGMNUMBEROFSUPERVISEDENTITIES.

7.5.1.5 Error Codes

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

7.5.2 Ports and Port Interface for Status Reporting

7.5.2.1 General Approach

To control the state-dependent behavior of SW-Cs 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 that connects to the mode port can use the mode information in two ways:

- The SW-C can query the current mode via the mode port.
- The SW-C 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 `SenderReceiverInterface`. The mode manager, here the Watchdog Manager, is the sender and the SW-Cs are the receivers.

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

- First, it provides the individual supervision status of each supervised entity. Therefore, the Watchdog Manager has a mode port for each supervised entity.
- Second, the Watchdog Manager provides the global supervision status, which reflects the combined supervision states of all supervised entities. Therefore, it has one additional mode port.

WDGM195: The mode ports of the Watchdog Manager shall declare the following modes:

ALIVE_OK
ALIVE_FAILED
ALIVE_EXPIRED
ALIVE_STOPPED
ALIVE_DEACTIVATED

This definition corresponds to the type `WdgM_AliveSupervisionStatusType`.

WDGM196: The Watchdog Manager shall notify SW-Cs 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_WdgM_Mode mode)
```

and for the global state

```
Rte_StatusType Rte_Switch_globalMode_currentMode(
      Rte_ModeType_WdgM_Mode 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 *7.5.2.3 Port Interfaces* may result in less code when reporting the state to individual supervised entities and may therefore be used alternatively to the above API.

WDGM197: When the individual supervision state of a single supervised entity changes, the Watchdog Manager shall report that change via the mode port for that supervised entity immediately after it has been recognized.

WDGM198: When the global supervision state changes, the Watchdog Manager shall report that change via the global mode port.

WDGM199: After computing the global supervision state from all individual supervision states, the Watchdog Manager shall report any change in the resulting global supervision state only once.

The resulting behavior is that first all changes in individual supervision states are reported. Afterwards the global supervision state is reported only once and only if it changed due to the individual changes.

For instance, if in one supervision cycle SE1 goes from ALIVE_OK to ALIVE_FAILED, ALIVE_FAILED is reported on the individual mode port for SE1. In the same supervision cycle SE2 goes from ALIVE_OK to ALIVE_EXPIRED directly, ALIVE_EXPIRED is reported on the individual mode port for SE2. The resulting global supervision state in this supervision cycle changes from ALIVE_OK to ALIVE_EXPIRED and only ALIVE_EXPIRED is reported on the global mode port. In that example ALIVE_FAILED is not reported on the global mode port, because it was only an intermediate state while evaluating a subset of supervised entities.

7.5.2.2 Data Types

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

```
ModeDeclarationGroup WdgMMode {
     { ALIVE_OK,
     ALIVE_FAILED,
     ALIVE_EXPIRED,
     ALIVE_STOPPED,
     ALIVE_DEACTIVATED
     }
     initialMode = ALIVE_OK
};
```

7.5.2.3 Port Interfaces

There are two different interfaces to indicate changes in the supervision status to interested SW-Cs and the RTE.

The interface WdgM_IndividualMode is used to signal the individual supervision status of a single supervised entity.

```
SenderReceiverInterface 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.

```
SenderReceiverInterface 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 provides only one single port with that interface. By contrast, for the WdgM_IndividualMode interface the Watchdog Manager 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.:

```
void signalOkay(WdgM_SupervisedEntityIdType se)
{
     Rte_PortHandle_WdgM_IndividualMode_P ph =
                          Rte_Ports_WdgM_IndividualMode_P();
     ph[se].Switch_currentMode(RTE_MODE_WdgM_Mode_ALIVE_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.

7.5.2.4 Mode Ports

Figure 11 shows how AUTOSAR Software components (single or multiple instances) are connected via mode and service ports to the Watchdog Manager. 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 state of all supervised entities.

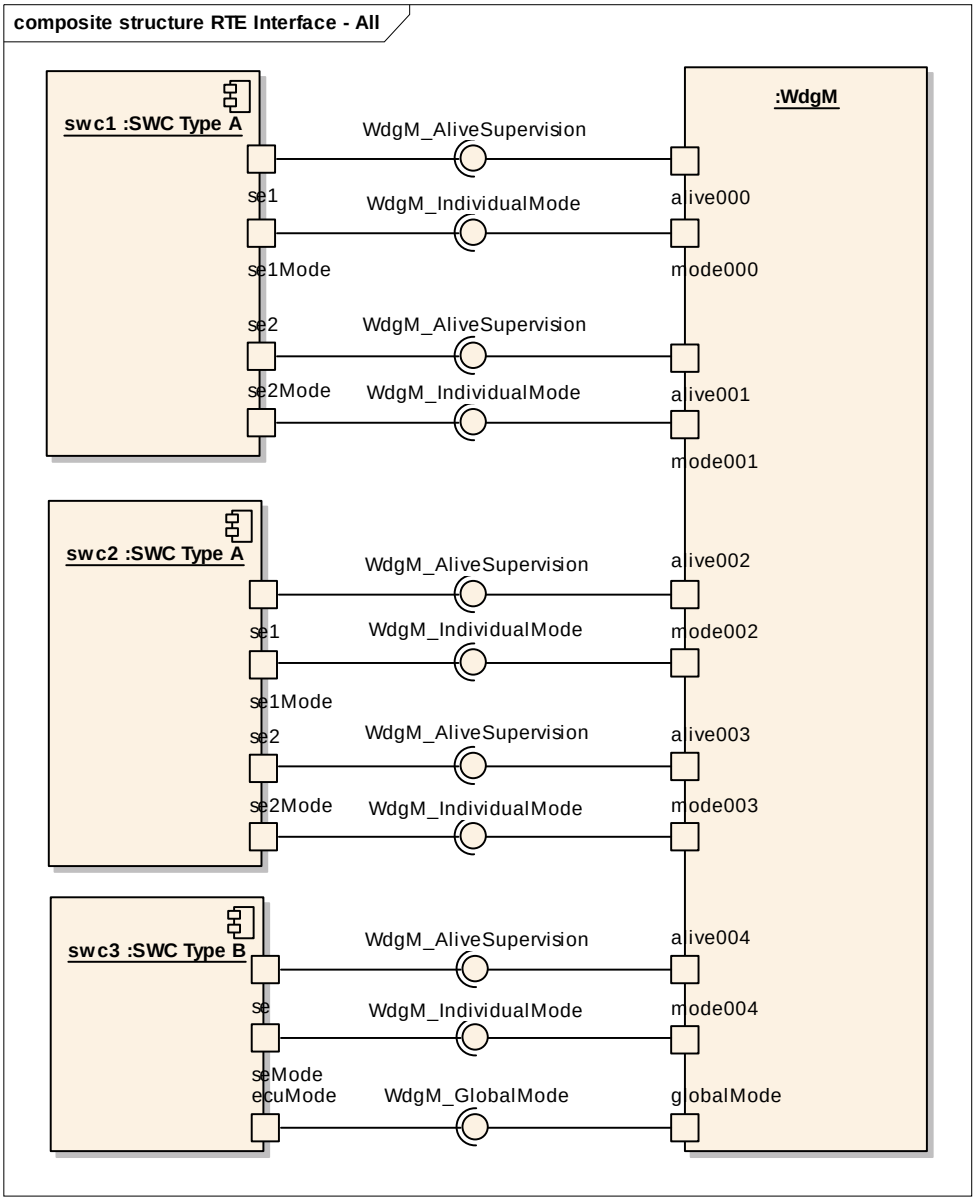

Figure 11: Example of SW-Cs connected to the Watchdog Manager via service ports and mode ports

This results in one mode port per supervised entity.

WDGM148: The Watchdog Manager shall provide a single mode port for reporting the 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.

WDGM149: The Status Reporting ports 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.

The number of supervised entities is defined in the configuration parameter WDGMNUMBEROFSUPERVISEDENTITIES.

Furthermore, the Watchdog Manager must be able to report the global supervision status.

WDGM150: The Watchdog Manager shall provide one mode port for reporting the combined, global supervision status of all supervised entities.

7.5.2.5 Error Codes

Mode ports are not able to signal any errors.

7.5.3 Additional services

The Watchdog Manager provides a set of additional APIs which have until now not been discussed.
These kinds of interfaces must be called from central parts within basic SW, e.g. the ECU State Manager. There is no use case for “normal” application SW components above the RTE to call these operations. Therefore the operations shall not be made available as service ports and interfaces.

7.5.4 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 UpdateAliveCounter
     symbol “WdgM_UpdateAliveCounter”
     canbeInvokedConcurrently = TRUE

RunnableEntity ActivateAliveSupervision
     symbol “WdgM_ActivateAliveSupervision”
     canbeInvokedConcurrently = TRUE

RunnableEntity DeactivateAliveSupervision
     symbol “WdgM_DeactivateAliveSupervision”
     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>};
```

7.5.5 Definition of the Watchdog Manager Service

This section shows the complete definition of the Watchdog Service. Note that these definitions can only be completed during ECU configuration (because it depends on certain configuration parameters of the Watchdog Manager which determine the number of ports provided by the Watchdog Manager Service). Also note that the implementation of a SW-C 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-Component, which uses the Service, must contain “service ports” in its own SW-C description which will be connected to the ports of the Watchdog Manager, so that the RTE can be generated.

```
/* 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 / ECU integrator */
Service WdgM {
     // For each supervised entity the Watchdog Manager
     // provides a port to update the alive counter and
     // to switch on and off alive supervision
     ProvidePort WdgM_AliveSupervision alive000;
     …
     ProvidePort WdgM_AliveSupervision alive<#SE-1>;

     // For each supervised entity the Watchdog Manager
     // provides a mode port to signal the individual
     // supervision status to interested SW-Cs 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 and the RTE
     ProvidePort WdgM_GlobalMode globalMode;
```

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

    RunnableEntity ActivateAliveSupervision
        symbol “WdgM_ActivateAliveSupervision”
            canbeInvokedConcurrently = TRUE

    RunnableEntity DeactivateAliveSupervision
        symbol “WdgM_DeactivateAliveSupervision”
        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>};
    };
};
```

7.6 Error classification

WDGM004: The Watchdog Manager shall be able to detect the following errors and exceptions depending on its configuration (development / production mode):

Type or error	*Relevance*	*Related error code*	*Value*
API service used in wrong context (without module initialization)	Development	`WDGM_E_NO_INIT`	`0x10`
API service called with "NULL pointer" parameter	Development	`WDGM_E_PARAM_CONFIG`	`0x11`
API service called with wrong "mode" parameter	Development	`WDGM_E_PARAM_MODE`	`0x12`
API service called with wrong "supervised entity identifier" parameter	Development	`WDGM_E_PARAM_SEID`	`0x13`
API service called with a null pointer parameter	Development	`WDGM_E_NULL_POINTER`	`0x14`
Disabling of watchdog not allowed (e.g. in safety relevant systems)	Development	`WDGM_E_DISABLE_NOT_ALLOWED`	`0x15`
Deactivation of alive-supervision for a supervised entity not allowed	Development	`WDGM_E_DEACTIVATE_NOT_ALLOWED`	`0x16`
Alive-supervision has failed and a watchdog reset will occur	Production	`WDGM_E_ALIVE_SUPERVISION`	`assigned by DEM`
Watchdog drivers' mode switch has failed	Production	`WDGM_E_SET_MODE`	`assigned by DEM`

WDGM128: Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.

7.7 Error detection

WDGM047: The detection of all development errors shall be configurable (on/off) at pre-compile time with the preprocessor switch WdgMDevErrorDetect.

WDGM015: The detection of production code errors cannot be switched off.

7.8 Error notification

WDGM048: Detected development errors shall be reported to the *`Det_ReportError`* service of the Development Error Tracer (DET) if the pre-processor switch *`WdgMDevErrorDetect`* is set (see chapter 10).

WDGM006: Production relevant errors shall be reported to the Diagnostic Event Manager (DEM).

WDGM022: The production error code “WDGM_E_ALIVE_SUPERVISION” shall be reported to DEM as an error status in case of alive-supervision failure because there is no way to heal the problem which will lead to a reset of the ECU.

WDGM138: The production error code “WDGM_E_SET_MODE” shall be reported to DEM as an error status in case of mode switch failure because the watchdog will not work as expected.

8 API specification

8.1 Imported types

WDGM011: The Watchdog Manager shall use only the following imported types of other modules:

Header file	*Imported Type*
Gpt_Types.h	Gpt_ValueType
	Gpt_ChannelType
Dem_Types.h	Dem_EventIdType
Std_Types.h	Std_VersionInfoType
	Std_ReturnType
WdgIf_Types.h	WdgIf_ModeType

8.2 Type definitions

WDGM038: The following Data Types shall be used for the functions defined in this specification.

8.2.1 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.	

WDGM042: The structure WdgM_ConfigType shall contain all post-build configurable parameters of the Watchdog Manager. The exact content of this structure depends on the selected configuration variant.

See Chapter 10.2 for information on configuration parameters.

8.2.2 WdgM_SupervisedEntityIdType

Name:	`WdgM_SupervisedEntityIdType`	
Type:	`uint8,uint16`	
Range:	`0-255, 0-65535`	The range of valid IDs depends on configuration and on the chosen platform type.
Description:	This type identifies an individual Supervised Entity for the Watchdog Manager.	

8.2.3 WdgM_ModeType

Name:	WdgM_ModeType	
Type:	uint8	
Range:	0-255	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.	

8.2.4 WdgM_AliveSupervisionStatusType

Name:	WdgM_AliveSupervisionStatusType		
Type:	uint8		
Range:	WDGM_ALIVE_OK	0	Timing constraints are fulfilled within the configured margins.
	WDGM_ALIVE_FAILED	1	Timing constraints have been violated including the margins, but the amount of failed supervision reference cycles has not been exceeded.
	WDGM_ALIVE_EXPIRED	2	Timing constraints have been violated including the margins for more often than the acceptable amount of failed supervision reference cycles.
	WDGM_ALIVE_STOPPED	3	Timing constraints have expired and the time limit to postpone the blocking of watchdog triggering is exceeded. (This value is only possible for the global supervision status.)
	WDGM_ALIVE_DEACTIVATED	4	Alive-supervision is disabled for this supervised entity. (This value is only possible for the individual supervision status.)
Description:	This type shall be used for variables that represent the individual supervision status of the alive-supervision of individual supervised entities. It also shall be used for the variable that represents the global supervision status of the Watchdog Manager.		

8.3 Function definitions

WDGM044: All APIs implemented by the BSW module Watchdog Manager shall respect naming rules defined by BSW00310.

WDGM050: BSW module Watchdog Manager shall not use generic interfaces.

8.3.1 WdgM_Init

WDGM151:

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.	

WDGM001: This routine initializes the Watchdog Manager. After execution of this routine, alive-supervision is activated according to the list of supervised entities defined in the configuration.

WDGM018: This routine shall initialize all module global variables.

WDGM135: This routine shall establish the initial mode of the Watchdog Manager.

The behavior in case the initial mode cannot be established is described in [WDGM139].

Since the Watchdog Manager is initialized very early during startup, where OS tasks are not yet running, it is recommended to configure the initial mode to use GPT triggering.

WDGM010: If development error detection is enabled, the contents of the given configuration set shall be checked for being within the allowed boundaries. If an error is detected the initialization of the Watchdog Manager shall not be executed and the error shall be reported to the Development Error Tracer with the value `WDGM_E_PARAM_CONFIG`.

WDGM030: If disabling the watchdog is not allowed by the parameter WdgMOffModeEnabled (e.g. in safety relevant systems), the routine shall check if the initial mode given in the provided configuration set will disable the watchdog (WDGIF_OFF_MODE). In this case the initialization routine shall not be executed and if development error detection is enabled, the error shall be reported to the Development Error Tracer with the value `WDGM_E_DISABLE_NOT_ALLOWED`.

8.3.2 WdgM_GetVersionInfo

WDGM153:

Service name:	WdgM_GetVersionInfo	
Syntax:	`void WdgM_GetVersionInfo( Std_VersionInfoType* VersionInfo )`	
Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Non 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.	

WDGM110: This service returns the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

WDGM111: This function shall be pre compile time configurable `On/Off` by the configuration parameter `WdgMVersionInfoApi`.

8.3.3 WdgM_SetMode

WDGM154:

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

WDGM062: The Watchdog Manager shall provide the service "*WdgM_SetMode*" to switch between different modes.

The behavior of this service and the corresponding functional requirements are described in chapter 7.3.4.

WDGM142: If the service fails, the Watchdog Manager shall report to the Diagnostic Event Manager an error with the value `WDGM_E_SET_MODE`.

WDGM020: If development error detection 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`.

WDGM021: If development error detection 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`.

WDGM031: If disabling the watchdog is not allowed by the parameter WDGMOFFMODEENABLED (e.g. in safety relevant systems), 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 development error detection 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`.

8.3.4 WdgM_GetMode

WDGM168:

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.	

WDGM170: 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.

If development error detection 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`.

If development error detection 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_NULL_POINTER` and the routine shall return the value `E_NOT_OK`.

8.3.5 WdgM_UpdateAliveCounter

WDGM155:

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:	Gives alive indications to the Watchdog Manager.	

WDGM026: This routine shall be used by the supervised entities under control of the Watchdog Manager to give alive indications to the Watchdog Manager.

WDGM027: If development error detection 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`.

WDGM028: If development error detection 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`.

8.3.6 WdgM_ActivateAliveSupervision

WDGM156:

Service name:	WdgM_ActivateAliveSupervision	
Syntax:	`Std_ReturnType WdgM_ActivateAliveSupervision( WdgM_SupervisedEntityIdType SEId )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SEId	Identifier of the entity under control of the WdgM whose alive-supervision shall be activated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Successfully activated Alive Supervision E_NOT_OK: Activating Alive Supervision failed
Description:	Activates the alive-supervision of a considered entity.	

WDGM053: This routine shall be used to activate the alive-supervision of a considered entity. Obviously, this entity shall have been statically configured to be under supervision of the Watchdog Manager.

WDGM055: If development error detection 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`.

WDGM056: If development error detection 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`.

8.3.7 WdgM_DeactivateAliveSupervision

WDGM157:

Service name:	WdgM_DeactivateAliveSupervision	
Syntax:	`Std_ReturnType WdgM_DeactivateAliveSupervision( WdgM_SupervisedEntityIdType SEId )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SEId	Identifier of the entity under control of the WdgM whose alive-supervision shall be deactivated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Successfully deactivated Alive Supervision E_NOT_OK: Deactivating Alive Supervision failed
Description:	Deactivates the alive-supervision of a considered entity.	

WDGM054: This routine shall be used to deactivate the alive-supervision of a considered entity. Obviously, this entity shall be statically configured to be under supervision of the Watchdog Manager.

WDGM108: If deactivation access is not allowed for this supervised entity, the service shall not be executed and shall return the value `E_NOT_OK`.

WDGM174: If development error detection is enabled and deactivation access is not allowed for this supervised entity, the error shall be reported to the Development Error Tracer with the error code `WDGM_E_DEACTIVATE_NOT_ALLOWED` and the routine shall return the value `E_NOT_OK`.

WDGM057: If development error detection 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`.

WDGM058: If development error detection 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`.

8.3.8 WdgM_GetAliveSupervisionStatus

WDGM169:

Service name:	WdgM_GetAliveSupervisionStatus	
Syntax:	`Std_ReturnType WdgM_GetAliveSupervisionStatus( WdgM_SupervisedEntityIdType SEId, WdgM_AliveSupervisionStatusType* Status )`	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SEId	Identifier of the supervised entity whose status shall be returned.
Parameters (inout):	None	
Parameters (out):	Status	Status of the given supervised entity.
Return value:	Std_ReturnType	E_OK: Current Alive Supervision Status successfully returned E_NOT_OK: Returning current Alive Supervision Status failed
Description:	--	

WDGM171: The WdgM_GetAliveSupervisionStatus service shall return the individual supervision status of the given supervised entity.

WDGM172: If development error detection 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`.

WDGM173: If development error detection 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`.

8.3.9 WdgM_GetGlobalStatus

WDGM175:

Service name:	WdgM_GetGlobalStatus
Syntax:	`Std_ReturnType WdgM_GetGlobalStatus( WdgM_AliveSupervisionStatusType* Status )`
Service ID[hex]:	0x0d
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	Status	Global supervision status of the Watchdog Manager.
Return value:	Std_ReturnType	E_OK: Current Alive Supervision Status successfully returned E_NOT_OK: Returning current Alive Supervision Status failed
Description:	--	

WDGM176: If development error detection 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`.

8.4 Call-back notifications

8.4.1 WdgM_Cbk_GptNotification

WDGM165:

Service name:	WdgM_Cbk_GptNotification

Syntax:	`void WdgM_Cbk_GptNotification(`
	
	`)`

Service ID[hex]:	0x0a
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	--

8.5 Scheduled functions

WDGM043: These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

Note: The concrete schedule-periods for usage of following services have to be derived from project context, but the following hints could be considered:
1. The cycle period watchdog-triggering service needs to be scheduled as often as the hardware watchdog requires it.
2. The alive-supervision needs to be scheduled with a cyclic period, that a delay of one period still allows a complete recovery of program execution by a reset power-on scenario without exceeding the maximum time-tolerance of a hanging application.
And considering additional latency of:
- one cycle period of watchdog-triggering service (max. latency to stop triggering)
- one configured time-period of hardware watchdog before it expires

WDGM049: Main processing function of the BSW module Watchdog Manager shall respect naming rules defined by BSW00373.

8.5.1 WdgM_MainFunction_AliveSupervision

WDGM159:

Service name:	WdgM_MainFunction_AliveSupervision

Syntax:	`void WdgM_MainFunction_AliveSupervision(`
	
	`)`

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

WDGM060: This main function shall perform the processing of the Watchdog Manager jobs (check the aliveness of the application entities under supervision of the Watchdog Manager).
This routine shall be scheduled according to the timing constraints defined with parameter WdgMSupervisionCycle.

WDGM061: This function shall check the alive counter values of each entity under control of the Watchdog Manager to verify that their timing constraints are respected. Values of the alive counters must be within min and max values provided in the configuration set.

WDGM024: When a supervised entity is detected as faulty (its alive counter value doesn't correspond to its timing constraints), its supervision status is set to "WDGM_ALIVE_FAILED" if the allowed number of failed supervision cycles is different from 0 and to "WDGM_ALIVE_EXPIRED" otherwise. The supervision status is then forwarded to the RTE for potential fault reaction to perform at application layer.

WDGM039: If development error detection 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`.

8.5.2 WdgM_MainFunction_Trigger

WDGM160:

Service name:	WdgM_MainFunction_Trigger

Syntax:	`void WdgM_MainFunction_Trigger(`
	
	`)`

Service ID[hex]:	0x09
Timing:	VARIABLE_CYCLIC
Description:	Invokes the triggering of a watchdog instance.

WDGM067: This service shall be scheduled according to the timing constraints defined with parameter WdgMTriggerCycle.

WDGM040: This service shall include an internal prescaling mechanism to derive trigger periods for each assigned watchdog instance in each mode, according to parameter WdgMTriggerReferenceCycle of each watchdog instance.

WDGM041: This service shall invoke the triggering of a watchdog instance if its trigger reference cycle is reached and the global supervision status is different from WDGM_ALIVE_STOPPED.

WDGM068: If development error detection is enabled, the routine shall check if the Watchdog Manager is initialized. In case of an error the trigger service shall not be executed, the error shall be reported to the Debug Error Tracer with the error code `WDGM_E_NO_INIT`.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

WDGM161:

API function	***Description***
Gpt_DisableNotification	Disables the notification for a channel.
Gpt_EnableNotification	Enables the notification for a channel.
Gpt_StopTimer	Stops a timer channel.
Gpt_StartTimer	Starts a timer channel.
WdgIf_SetMode	--

WdgIf_Trigger	--
Dem_ReportErrorStatus	Reports errors to the DEM.

8.6.2 Optional Interfaces

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

WDGM162:

API function	***Description***
Det_ReportError	Service to report development errors.
Mcu_PerformReset	The service performs a microcontroller reset.

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

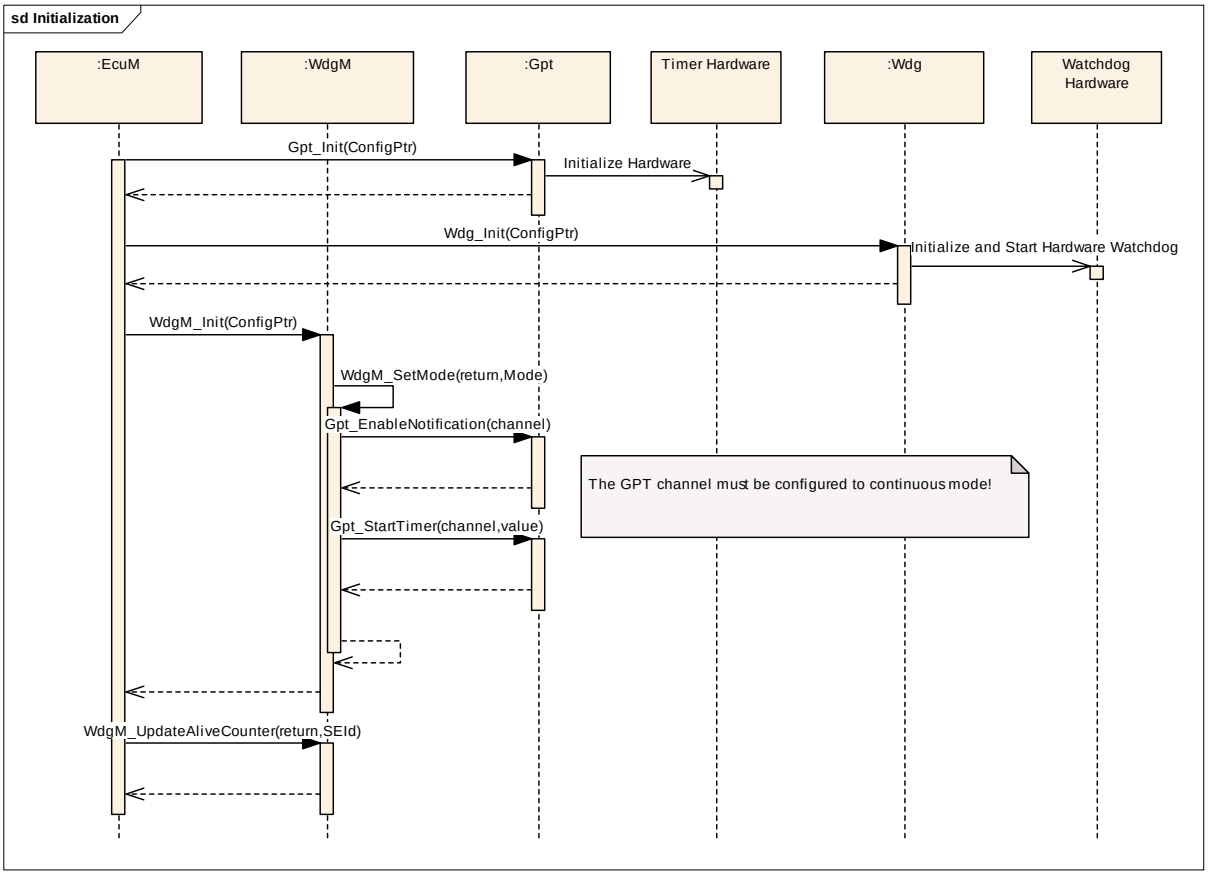

Figure 12: Watchdog Manager Initialization

First, the ECU State Manager initializes the GPT. Second, it initializes the Watchdog Driver and thereby starts the watchdog hardware. Note that for multiple watchdogs multiple Watchdog Drivers must be initialized. Third, the ECU State Manager initializes the Watchdog Manager. During initialization the Watchdog Manager assumes that the Watchdog Driver and the General Purpose Timer are already initialized. Since the Watchdog Manager is initialized before the OS and the scheduler are running, the initial mode of the Watchdog Manager must rely on GPT activation.

9.2 GPT Activation

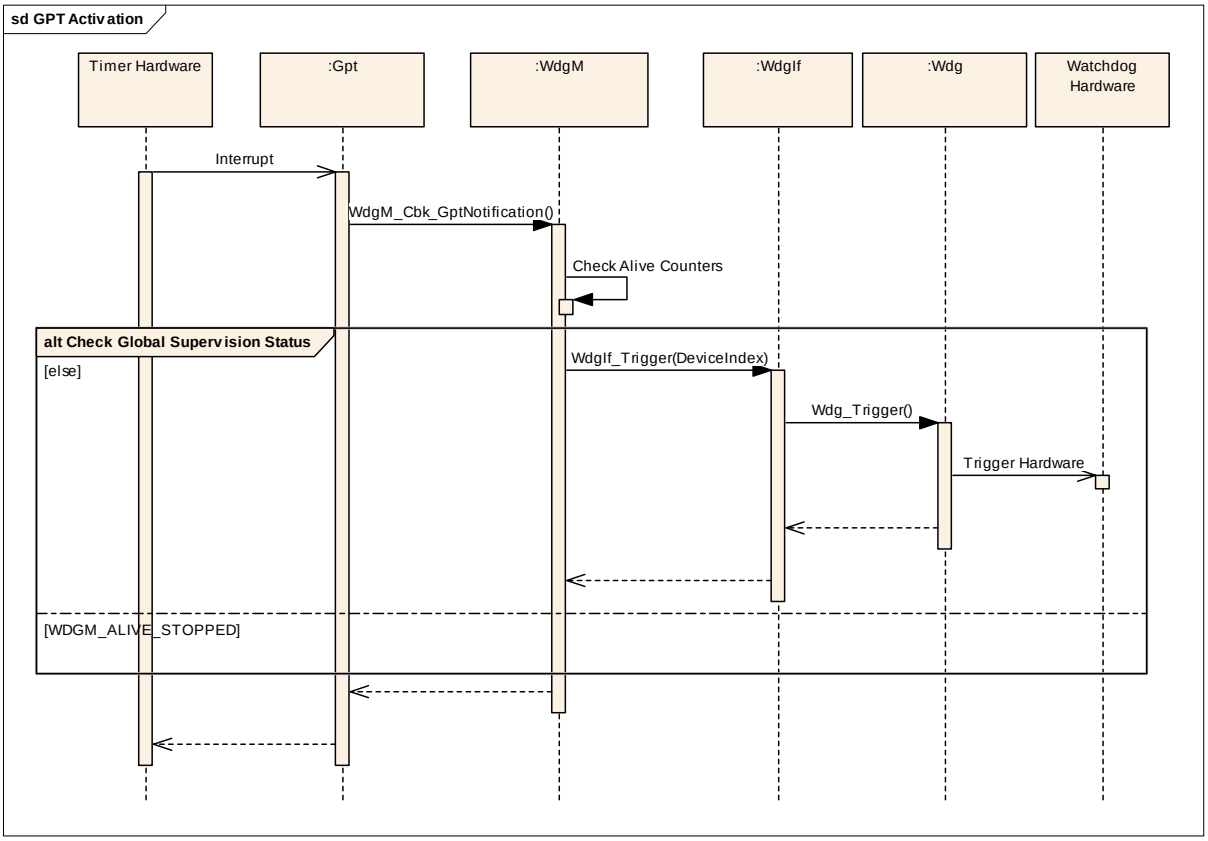

Figure 13: GPT Activation

When the Watchdog Manager is activated by the GPT, a callback notification is sent by the GPT. The Watchdog Manager then computes the individual and the global supervision status and triggers the watchdogs accordingly.

9.3 Startup and Shutdown Activities

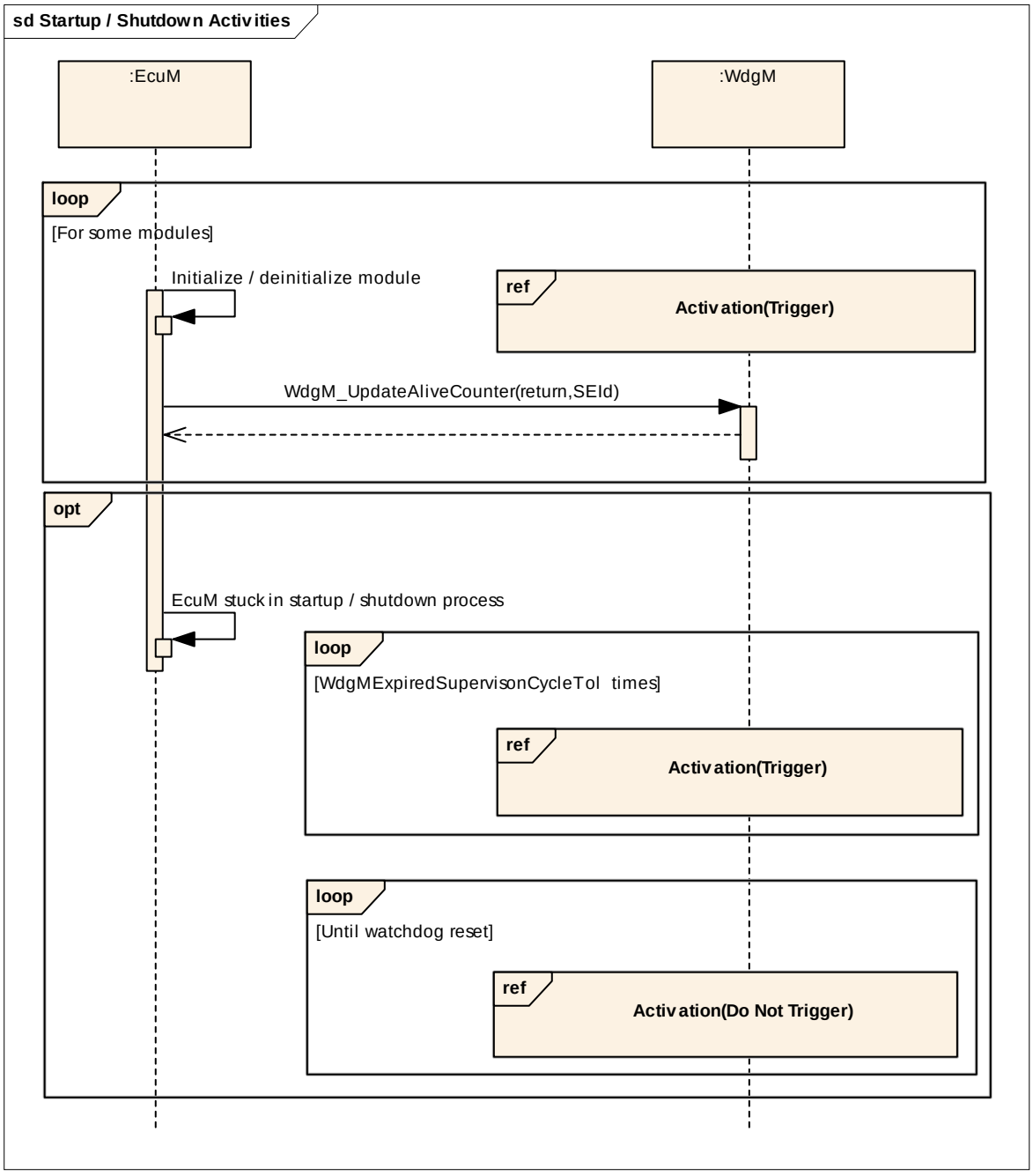

Figure 14: Startup and Shutdown Activities

During startup, shutdown and sleep the ECU State Manager is the only entity running. Therefore, it should also be the only Supervised Entity to be supervised by the Watchdog Manager. As long as the ECU State Manager updates its alive counter, the Watchdog Manager will trigger the watchdogs. If the ECU State Manager gets stuck at any point, the Watchdog Manager will recognize that (alive counter not updated) and will stop triggering the watchdogs, which in turn will cause a watchdog reset.

9.4 Switching from GPT to Main Function Activation

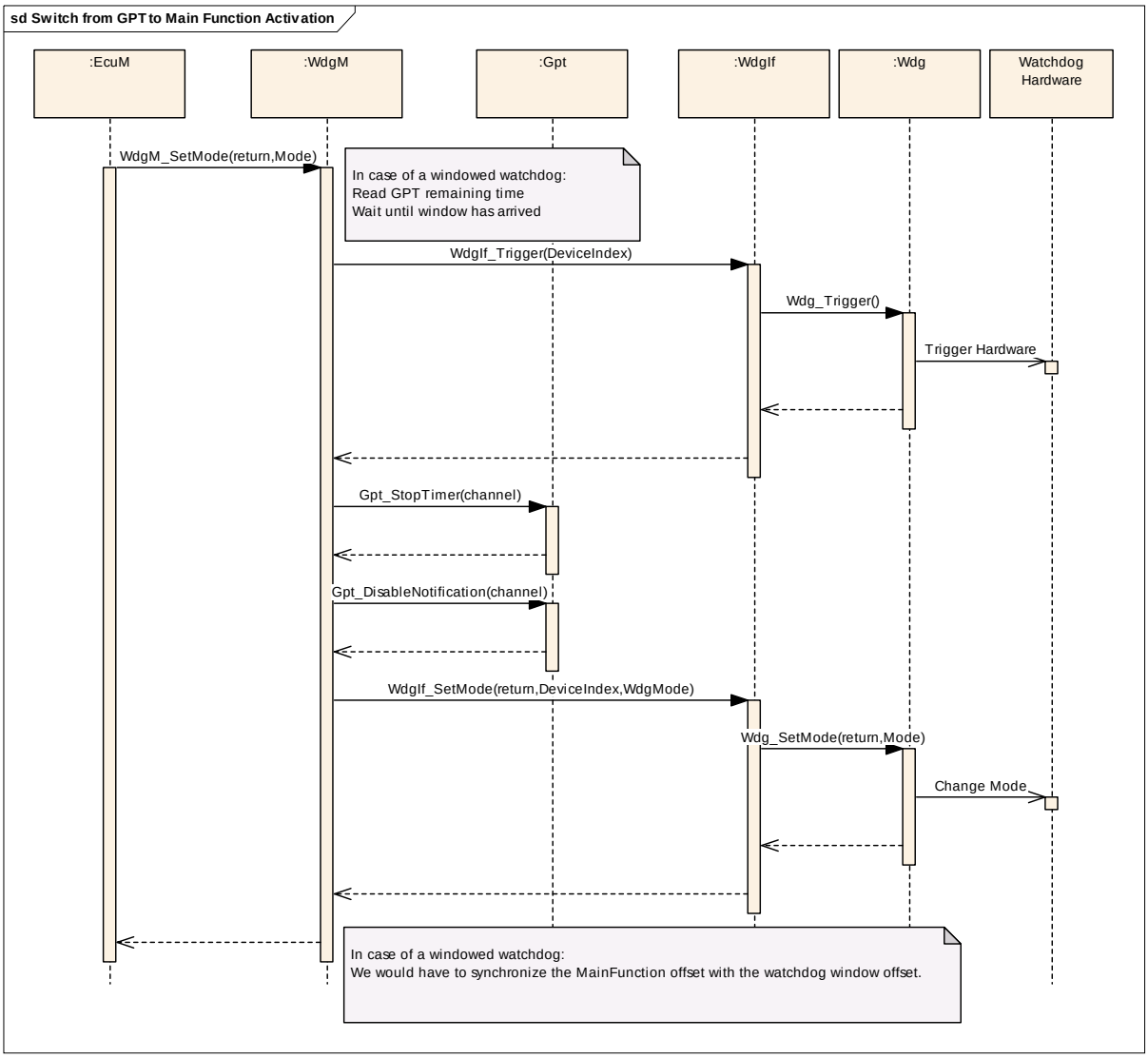

Figure 15: Switching from GPT to Main Function Activation

When the OS has been started the ECU State Manager will switch the mode of the Watchdog Manager. This mode now can use Main Function Activation to perform the Watchdog Manager tasks. For this switch Watchdog Manager has to stop the GPT and has to re-synchronize the watchdogs with the new schedule. If configured, the Watchdog Manager will also switch the watchdogs into different modes.

9.5 Main Function Activation

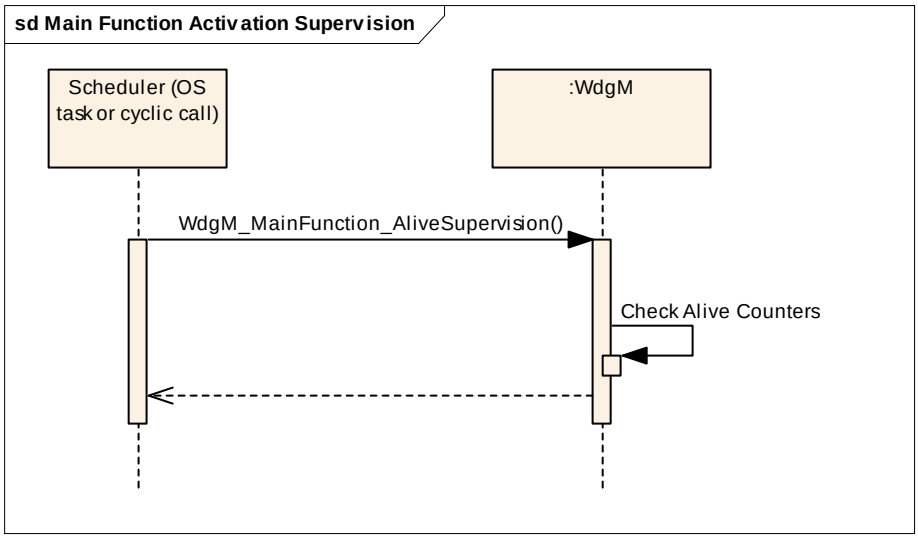

Figure 16: Main Function Activation for Alive-supervision

In Main Function activation the alive-supervision and the watchdog triggering are activated in separate tasks. The performed actions are the same as for GPT activation.

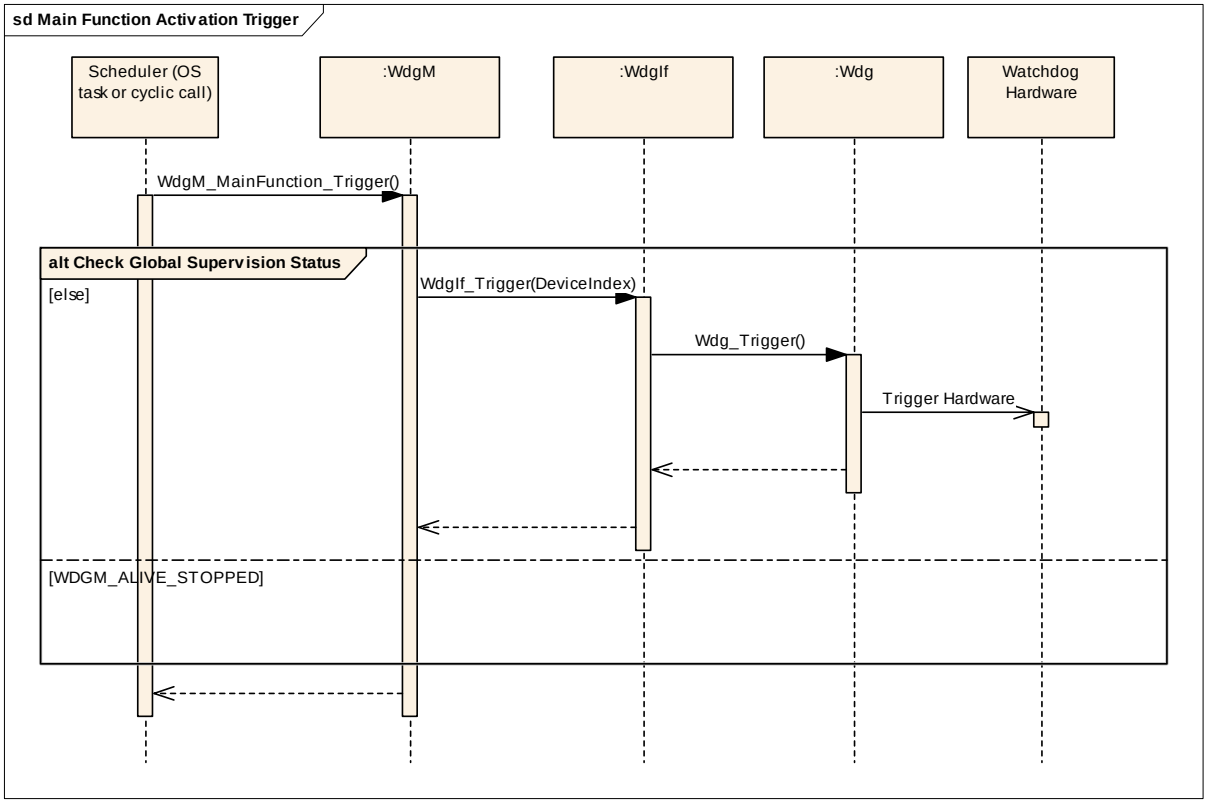

Figure 17: Main Function Activation for Watchdog Triggering

9.6 Switching from Main Function to GPT Activation

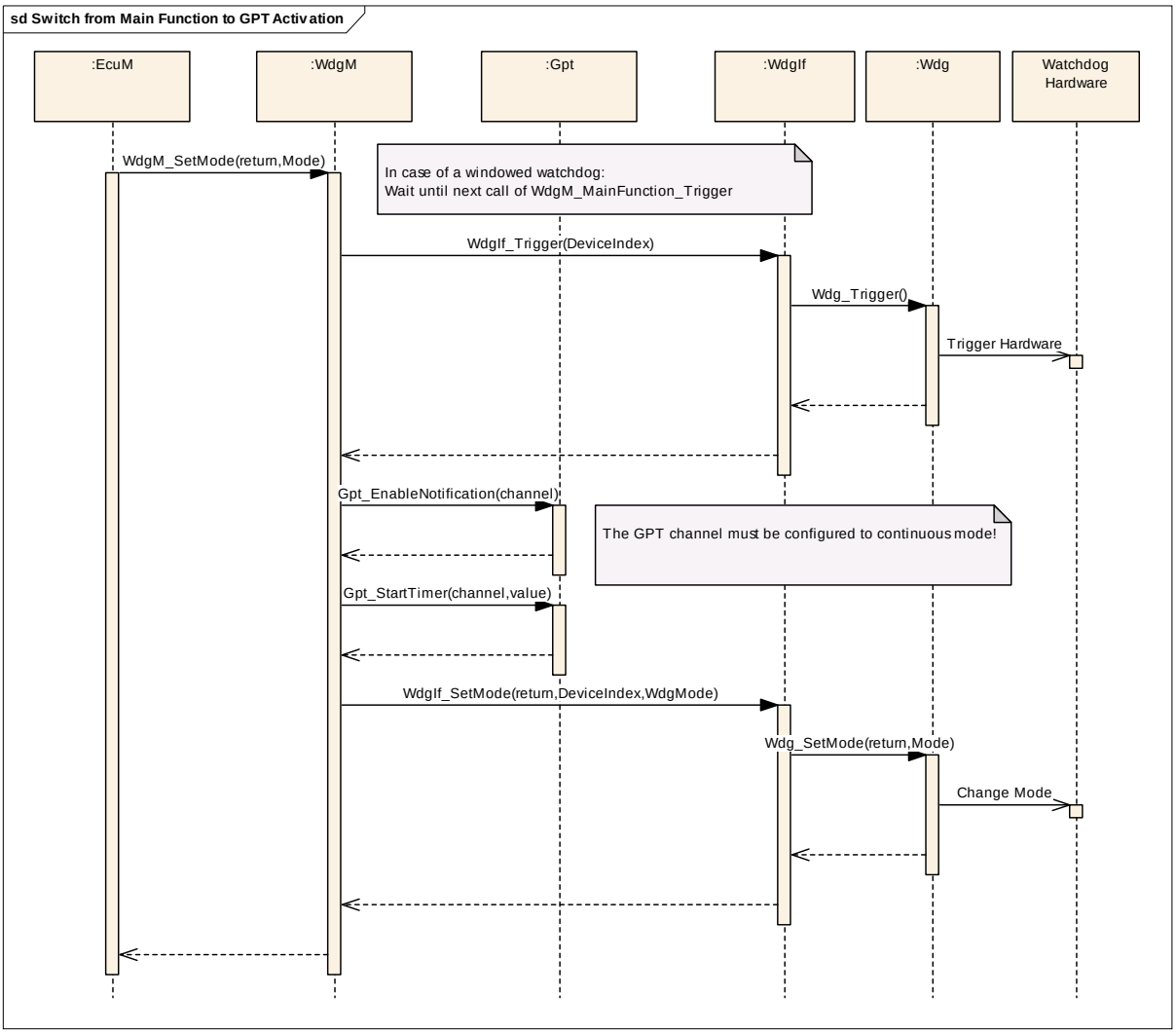

Figure 18: Switching from Main Function to GPT Activation

Similar to switching from GPT to Main Function activation when switching modes, it is also possible to switch back to GPT activation. This is necessary in when entering the shutdown and sleep phases where OS and scheduler are not active.

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.

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.1.1 Static configuration parameters

WDGM025: The parameters 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.

10.1.2 Runtime configuration parameters

WDGM029: The parameters that shall be configurable during runtime, 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

WDGM104: 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 Watchdog Manager according to settings of static configuration. They should be located at the module's configuration header file WdgM_Cfg.h

10.2 Containers and configuration parameters

10.2.1 Variants

VARIANT-PRE-COMPILE: This variant contains only pre-compile time configuration parameters.

VARIANT-POST-BUILD: This variant is a mixture of pre-compile time and post build time configuration parameters.

10.2.2 WdgM

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 structures all general configuration parameters of the Watchdog Manager.

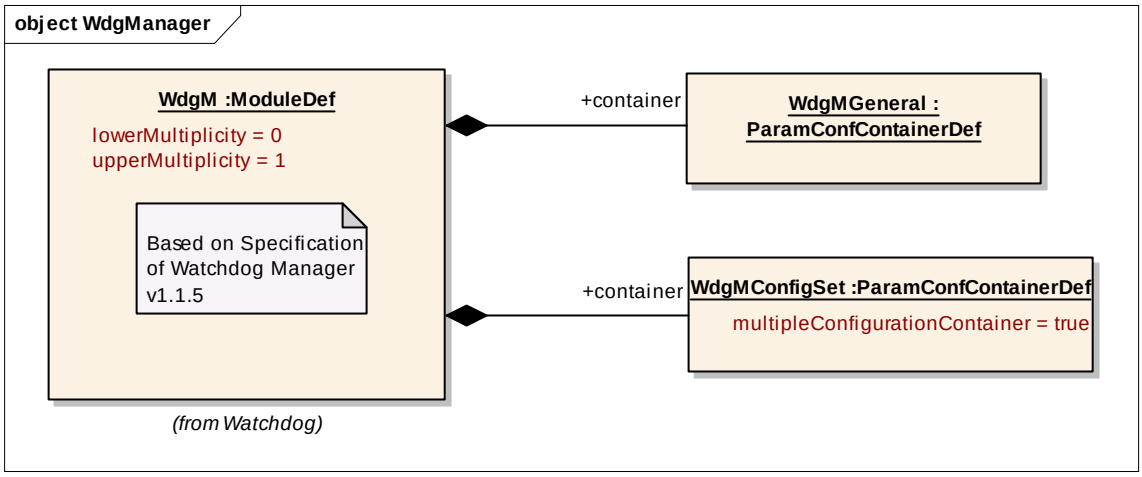

Figure 19: Configuration Module WdgM

10.2.3 WdgMGeneral

SWS Item	**--**
Container Name	WdgMGeneral{WdgMConfiguration}
Description	Container structures all general configuration parameters of the Watchdog Manager.
Configuration Parameters	

SWS Item	**--**		
Name	WdgMAliveSupervisionEnabled {WDGM_ALIVE_SUPERVISION}		
Description	Parameter to enable/disable the alive-supervision mechanism. Use Case: no supervised entity assigned true: Alive-supervision mechanism is enabled false: Alive-supervision mechanism is disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: WDGM052		

SWS Item	**--**		
Name	WdgMDemAliveSupervisionReport {WDGM_DEM_ALIVE_SUPERVISION_REPORT}		
Description	Parameter to enable/disable alive-supervision error reporting to DEM. true: Alive-supervision error reporting is enabled false: Alive-supervision error reporting is disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: WDGM129		

SWS Item	**--**		
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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: WDGM047, WDGM048		

SWS Item	**--**
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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: WDGM132		

SWS Item	**--**		
Name	WdgMNumberOfModes		
Description	This parameter shall contain the number of configured modes of the Watchdog Manager.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: WDGM177		

SWS Item	**--**		
Name	WdgMNumberOfSupervisedEntities {WDGM_NUMBER_OF_SUPERVISED_ENTITIES}		
Description	This parameter shall contain the amount of supervised entities configured for the Watchodg Manager.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**--**		
Name	WdgMNumberOfWatchdogs {WDGM_NUMBER_OF_WATCHDOG_INSTANCES}		
Description	This parameter shall contain the maximum amount of watchdogs to configured for Watchdog Manager.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: WDGM002		

SWS Item	**--**
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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	dependency: WDGM030, WDGM031		

SWS Item	**--**		
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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: WDGM111		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
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.

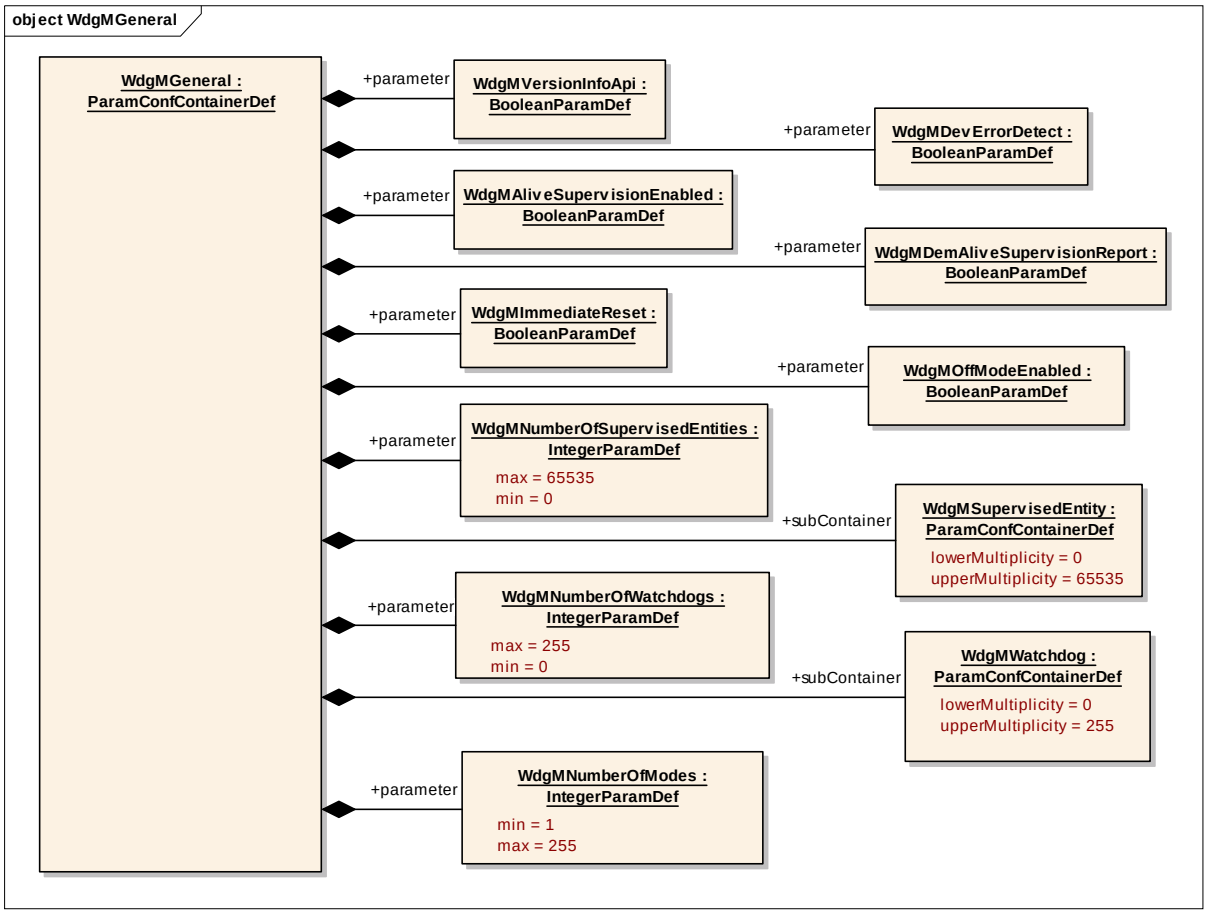

Figure 20: Configuration Container WdgMGeneral

10.2.4 WdgMSupervisedEntity

SWS Item	**--**
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	**--**		
Name	WdgMDeactivationAccessEnabled {WDGM_DEACTIVATION_ACCESS_ENABLED}		
Description	This parameter shall be used to grant the access of the service WdgM_DeactivateAliveSupervision() for affecting the activation status of the corresponding Supervised Entity.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: instance dependency: WDGM093		

SWS Item	**--**		
Name	WdgMSupervisedEntityId {WDGM_SUPERVISED_ENTITY_ID}		
Description	This parameter shall contain the identifier of he supervised entity.		
Multiplicity	1		
Type	IntegerParamDef (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: instance dependency: WDGM046		

No Included Containers

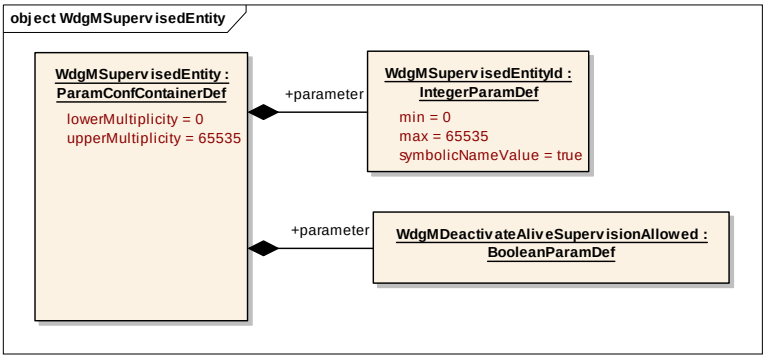

Figure 21: Configuration Container WdgMSupervisedEntity

10.2.5 WdgMWatchdog

SWS Item	**--**
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	**--**		
Name	WdgMWatchdogName {WDGM_WATCHDOG_INSTANCE_ID}		
Description	This parameter shall contain the symbolic name of the watchdog instance.		
Multiplicity	1		
Type	StringParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: instance dependency: WDGM003		

SWS Item	**--**
Name	WdgMDeviceRef
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_Trigger calls for that watchdog instance.		
Multiplicity	1		
Type	Reference to WdgIfDevice		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	dependency: WDGM003		

No Included Containers

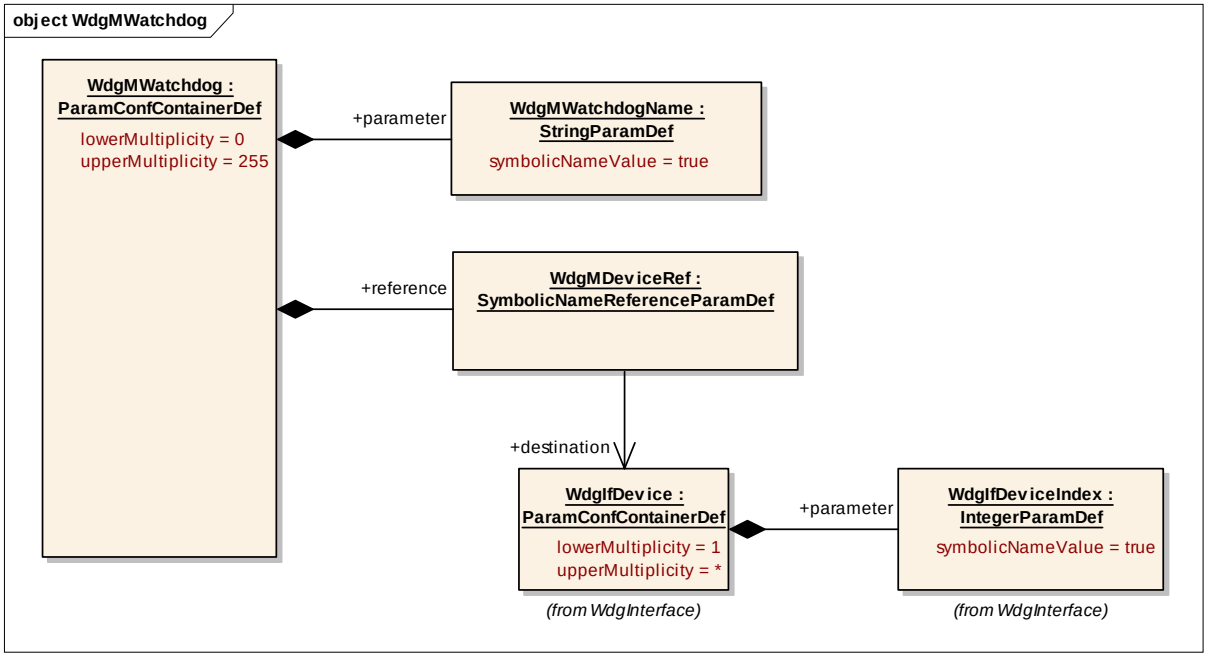

Figure 22: Configuration Container WdgMWatchdog

10.2.6 WdgMConfigSet

SWS Item	**--**
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	**--**		
Name	WdgMInitialMode		
Description	The mode that the Watchdog Manager is in after it has been initialized.		
Multiplicity	1		
Type	Reference to WdgMMode		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: WDGM179		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMMode	1..255	The container describes one of several modes of the Watchdog Manager.

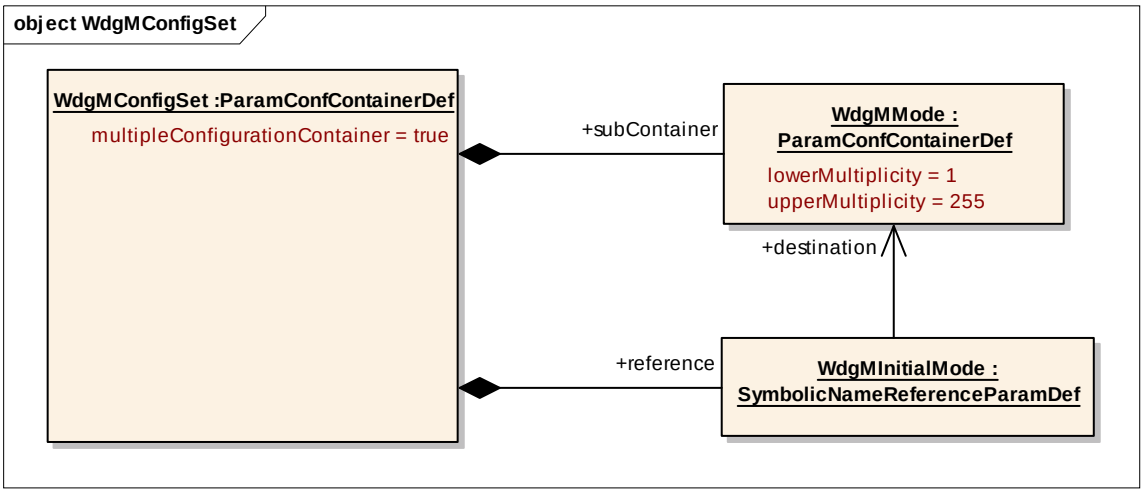

Figure 23: Configuration Container WdgMConfigSet

10.2.7 WdgMMode

SWS Item	--
Container Name	WdgMMode{WDGM_MODE}
Description	The container describes one of several modes of the Watchdog Manager.
Configuration Parameters	

SWS Item	--		
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.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: WDGM118		

SWS Item	--	
Name	WdgMModeId	
Description	This parameter fixes the identfier for the mode. This identifier is for instance passed as a parameter to the WdgM_SetMode service.	
Multiplicity	1	
Type	IntegerParamDef (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	dependency: WDGM178		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMActivation	1	This container allows to choose between Main Function activation and GPT activation.
WdgMAliveSupervision	0..65535	This container collects all configuration parameters of Alive-Supervision.
WdgMTrigger	0..255	This container collects all configuration parameters for the triggering of hardware watchdogs.

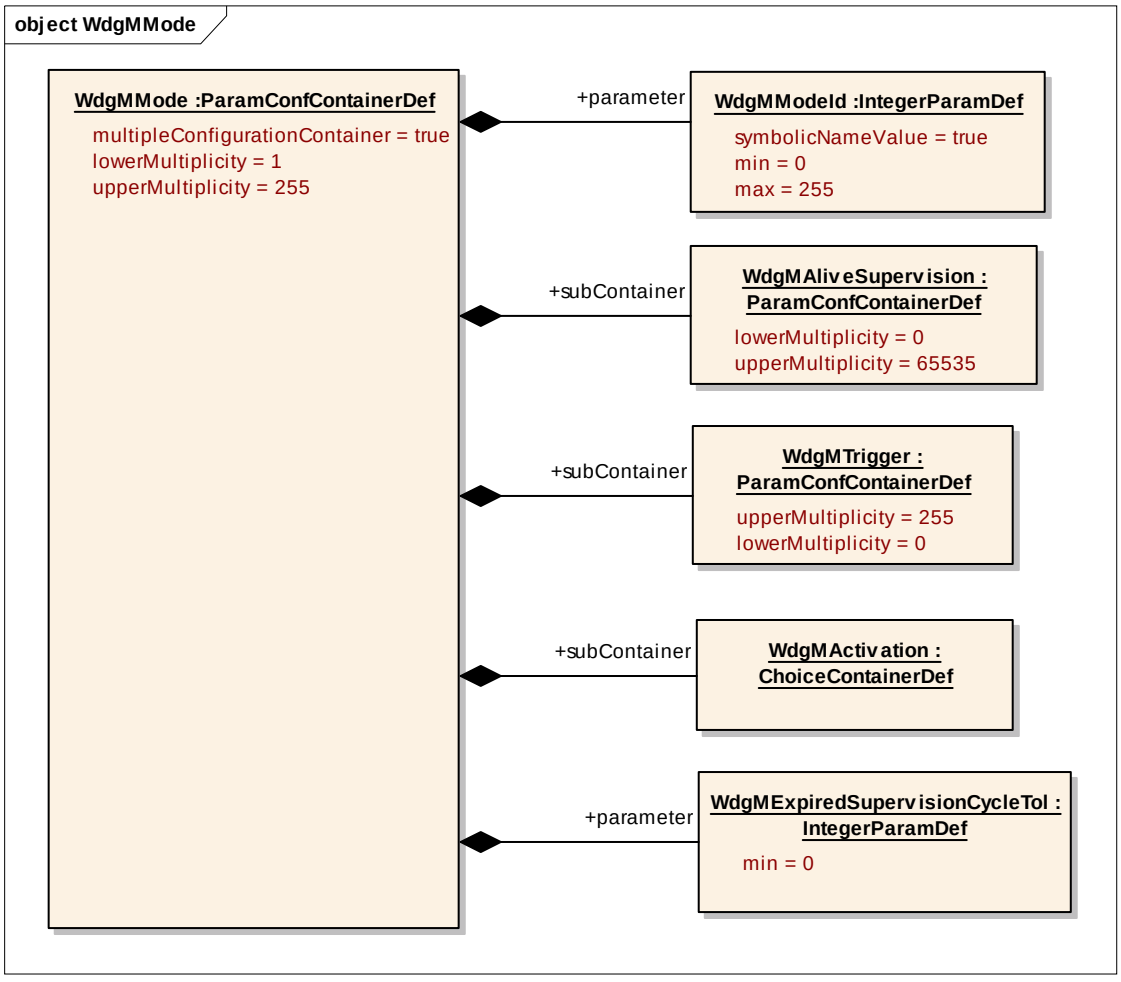

Figure 24: Configuration Container WdgMMode

10.2.8 WdgMAliveSupervision

SWS Item	**--**
Container Name	WdgMAliveSupervision{WdgMAliveSupervision}
Description	This container collects all configuration parameters of Alive-Supervision.
Configuration Parameters	

SWS Item	**--**		
Name	WdgMActivationActivated {WDGM_ACTIVATION_STATUS}		
Description	This parameter shall contain the activation status of the corresponding Supervised Entity.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance dependency: WDGM096		

SWS Item	**--**		
Name	WdgMAliveSupervisionConfigId {WDGM_ALIVE_SUPERVISION_CONFIG_ID}		
Description	This parameter shall contain the identifier of the alive-supervision configuration set.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
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: instance dependency: WDGM105		

SWS Item	**--**		
Name	WdgMExpectedAliveIndications {WDGM_EXPECTED_ALIVE_INDICATIONS}		
Description	This parameter shall contain the amount of expected alive indications within the referenced amount of defined supervision cycles according to corresponding SE.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance dependency: WDGM090		

SWS Item	**--**		
Name	WdgMFailedSupervisionRefCycleTol {WDGM_FAILED_SUPERVISION_REFERENCE_CYCLE_TOLERANCE}		
Description	This parameter shall contain the acceptable amount of failed supervision reference cycles that shall be used by checkup of alive supervision according to corresponding SE.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance		

	dependency: WDGM095

SWS Item	**--**		
Name	WdgMMaxMargin {WDGM_MAX_MARGIN}		
Description	This parameter shall contain the amount of alive indications that are acceptable to be additional on the expected alive indications within the corresponding supervision reference cycle.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance dependency: WDGM091		

SWS Item	**--**		
Name	WdgMMinMargin {WDGM_MIN_MARGIN}		
Description	This parameter shall contain the amount of alive indications that are acceptable to be missed from the expected alive indications within the corresponding supervision reference cycle.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance dependency: WDGM091		

SWS Item	**--**		
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	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance dependency: WDGM090		

SWS Item	**--**		
Name	WdgMSupervisedEntityRef		
Description	--		
Multiplicity	1		
Type	Reference to WdgMSupervisedEntity		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: WDGM105		

No Included Containers

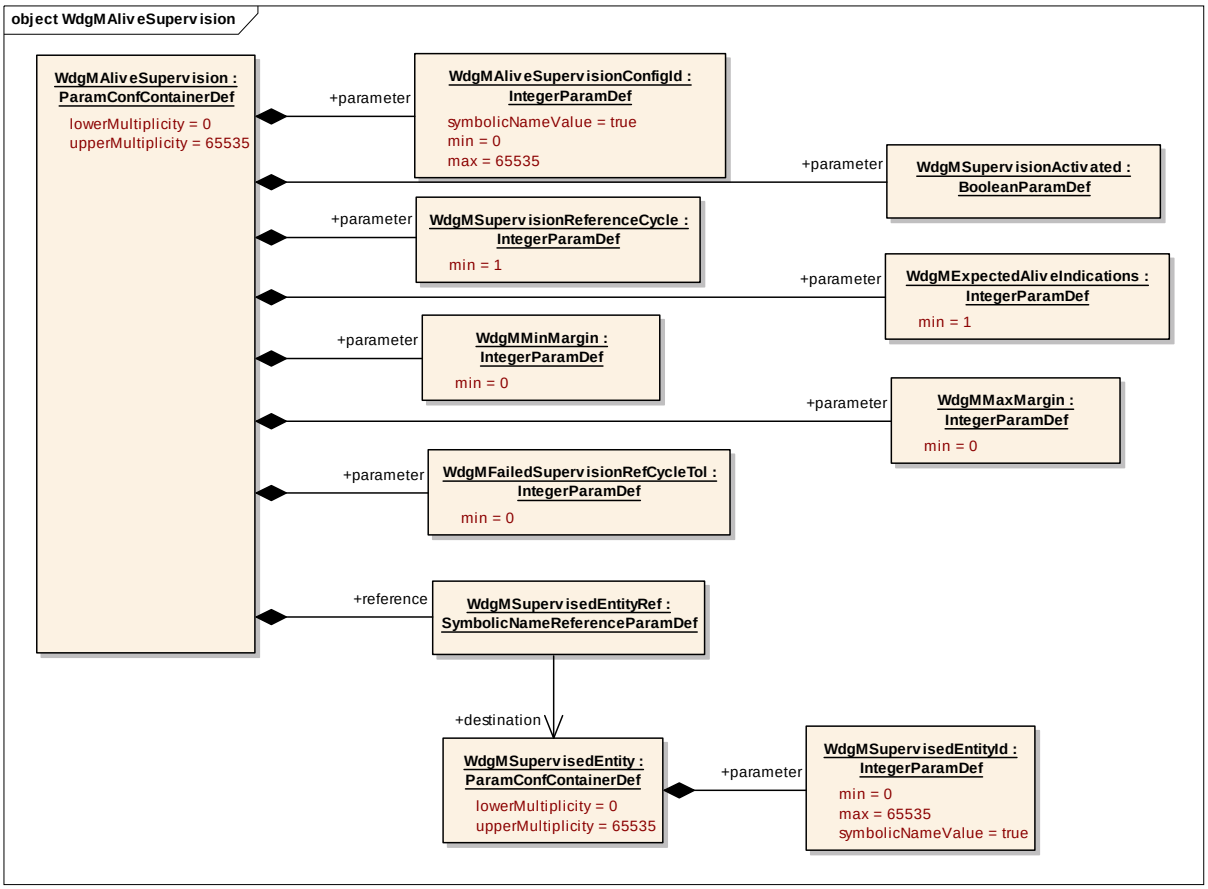

Figure 25: Configuration Container WdgMAliveSupervision

10.2.9 WdgMTrigger

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

SWS Item	**--**		
Name	WdgMTriggerReferenceCycle		
Description	This parameter shall contain the amount of trigger cycles to be used as reference by the trigger service of the Watchdog Manager to perform the triggering of the corresponding watchdog.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: instance dependency: WDGM116		

SWS Item	**--**		
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	EnumerationParamDef		
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	dependency: WDGM181		

SWS Item	**--**		
Name	WdgMWatchdogRef		
Description	This parameter is a reference to the configured watchdog.		
Multiplicity	1		
Type	Reference to WdgMWatchdog		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: WDGM180		

No Included Containers

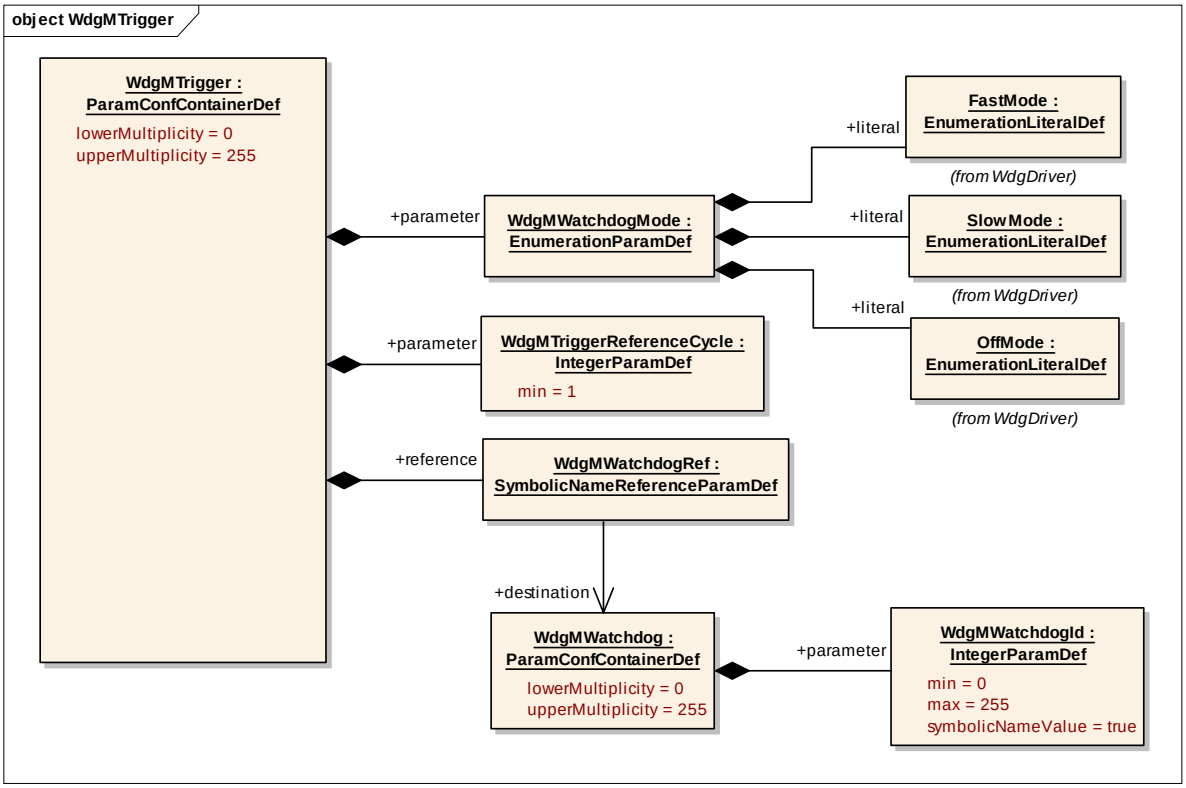

Figure 26: Configuration Container WdgMTrigger

10.2.10 WdgMActivation

SWS Item	**--**
Choice Container Name	WdgMActivation
Description	This container allows to choose between Main Function activation and GPT activation.

Container Choices		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMActivationGpt	0..1	GPT activation.
WdgMActivationSchM	0..1	Main Function activation.

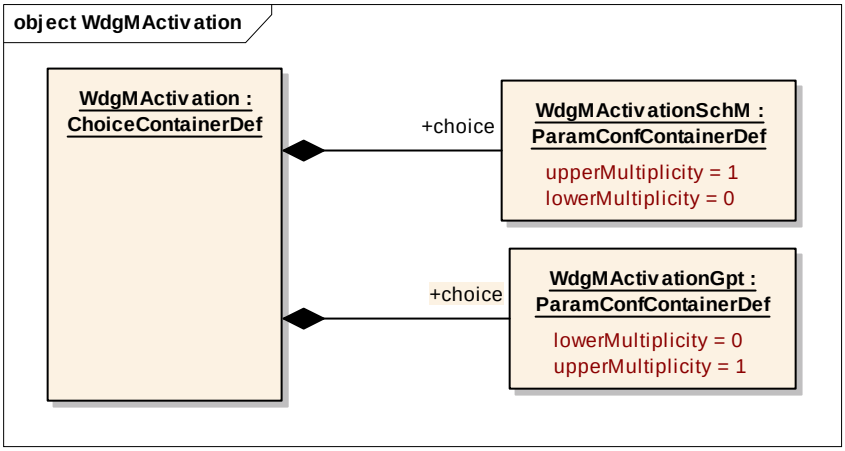

Figure 27: Configuration Container WdgMActivation

10.2.11 WdgMActivationSchM

SWS Item	**--**
Container Name	WdgMActivationSchM
Description	Main Function activation.
Configuration Parameters	

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

SWS Item	**--**
Name	WdgMTriggerCycle {WDGM_TRIGGER_CYCLE}
Description	This parameter defines the schedule period of the main function WdgM_MainFunction_Trigger. Unit: [s]
Multiplicity	1

Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU (integration)		

No Included Containers

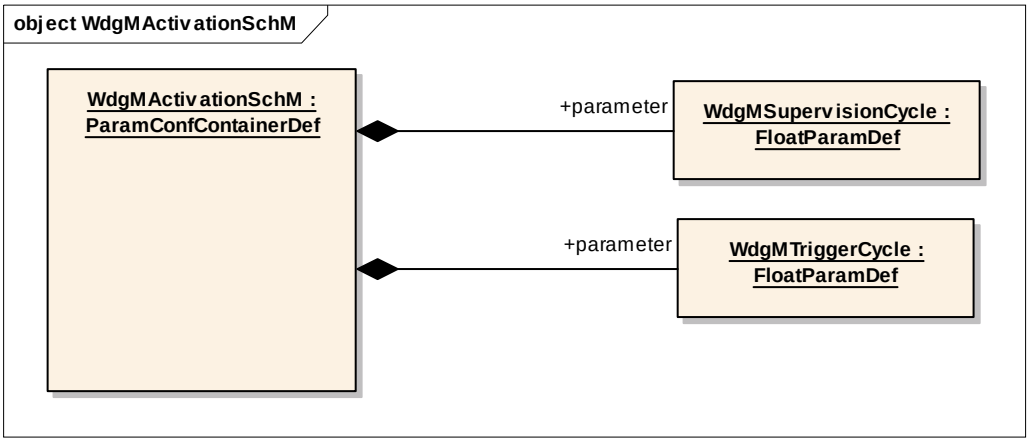

Figure 28: Configuration Container WdgMActivationSchM

10.2.12 WdgMActivationGpt

SWS Item	**--**
Container Name	WdgMActivationGpt
Description	GPT activation.
Configuration Parameters	

SWS Item	**--**		
Name	WdgMGptCycle		
Description	This parameter defines the timeout value for GPT notifications to the Watchdog Manager.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: WDGM194		

SWS Item	**--**		
Name	WdgMGptChannelRef		
Description	Reference to the GPT channel to be used for Watchdog Manager activation.		
Multiplicity	1		
Type	Reference to GptChannelConfiguration		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: WDGM193		

No Included Containers

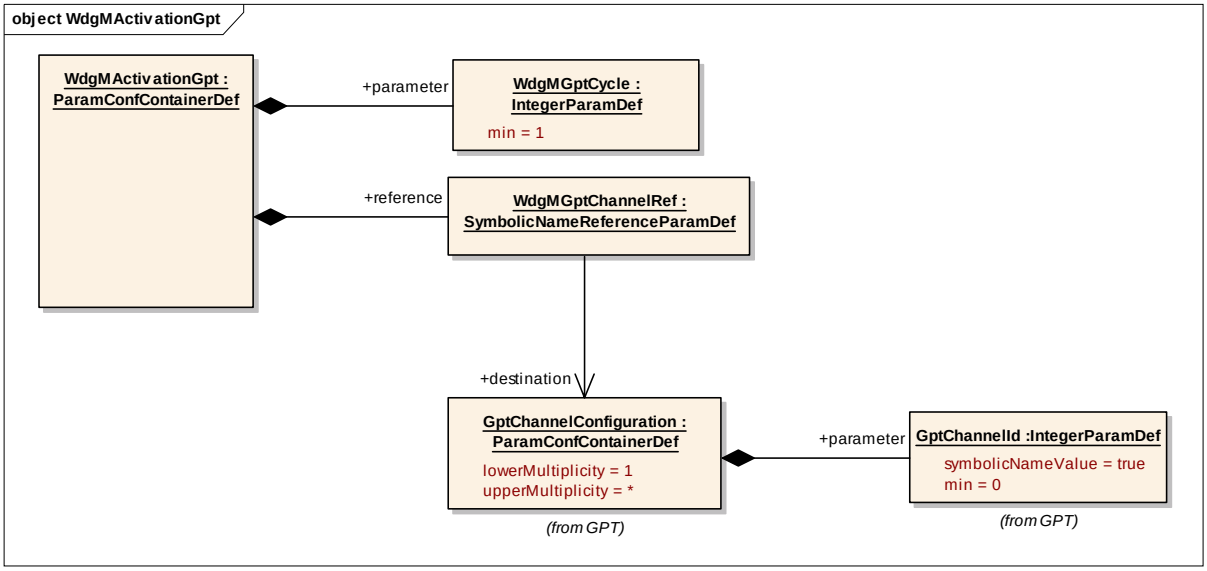

Figure 29: Configuration Container WdgMActivationGpt

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

WDGM013: The configuration parameters shall be checked statically (at least during compile time) for correctness. The version information in the module header and source files shall be validated and consistent (e.g. by comparing the version information in the module header and source files with a pre-processor macro).

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [7] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

10.4 Callback routines

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

11 Changes to Release 2.1

11.1 Deleted SWS Items

SWS Item	*Rationale*
WDGM019	Redundant requirement
WDGM033	Changed behavior when switching modes
WDGM069	Redundant requirement to WDGM080
WDGM106, WDGM107, WDGM112	Removed service WdgM_ChangeAliveSupervision
WDGM136	Synchronized init and mode switch behavior
WDGM140	Changed behavior when switching modes

11.2 Replaced SWS Items

None

11.3 Changed SWS Items

SWS Item	*Rationale*
WDGM011	All imported types are now in a generated table
WDGM116	Only one reference cycle per watchdog and mode needed.
WDGM139	Made behavior of mode switch safer in error cases
WDGM145	Precondition for mode switch is now more restrictive

11.4 Added SWS Items

SWS Item	*Rationale*
WDGM146 to WDGM150	Define ports
WDGM151	UML model linking of WdgM_Init
WDGM153	UML model linking of WdgM_GetVersionInfo
WDGM154	UML model linking of WdgM_SetMode
WDGM155	UML model linking of WdgM_UpdateAliveCounter
WDGM156	UML model linking of WdgM_ActivateAliveSupervision
WDGM157	UML model linking of WdgM_DeactivateAliveSupervision
WDGM159	UML model linking of WdgM_MainFunction_AliveSupervision
WDGM160	UML model linking of WdgM_MainFunction_Trigger
WDGM161	UML model linking of mandatory interfaces
WDGM162	UML model linking of optional interfaces
WDGM165	UML model linking of WdgM_Cbk_GptNotification
WDGM168	UML model linking of WdgM_GetMode
WDGM169	UML model linking of WdgM_GetAliveSupervisionStatus
WDGM170	Behavior of WdgM_GetMode
WDGM171 to WDGM173	Behavior of WdgM_GetAliveSupervisionStatus
WDGM174	Behavior of WdgM_DeactivateAliveSupervision in error case
WDGM175	UML model linking of WdgM_GetGlobalStatus
WDGM176	Behavior of WdgM_GetGlobalStatus
WDGM177 to WDGM194	Generalized mode concept
WDGM195 to WDGM199	Behavior of mode ports