Document Title	Specification of Watchdog Manager
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	1.1.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History

Date	Version	Changed by	Change Description
31.01.2007	1.1.0	AUTOSAR Administration	• New chapter "Specification of the ports and port interfaces" added from "AUTOSAR Services" document • New feature added : active reset as optional behavior • New behavior of Deinit function : triggering of the Watchdog Driver added • Default mode for the Watchdog Manager when SetMode service fails • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
12.05.2006	1.0.0	AUTOSAR Administration	Initial release

Release Notes

Errata and known deficiencies

The current concept of how to trigger the watchdog hardware does not represent the startup, wakeup, sleep and shutdown phases properly.

Known and potential problems resulting from known deficiencies

During startup, wakeup, sleep and shutdown the Watchdog Manager (WdgM) is not scheduled to run according to the shutdown and/or wakeup concept. Therefore, the watchdog hardware would not be triggered anymore resulting in a situation where the ECU could continuously be reset.

Changes planned for next release

The specification document of the Watchdog Manager will be extended by a functionality that

- is triggered by a timer interrupt during startup, wakeup, sleep and shutdown
- supervises the sole running entity ECU State Manager during startup, wakeup, sleep and shutdown
- triggers the watchdog hardware according to the state of the supervised entity ECU State Manager
- synchronizes the trigger of the watchdog hardware during the different operation modes

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

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 from application timing constraints and 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 service different concepts of Watchdog Manager usage to proof 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.

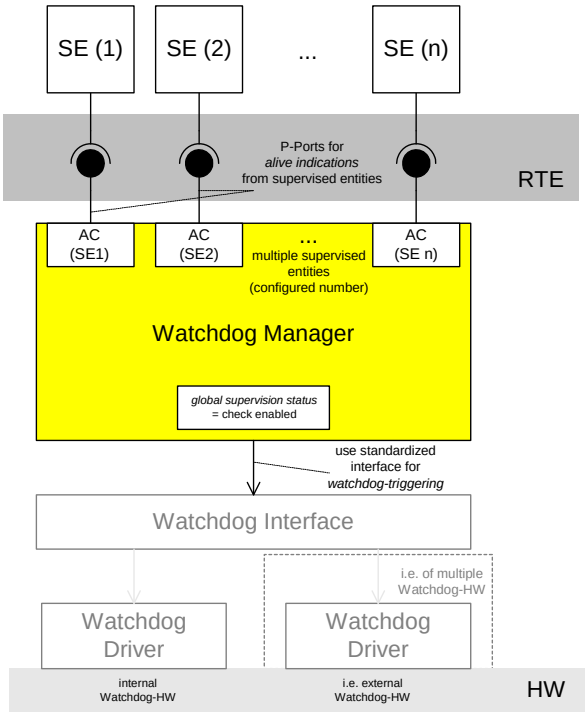


Figure 1-1: supervision of multiple entities

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.

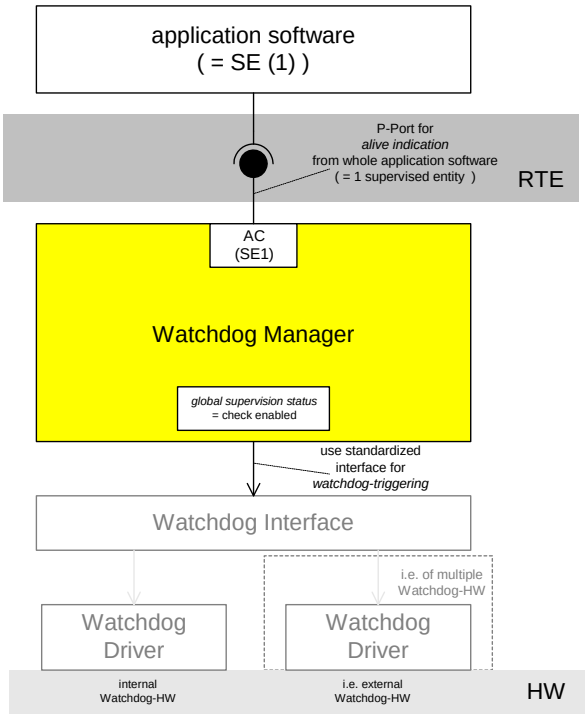


Figure 1-2 : supervision of a single entity

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.

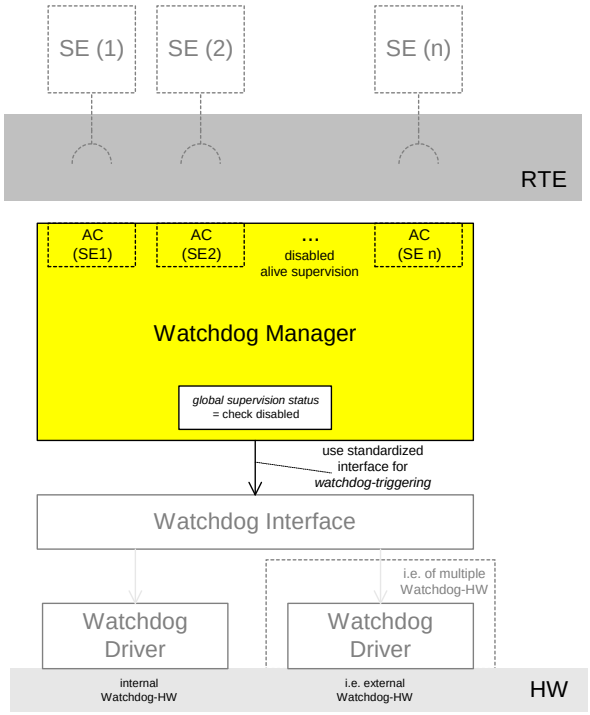


Figure 1-3 : no alive-supervision

1.1.4 Advantages in scalability of supervision concepts

The scalability of the “supervision concept” inserts flexibility to the Watchdog Manager to accept and 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 mechanism of alive-supervision for different kind 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 and abbreviations

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

terms used	*Description:*
activation status	individual status of supervised entities that indicates whether the alive-supervision of the corresponding entity is activated or deactivated.
alive counter	an independent data resource in context of a supervised entity to track and handle its amount of alive indications by the Watchdog Manager.
alive indication	an indication provided by a supervised entity to signal its aliveness to the Watchdog Manager alive-supervision.
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.
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) for 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 supervision elaborated by the internal checkup mechanism of the Watchdog Manager according to status of all supervised entities.
individual supervision status	status, that represent the current result of alive-supervision of a single supervised entity.
supervised entity	entity, which is assigned for alive-supervision by Watchdog Manager. Therefore the entity should have cyclic timing constraints on its execution, which shall be supervised.
supervision counter	an independent data resource in context of a supervised entity which is updated by the Watchdog

terms used	*Description:*
	Manager during each supervision cycle and which is used by the alive-supervision mechanism to perform the checkup with counted alive indications.
supervision cycle	the time period of Watchdog Manager, where the cyclic checkup of alive-supervision is performed.
supervision reference cycle	the amount of supervision cycles to be used as reference by the alive-supervision mechanism to perform the checkup with counted alive indications. (individual for each supervised entity)
trigger cycle	the time period of 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. (individual for each watchdog instance)

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
https://svn.autosar.org/repos/10Releases
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases
AUTOSAR_SRS_General.pdf

[3] Requirements on Mode Management
https://svn.autosar.org/repos/10Releases
AUTOSAR_SRS_ModeManagement.pdf

[4] Specification of Platform Types
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_PlatformTypes.pdf

[5] Specification of RTE Software
https://svn.autosar.org/repos/10Releases
AUTOSAR_SWS_RTE.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
  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
  The ECU State Manager is responsible for initializing, de-initializing and changing the mode of the Watchdog Manager.

- Micro Controller Unit Driver
  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
  If development error detection is enabled, the Watchdog Manager shall inform the Development Error Tracer about detected development errors.

- Diagnostic Event Manager
  The Watchdog Manager shall inform the Diagnostic Event Manager about detected functional / production-code relevant errors.

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_Lcfg.c – for link time configurable parameters and
- WdgM_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and 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:

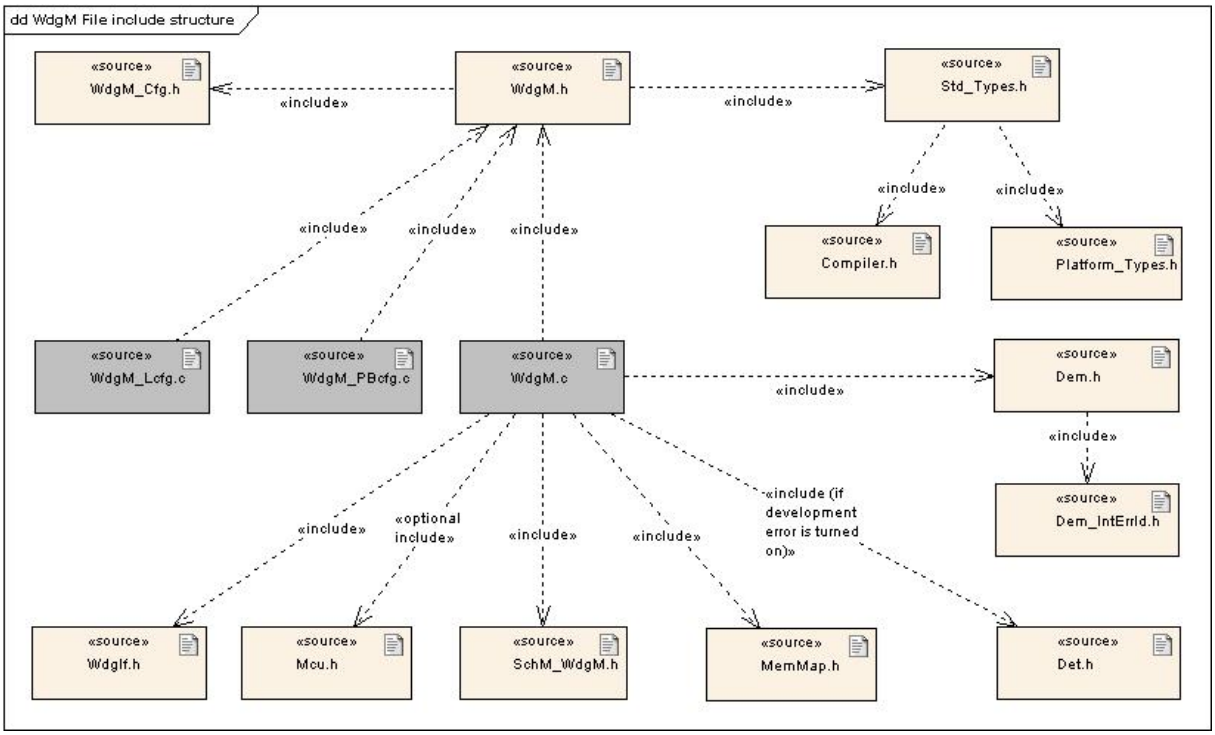


Figure 5-1: File include structure for the watchdog manager

6 Requirements traceability

Document: AUTOSAR General requirements on Basic Software Modules [3]

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], [WDGM107]
[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 values	[WDGM004]

[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], [WDGM010], [WDGM030], [WDGM020], [WDGM021], [WDGM031], [WDGM027], [WDGM028], [WDGM055], [WDGM056], [WDGM108], [WDGM057], [WDGM058], [WDGM107], [WDGM112], [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], [WDGM111], [WDGM032], [WDGM092]
[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]
[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], [WDGM112], [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	[WDGM052], [WDGM047], [WDGM111]
[BSW00411] Get version info keyword	[WDGM111]
[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], [WDGM111]
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (requirement on documentation, not on specification)

Document: AUTOSAR Requirements on Mode Management [4]

Requirement	***Satisfied by***
[BSW09028] Support multiple watchdog instances	[WDGM002], [WDGM051], [WDGM103], [WDGM109]
[BSW09106] Configuration of Watchdog Manager	[WDGM042], [WDGM046], [WDGM090], [WDGM091], [WDGM095], [WDGM118], [WDGM096], [WDGM093], [WDGM003], [WDGM085], [WDGM087], [WDGM116]
[BSW09107] Watchdog Manager initialization	[WDGM001], [WDGM018], [WDGM135], [WDGM136]
[BSW09109] Prohibit disabling of watchdog	[WDGM030], [WDGM031]
[BSW09110] Watchdog Manager mode selection service	[WDGM062], [WDGM033], [WDGM019], [WDGM139], [WDGM140]
[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	[WDGM064], [WDGM026]
[BSW09142] Enabling / Disabling alive-supervision	[WDGM059], [WDGM079], [WDGM053], [WDGM054], [WDGM099], [WDGM083], [WDGM080], [WDGM144]
[BSW09143] Behavior of the Watchdog Manager during inactive alive-supervision	[WDGM069]
[BSW09153] Access protection for disabling alive-supervision	[WDGM082], [WDGM099]
[BSW09154] Parameter for access protection for disabling alive-supervision	[WDGM082], [WDGM093]
[BSW09158] Different configuration sets for the Watchdog Manager	[WDGM106], [WDGM105], [WDGM145]
[BSW09159] Reporting failure of the alive-supervision to DEM	[WDGM022], [WDGM129]
[BSW09160] Indication of failed alive-supervision	[WDGM113], [WDGM024], [WDGM130]
[BSW09161] Condition to stop triggering the watchdog driver	[WDGM114]
[BSW09162] Indication of an upcoming watchdog reset	[WDGM143]
[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]

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 `WDGM_ALIVE_SUPERVISION` (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 (refer to chapter 8.3.5 for details of this service). 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.

WDGM069: If all supervised entities are switched off from the alive-supervision, all the *individual supervision status* values shall be set to 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 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 stat*us 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 `WDGM_IMMEDIATE_RESET` (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.

Figure 7.1-1 describes in a formal way the algorithm for calculation of *global supervision status*.

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.

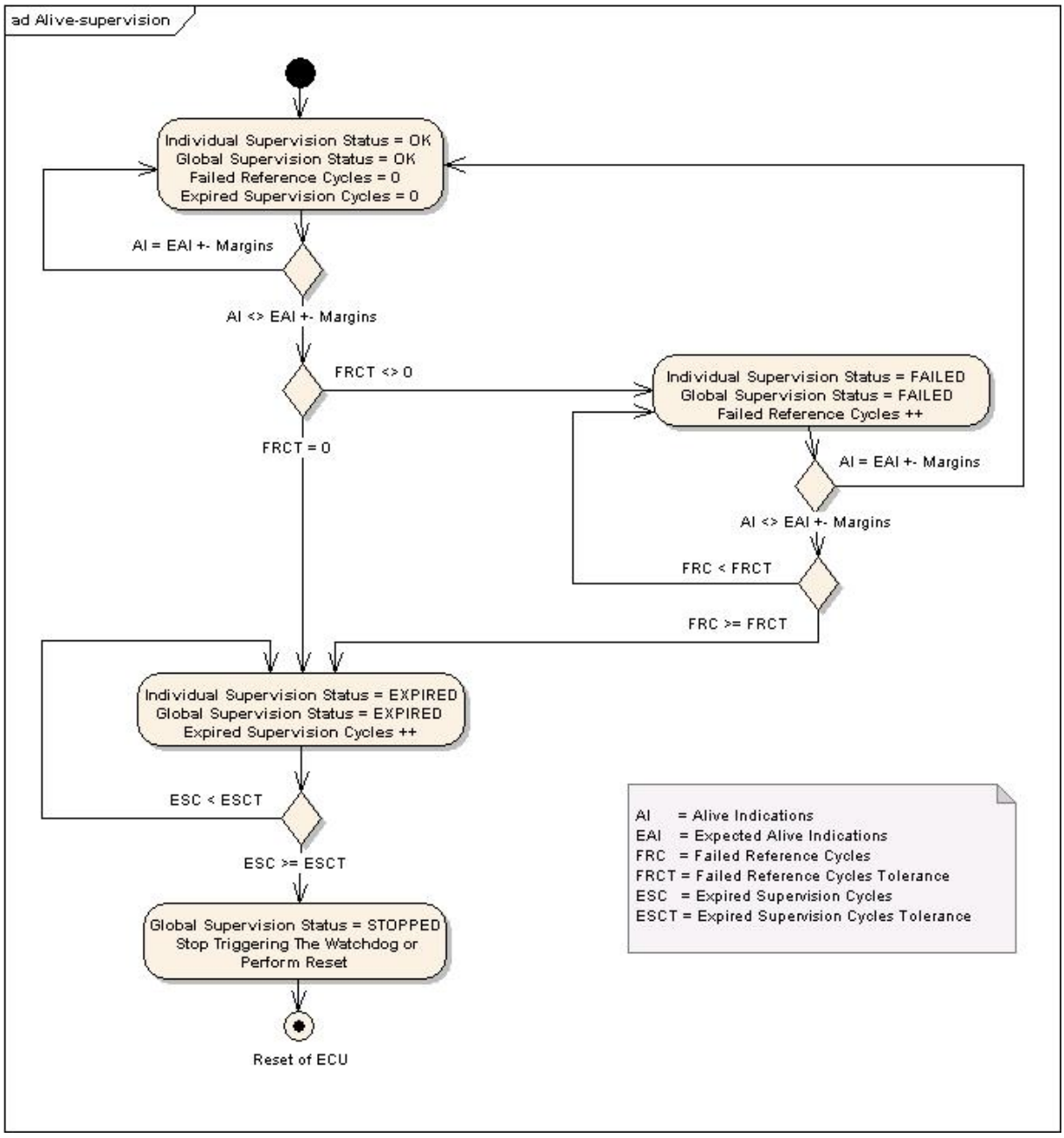


Figure 7.1-1: alive-supervision mechanism

7.1.2.6 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 WDGM_DEM_ALIVE_SUPERVISION_REPORT is set (for details see chapters 7.5 and 7.7).

7.1.2.7 Changing of alive-supervision settings

It may be possible, according to application context, that some system parameters, like schedule-table, will change during runtime. This could lead 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 the service *"WdgM_ChangeAliveSupervision"* to switch from different statically configured alive-supervision sets of parameters.

WDGM145: The service *"WdgM_ChangeAliveSupervision"* shall not be executed if the *global supervision status* is equal to WDGM_ALIVE_STOPPED i.e. the watchdog will not be triggered anymore.

7.1.3 Configurable features

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 entities by static configuration within the file `WdgM_Cfg.h`.

7.1.3.2 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 needs to be represented by the following two parameters (for details see chapter 10.2.5):

- WDGM_EXPECTED_ALIVE_INDICATIONS
- WDGM_SUPERVISION_REFERENCE_CYCLE

WDGM091: The absolute time tolerances of alive-supervision need to be transferred into countable margins as deviations regarding the expected amount of *alive indications* by the following two parameters (for details see chapter 10.2.5):

- WDGM_MIN_MARGIN
- WDGM_MAX_MARGIN

7.1.3.3 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 (for details see chapter 10.2.5):

- WDGM_FAILED_SUPERVISION_REFERENCE_CYCLE_TOLERANCE

7.1.3.4 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 (for details see chapter 10.2.4):

- WDGM_EXPIRED_SUPERVISION_CYCLE_TOLERANCE

7.1.3.5 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. (derived from statically configured supervised entities ; see details in chapter 10.2.5):
WDGM_ACTIVATION_STATUS

7.1.3.6 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. (derived from statically configured supervised entities ; see details in chapter 10.2.5) :

- WDGM_DEACTIVATION_ACCESS_ENABLED

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.

7.1.4 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 "WDGM_EXPECTED_ALIVE_INDICATIONS" (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

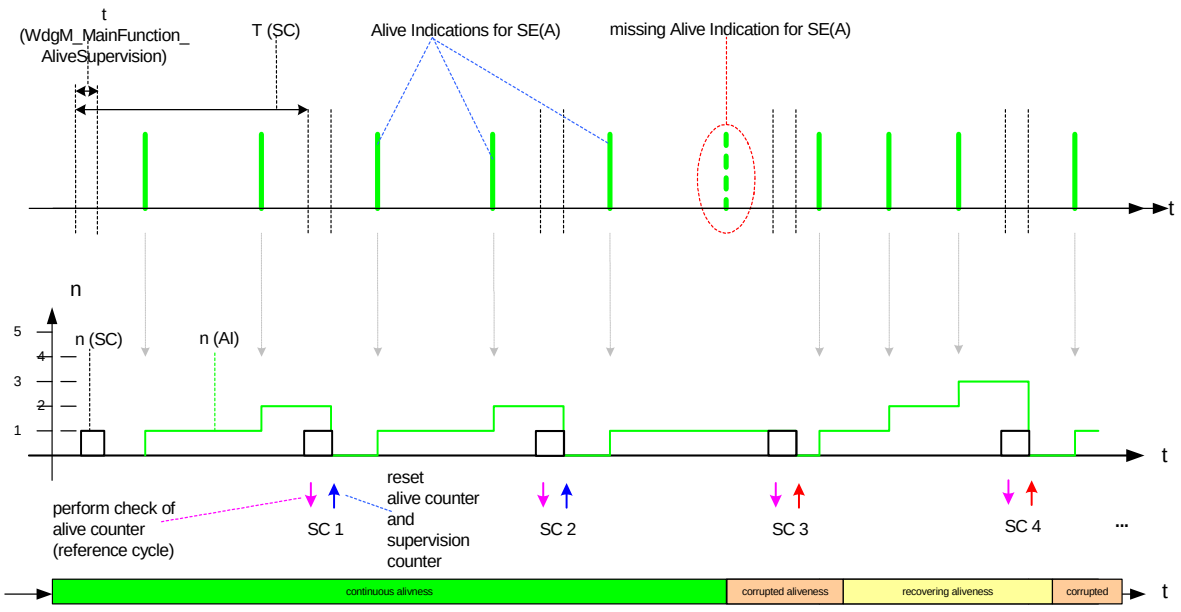
To avoid the detection of too many supervision errors for the supervised entities, we shall use the parameters “WDGM_MIN_MARGIN” and “WDGM_MAX_MARGIN” to define tolerances on the timing constraints. For some non critical supervised entities, it may be allowed to have more or less executions than expected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

(n (AI) – n (SC) + EAI <= WDGM_MAX_MARGIN) and
(n (AI) – n (SC) + EAI >= - WDGM_MIN_MARGIN)

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*.

scenario A : one or several alive indications within one supervision cycle



n () : counted number of
AI : alive indications
SC : supervision cycle

EAI : - expected n(AI) + 1 = - 1

MinMargin = MaxMargin = 0

supervision reference cycle = 1 SC

in this example : expected n(AI) = 2 ;
2 expected alive indications within one supervision cycle ;

CYCLE 1 : $n(AI)_{cycle\ 1} - n(SC) + EAI = 2 - 1 - 1 = 0$ → o.k.

CYCLE 2 : $n(AI)_{cycle\ 2} - n(SC) + EAI = 2 - 1 - 1 = 0$ → o.k.

CYCLE 3 : $n(AI)_{cycle\ 3} - n(SC) + EAI = 1 - 1 - 1 = -1$ → not o.k.

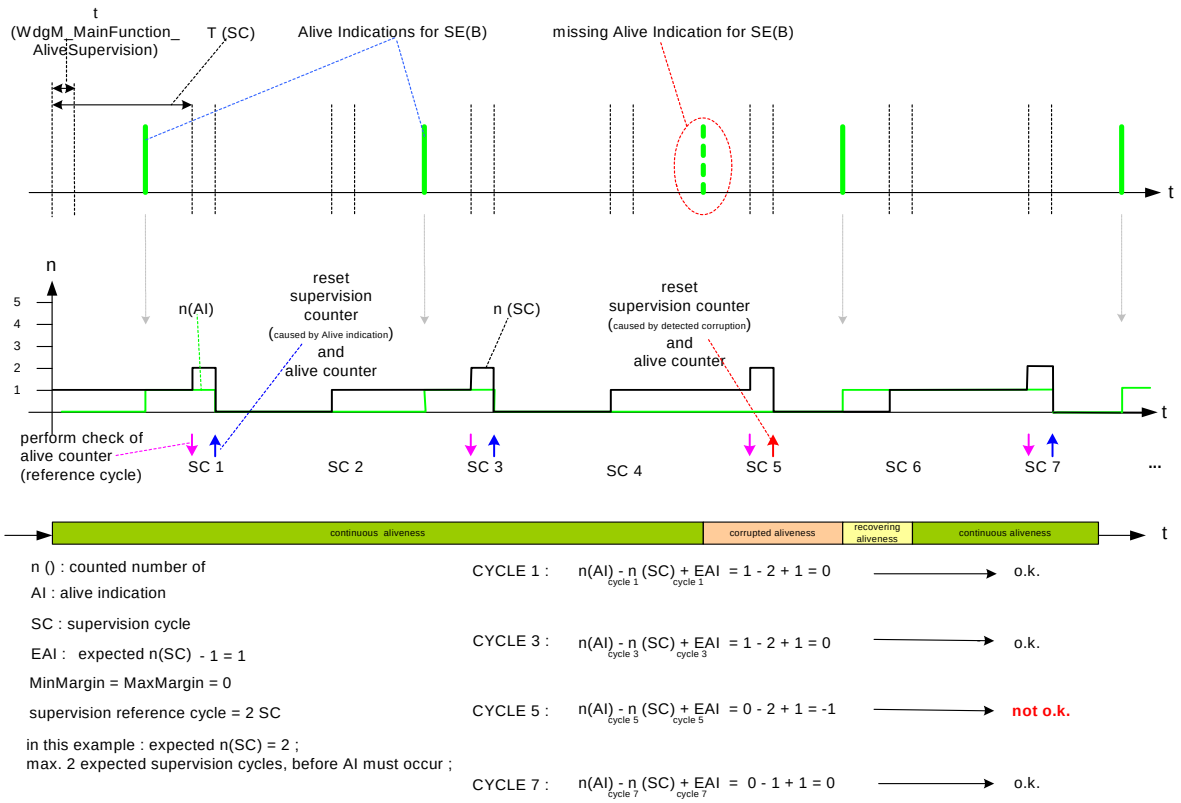
CYCLE 4 : $n(AI)_{cycle\ 4} - n(SC) + EAI = 3 - 1 - 1 = 1$ → not o.k.

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 “WDGM_FAILED_SUPERVISION_REFERENCE_CYCLE_TOLERANCE” different from 0, because this brings back alive-supervision in suitable condition for the next *supervision cycle*.

scenario B : alive indication period longer than one supervision cycle



n () : counted number of
AI : alive indication
SC : supervision cycle
EAI : expected n(SC) - 1 = 1
MinMargin = MaxMargin = 0
supervision reference cycle = 2 SC
in this example : expected n(SC) = 2 ;
max. 2 expected supervision cycles, before AI must occur ;

CYCLE 1 : $n(AI)_{cycle\ 1} - n(SC)_{cycle\ 1} + EAI = 1 - 2 + 1 = 0$ → o.k.

CYCLE 3 : $n(AI)_{cycle\ 3} - n(SC)_{cycle\ 3} + EAI = 1 - 2 + 1 = 0$ → o.k.

CYCLE 5 : $n(AI)_{cycle\ 5} - n(SC)_{cycle\ 5} + EAI = 0 - 2 + 1 = -1$ → **not o.k.**

CYCLE 7 : $n(AI)_{cycle\ 7} - n(SC)_{cycle\ 7} + EAI = 0 - 1 + 1 = 0$ → o.k.

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

Notes:

- 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.

- 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).

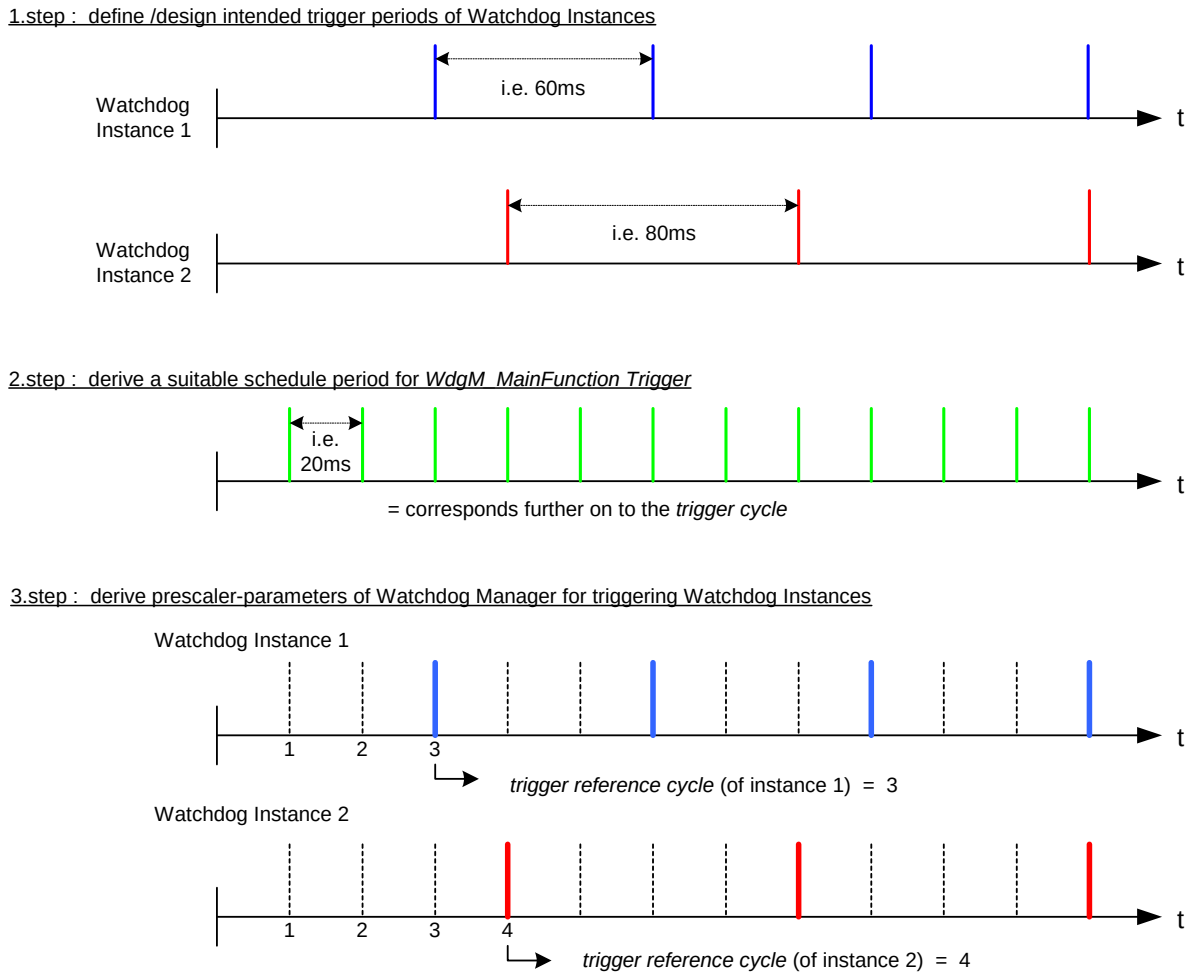


Figure 7.2-2 trigger schedule for 2 watchdog instances

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 features

WDGM003: The "Device Index" which is necessary to address different watchdog drivers shall be statically configured and represented by the following parameter (for details see chapter 10.2.7):

- WDGM_WATCHDOG_INSTANCE_ID

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 parameters (for details see chapter 10.2.7)

- WDGM_TRIGGER_SLOW_REFERENCE_CYCLE
- WDGM_TRIGGER_FAST_REFERENCE_CYCLE

7.3 Modes of the Watchdog Manager

7.3.1 Mode setting of watchdog

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

The Watchdog Manager change mode service is defined to support some "intelligent" hardware watchdogs which can be reconfigured dynamically (reconfiguration means changing the timeout period of the hardware watchdog). Therefore, the Watchdog Interface, which is used to address the Watchdog Driver, has defined the following modes:

- WDGIF_OFF_MODE : the watchdog driver is disabled (switched off)
- WDGIF_SLOW_MODE : the watchdog driver is set up for a long timeout period (slow triggering)
- WDGIF_FAST_MODE : the watchdog driver is set up for a short timeout period (fast triggering)

These modes shall also be used when calling the *"WdgM_SetMode"* service.

When the change mode service of the Watchdog Manager is called, it calls the *"WdgIf_SetMode"* of the Watchdog Interface with a "Device Index" and a "Mode" as parameters. The *"WdgIf_SetMode"* service will then call the *"Wdg_SetMode"* service of the corresponding watchdog driver with the expected "Mode" as parameter.

As a consequence, the schedule period of the hardware watchdog triggering shall be changed according to the new timing constraints. This is realized by using the appropriate *trigger reference cycle* which defines the prescaler value for the corresponding mode and which is represented by the following parameters (for details see chapter 10.2.7).

- WDGM_TRIGGER_SLOW_REFERENCE_CYCLE
- WDGM_TRIGGER_FAST_REFERENCE_CYCLE

WDGM139: If the *"WdgIf_SetMode"* service fails, which means that the switch mode of the watchdog driver has not been executed correctly, the Watchdog Manager shall assume that the watchdog driver is in WDGIF_FAST_MODE mode in order to trigger the watchdog quickly enough to avoid unintended reset.

WDGM140: In case of multiple watchdog instances, if one of the *"WdgIf_SetMode"* service fails, the Watchdog Manager shall assume that all the watchdog drivers are in WDGIF_FAST_MODE mode.

WDGM033: If the hardware watchdog supports to be switched off and the environment allows the usage of this feature, if parameter WDGM_OFF_MODE_ENABLED enables this feature (e.g. in non safety related systems), it shall be achieved by calling the *"WdgM_SetMode"* routine with the corresponding mode parameter WDGIF_OFF_MODE..

7.3.2 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 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 which will be typed by the same interfaces and which has to be connected to the ports of the Watchdog Manager, so that the RTE, the appropriate IDs and the required symbols can be generated.
The service port of the Watchdog Manager which is used to request a mode switch within RTE will only be defined in the SW-C description of the Watchdog Manager Service.
The following interface definitions are interpreted to be in:
`ARPackage AUTOSAR/Services/WdgM`

7.4.1 Ports and Port Interface for Interfaces between Watchdog Manager and Software Components

7.4.1.1 General Approach

To reduce the number of ports provided by a watchdog service all interfaces between SW-C and service are modeled as Client/Server communication. To model updating of an alive counter the sender-receiver paradigm seems to be more obvious. 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 Entity IDs.

The Supervised Entity IDs are used to address or to identify the supervised entities with an ECU wide unique ID. In order to keep the application code independent from the configuration of ECU dependent supervised entity IDs, these IDs are not modeled explicitly as data elements to be passed between application and service. These IDs are modeled as “port defined argument values”[1] 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).

7.4.1.2 Data Types

For the port interface of the watchdog service no type is 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 `uint16` (specified as Uint16). Therefore the following type description is required:
`Uint16 SupervisedEntityIdType`

Due to the reason that 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[2].

[1] New feature which will be added to the SWC Template and the RTE
[2] If the type is changed in a dedicated ECU this has a strong implication on relocatability of object code delivered applications! The applications might expect another type than implemented within a dedicated ECU.

Therefore the ECU specific type definition has to be provided by the Watchdog Manager instead to provide an “AUTOSAR type”.

7.4.1.3 Port Interface

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 (notation in pseudo code).

```
ClientServerInterface WdgMService {
     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.

7.4.1.4 Ports

Figure 3 shows how AUTOSAR Software components (single or multiple instances) are connected via service ports to the Watchdog Manager.

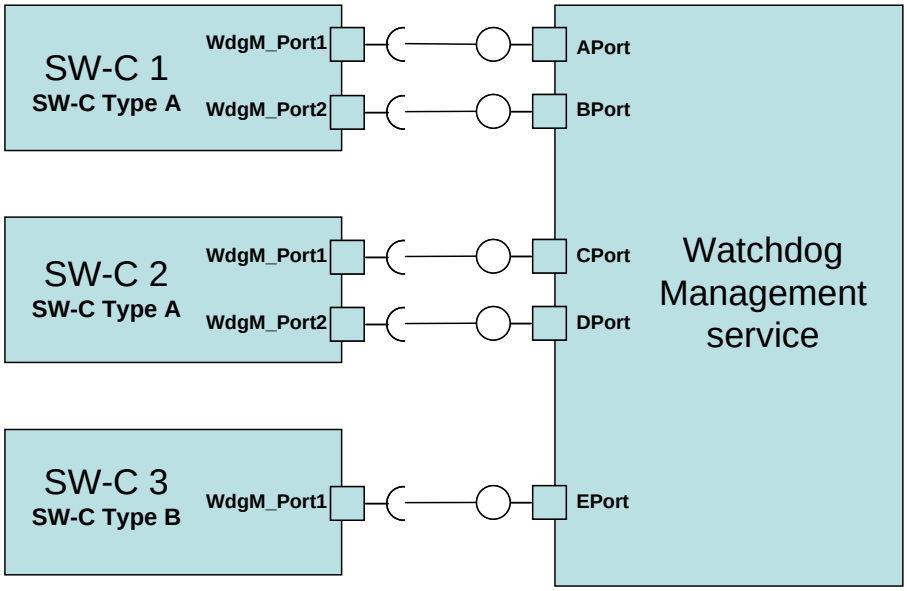


Figure 3: Example of SW-Cs connected to the Watchdog Manager via service ports. On the left side, there are two instances of component SW-C TypeA and one instance of component SW-C TypeB.

On the Watchdog Manager side, there is one port per supervised entity providing all the services of the interface `WdgMService` described above. Each supervised entitiy has one port for requiring those services for each supervised entity associated with that application.
There is no naming convention for the ports providing the services. In this example they are named:
`APort, BPort, … , EPort`
The names shall be equal to the names declared within the typed enumeration defined within this specification: *WdgM_SupervisedEntityIdType*.

7.4.1.5 Error Codes

The watchdog manager service does not return any service specific error codes.

7.4.2 Ports and Port Interface for Interfaces between Watchdog Manager and RTE

7.4.2.1 General Approach

It is very important to point out, that the mode management concept of AUTOSAR needs refinement. It is very likely that interfaces related to mode management will in future be changed. Therefore the interface described below is very likely to be changed/deleted in future!

WDGM143: The watchdog manager accesses the mode port of the RTE to change modes due to erroneous alive counter updates. According to RTE specification [5] this interface shall be modeled as `SenderReceiverInterface`.

7.4.2.2 Port Interface and Data Types

The `SenderReceiverInterface` of mode ports is defined as follows:
It shall

- be a `SenderReceiverInterface`,
- contain exactly one `DataElementPrototype` named `ModeIndex`.
- The `ModeIndex` shall have `isQueued = true` and
- be of type `ModeIndexType`.
- The `ModeIndexType` shall be a globally defined `PrimitiveTypeWithSemantics`.
- The type of `ModeIndexType` shall be `UInt8`.

Thus the pseudo code description used within this document of the interface looks as follows:

```
SenderReceiverInterface RteModeInterface {
     ModeIndexType ModeIndex {isQueued = true};
```

```
};
```

7.4.2.3 Mode Port

The port definition is very simple. Each Watchdog Manager service provides exactly one port of the interface `ModeInterface`. For harmonization reasons this port should be called: `WdgMModePort`. This port is connected to the mode port.
In the VFB View the `WdgMModePort` is not shown. Only application ports are visualized.
After mapping of SW-C and services to ECUs this Port will be connected to the appropriate mode port.

7.4.3 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.

7.4.4 Summary of Ports

Taking all decisions together we end up with the following structure for the AUTOSAR interface of the Watchdog Manager:

- For each supervised entity:
  - One port with a client-server interface providing all SW-C accessible services is provided by the Watchdog Manager
  - One port with a client-server interface requiring all SW-C accessible services is provided by each SW-C having supervised entities.
- For the watchdog manager service
  - One port with a sender-receiver interface providing the watchdog manager the ability to request mode switches.

The Provide Ports for single-supervised entity operations have a certain relation to the `InternalBehavior` of the Watchdog Manager. With each call, the supervised entity identifier will be passed as an additional argument by the RTE to the service as first parameter of the C-API. The type of this identifier is ECU specific and must be defined for each ECU like:

```
Uint16 SupervisedEntityIdType {
      LOWER-LIMIT = 0;
      UPPER-LIMIT = <xx>;
};
```

This type does not show up in the service ports of the client components, because the supervised entity identifier is implemented as port defined argument value, which

are part of the InternalBehavior of the Watchdog Manager Service. So the ECU dependency of *WdgM_SupervisedEntityIdType* is not visible for the clients.
The InternalBehavior of the Watchdog Manager Service is only seen by the local RTE. The description of the port defined argument values looks as follows

```
InternalBehavior
{
     PortArgument{port= APort, value.type=
     SupervisedEntityIdType,
     value.value=1}
}
```

7.4.5 Internal Behavior

In addition to the port defined argument values 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
```

7.4.6 Definition of the Service

The upper definitions lead to the following description of the watchdog manager service:

```
Service WdgM
{
     ProvidePort WdgMService APort;
     ProvidePort WdgMService BPort;
     ProvidePort WdgMService CPort;
     …
     ProvidePort RteModeInterface WdgMModePort;

     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= APort, value.type=
            SupervisedEntityIdType,
            value.value=1};

            PortArgument{port= BPort, value.type=
            SupervisedEntityIdType,
            value.value=2};

            PortArgument{port= CPort, value.type=
            SupervisedEntityIdType,
            value.value=3};
            …
        };
};
```

It is obvious, that we must generate the descriptive parts, which depend on the number of components and supervised entities, per ECU, while the description of the port interfaces can be standardized beforehand.
In addition, but not visible on the VFB, the BSW internal C-APIs are used to access the services which are only allowed to be accessed by other BSW modules (e.g. services for initialization).

7.5 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 wrong “alive-supervision configuration identifier”	Development	WDGM_E_PARAM_CONFIGID	0x14

parameter			
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.

7.6 Error detection

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

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

7.7 Error notification

WDGM048: Development errors shall be reported to the error hook of the Development Error Tracer (DET) if the pre-processor switch WDGM_DEV_ERROR_DETECT is set. They shall not be used as the return value of the called function.

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

8.1.1 Standard types

WDGM011: The standard API return type “Std_ReturnType” defined in Std_Types.h shall be used to indicate success / failure of an API call.

- Std_VersionInfoType

8.1.2 WdgIf types

- 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

Type:	WdgM_ConfigType	
Range:	Structure	A structure to hold the Watchdog Manager configuration set.
Description:	A pointer to such a structure is provided to the Watchdog Manager initialization routine for configuration. **WDGM042:** Configuration file shall contain: • Number of watchdog instances to trigger • Watchdog instances identifiers • Number of entities to supervise • Supervised entities identifiers • Flag to activate / deactivate the supervision of an entity under control of the Watchdog Manager • Timing constraints of each entity under control of the Watchdog Manager (min and max margins values, number of expected alive indications, supervision reference cycle and number of failed supervision reference cycles for the checking algorithm) • Number of supervision cycles allowed before stop triggering the HW watchdog in case of alive-supervision failure • Timing constraints of each watchdog instance to trigger	

8.2.2 WdgM_SupervisedEntityIdType

Type:	Uint16	
Range:	0-65535	Identifiers of the entities which are under control of the Watchdog Manager
Description:	Supervised entity Identifiers are provided to the Watchdog Manager initialization routine for configuration.	

8.2.3 WdgM_WatchdogInstanceIdType

Type:	Uint8	
Range:	0-255	Identifiers of the watchdog driver instances which are to be triggered by the Watchdog Manager
Description:	Watchdog driver Identifiers are provided to the Watchdog Manager initialization routine for configuration.	

8.2.4 WdgM_AliveSupervisionConfigIdType

Type:	Uint8	
Range:	0-255	Identifiers of the alive-supervision configuration sets
Description:	Alive-supervision configuration identifier is provided to the WdgM_ChangeAliveSupervision service	

8.2.5 WdgM_AliveSupervisionStatusType

Type:	Uint8		
Ranges:	`WDGM_ALIVE_OK`	0	timing constraints are fulfilled within its acceptable margins.
	`WDGM_ALIVE_FAILED`	1	timing constraints have failed including the margins, but the amount of *failed supervision reference cycles* has not been exceeded.
	`WDGM_ALIVE_EXPIRED`	2	timing constraints have failed including the margins for more often than the acceptable amount of *failed supervision reference cycles.*
	`WDGM_ALIVE_STOPPED`	3	timing constraints have been expired and the time-limit to postpone the blocking of watchdog triggering is exceeded.
	`WDGM_ALIVE_DEACTIVATED`	4	alive-supervision is disabled for this supervised entity
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.		

8.2.6 WdgM_ActivationStatusType

Type:	enum	
Ranges:	`WDGM_SUPERVISION_ENABLED`	alive-supervision shall be performed
	`WDGM_SUPERVISION_DISABLED`	alive-supervision shall not be performed; the individual supervision status of the corresponding entity shall be set to WDGM_ALIVE_DEACTIVATED.
Description:	This Type shall be used for any variables that contain *activation status* data of supervised entities. Therefore this Type shall also be used for parameters that define e.g. initial values of *activation status*.	

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

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 configuration set
Parameters (out):	--
Return value:	--
Description:	**WDGM001:** This routine initializes the Watchdog Manager, i.e. it sets the default Watchdog Manager mode as provided in the configuration set. After execution of this routine, alive-supervision is activated according to the list of supervised entities defined in the configuration set. **WDGM018:** This routine is called by the ECU State Manager in Startup phase and shall initialize all module global variables. **WDGM135:** This routine shall call the SetMode routine of the watchdog interface to switch the watchdog driver into the default mode of the Watchdog Manager. **WDGM136:** If the SetMode routine of the watchdog interface returns `E_NOT_OK`, the Watchdog Manager shall assume that the watchdog driver is in FAST mode. **WDGM137:** The error shall be reported to the Diagnostic Event Manager with the value `WDGM_E_SET_MODE`. **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 WDGM_OFF_MODE_ENABLED (e.g. in safety relevant systems), the routine shall check if the default 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`.
Caveats:	--
Configuration:	--

8.3.2 WdgM_DeInit

Service name:	WdgM_DeInit
Syntax:	void WdgM_DeInit (void)
Service ID [hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	--
Parameters (out):	--
Return value:	--
Description:	**WDGM084:** This routine deinitializes the Watchdog Manager, i.e. after execution of this routine, the alive-supervision and the triggering of the watchdog driver are deactivated. **WDGM045:** This routine shall trigger the watchdog driver to avoid watchdog reset during shutdown phase. This routine is called by the ECU State Manager in shutdown phase. If watchdog triggering is needed after execution of this service, this shall be handled by the ECU State Manager. **WDGM088:** If development error detection is enabled, calling any other Watchdog Manager service except WdgM_Init after WdgM_DeInit being called will cause a development error which shall be reported to the Development Error Tracer with the value `WDGM_E_NO_INIT`.
Caveats:	--
Configuration:	--

8.3.3 WdgM_GetVersionInfo

Service name:	WdgM_GetVersionInfo	
Syntax:	void WdgM_GetVersionInfo (Std_VersionInfoType *VersionInfo)	
Service ID [hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Non reentrant	
Parameters (in):	--	
Parameters (out):	`VersionInfo`	Pointer to where to store the version information of the module WdgM.
Return value:	--	
Description:	**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: `WDGM_VERSION_INFO_API`
Caveats:	--
Configuration:	--

8.3.4 WdgM_SetMode

Service name:	WdgM_SetMode
Syntax:	`Std_ReturnType WdgM_SetMode (WdgIf_ModeType Mode)`
Service ID [hex]:	0x03
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	`Mode` One of the following statically configured modes: • `WDGIF_OFF_MODE` • `WDGIF_SLOW_MODE` • `WDGIF_FAST_MODE`
Parameters (out):	--
Return value:	`Std_ReturnType`
Description:	**WDGM016:** The Watchdog Manager can be switched between different modes that are statically configured and contained in the Watchdog Manager configuration set provided to the initialization routine. **WDGM019:** This routine is called by the ECU State Manager to change the Watchdog Manager mode. The routine shall call the SetMode routine of the watchdog interface to switch the watchdog driver into the corresponding mode. For instance : • During Start-up mode : Slow mode (long timeout period) for the watchdog • During Run mode : Fast mode (short timeout period) for the watchdog If this mode switch doesn't succeed (SetMode routine of the watchdog driver returns `E_NOT_OK`), the routine shall return the value `E_NOT_OK` and consider that the watchdog driver is in Fast mode. **WDGM142:** The error shall be reported to the Diagnostic Event Manager 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 WDGM_OFF_MODE_ENABLED (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`.
Caveats:	--
Configuration:	--

8.3.5 WdgM_UpdateAliveCounter

Service name:	WdgM_UpdateAliveCounter
Syntax:	`Std_ReturnType WdgM_UpdateAliveCounter (WdgM_SupervisedEntityIdType SEId)`
Service ID [hex]:	0x04
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	`SEId` Identifier of the entity under control of the WdgM whose alive counter shall be updated
Parameters (out):	--
Return value:	`Std_ReturnType`
Description:	**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`.
Caveats:	--
Configuration:	--

8.3.6 WdgM_ActivateAliveSupervision

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 (out):	--
Return value:	`Std_ReturnType`
Description:	**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`.
Caveats:	--
Configuration:	--

8.3.7 WdgM_DeactivateAliveSupervision

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 (out):	--
Return value:	`Std_ReturnType`
Description:	**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 If development error detection is enabled, 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`.
Caveats:	--
Configuration:	--

8.3.8 WdgM_ChangeAliveSupervision

Service name:	WdgM_ChangeAliveSupervision
Syntax:	`Std_ReturnType WdgM_ChangeAliveSupervision`

	`(WdgM_AliveSupervisionConfigIdType ConfigId)`
Service ID [hex]:	0x07
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	`ConfigId` Identifier of a WdgM_AliveSupervision configuration set
Parameters (out):	--
Return value:	`Std_ReturnType`
Description:	**WDGM106:** This routine shall be used to change the configuration set of the alive-supervision for the Watchdog Manager. **WDGM107:** If development error detection is enabled, the parameter <ConfigId> shall be checked for being within the allowed boundaries. If an error is detected the service shall not be executed and the error shall be reported to the Development Error Tracer with the value `WDGM_E_PARAM_CONFIGID` and the routine shall return the value `E_NOT_OK`. **WDGM112:** 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`.
Caveats:	**--**
Configuration:	**--**

8.4 Call-back notifications

Not Applicable

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

8.5.1 WdgM_MainFunction_AliveSupervision

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

Service name:	WdgM_MainFunction_AliveSupervision
Service ID [hex]:	0x08
Description:	**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 WDGM_SUPERVISION_CYCLE. **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`.
Timing:	Fixed cyclic
Pre condition:	--
Configuration:	--

8.5.2 WdgM_MainFunction_Trigger

Service name:	WdgM_MainFunction_Trigger
Service ID [hex]:	0x09
Description:	**WDGM067:** This service shall be scheduled according to the timing constraints defined with parameter WDGM_TRIGGER_CYCLE. **WDGM040:** This service shall include an internal prescaling mechanism to derive trigger periods for each assigned watchdog instance in each mode, according to parameters WDGM_TRIGGER_SLOW_REFERENCE_CYCLE and WDGM_TRIGGER_FAST_REFERENCE_CYCLE 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`.
Timing:	Variable cyclic
Pre condition:	--

Configuration:	--

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

WDGM008: This chapter defines all interfaces which are required to fulfill the core functionality of the module.

API function	*Module*	*Description*
WdgIf_SetMode	WdgIf	To access to the SetMode service of the watchdog driver
WdgIf_Trigger	WdgIf	To access to the Trigger service of the watchdog driver
Dem_ReportErrorStatus	Dem	Production error notification

8.6.2 Optional Interfaces

WDGM009: This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
Det_ReportError	Det	Development error notification	WDGM_DEV_ERROR_DETECT
Mcu_PerformReset	Mcu	To perform a direct microcontroller reset	WDGM_IMMEDIATE_RESET

8.6.3 Configurable interfaces

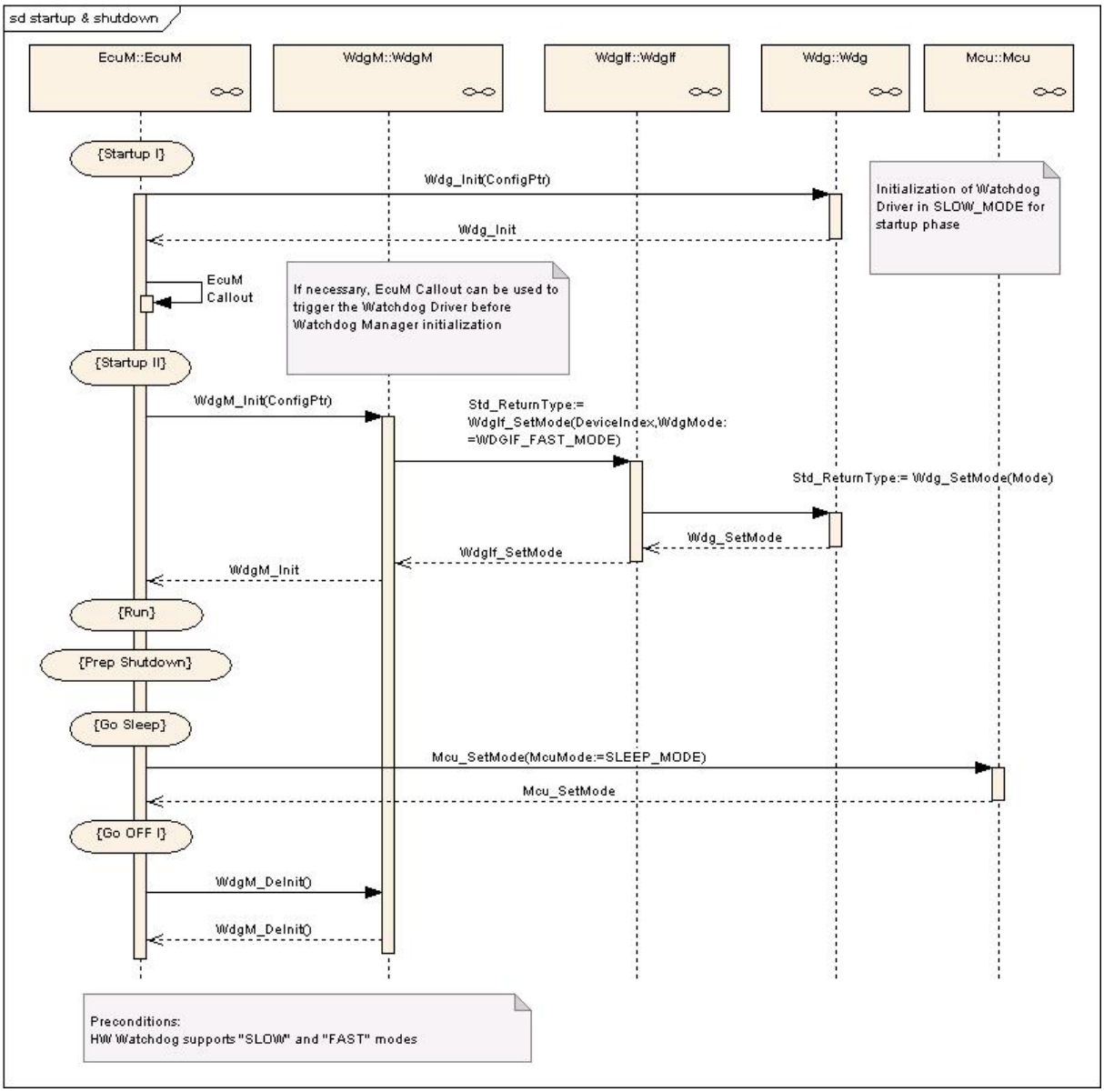
Not Applicable

8.6.4 Job End Notification

Not Applicable

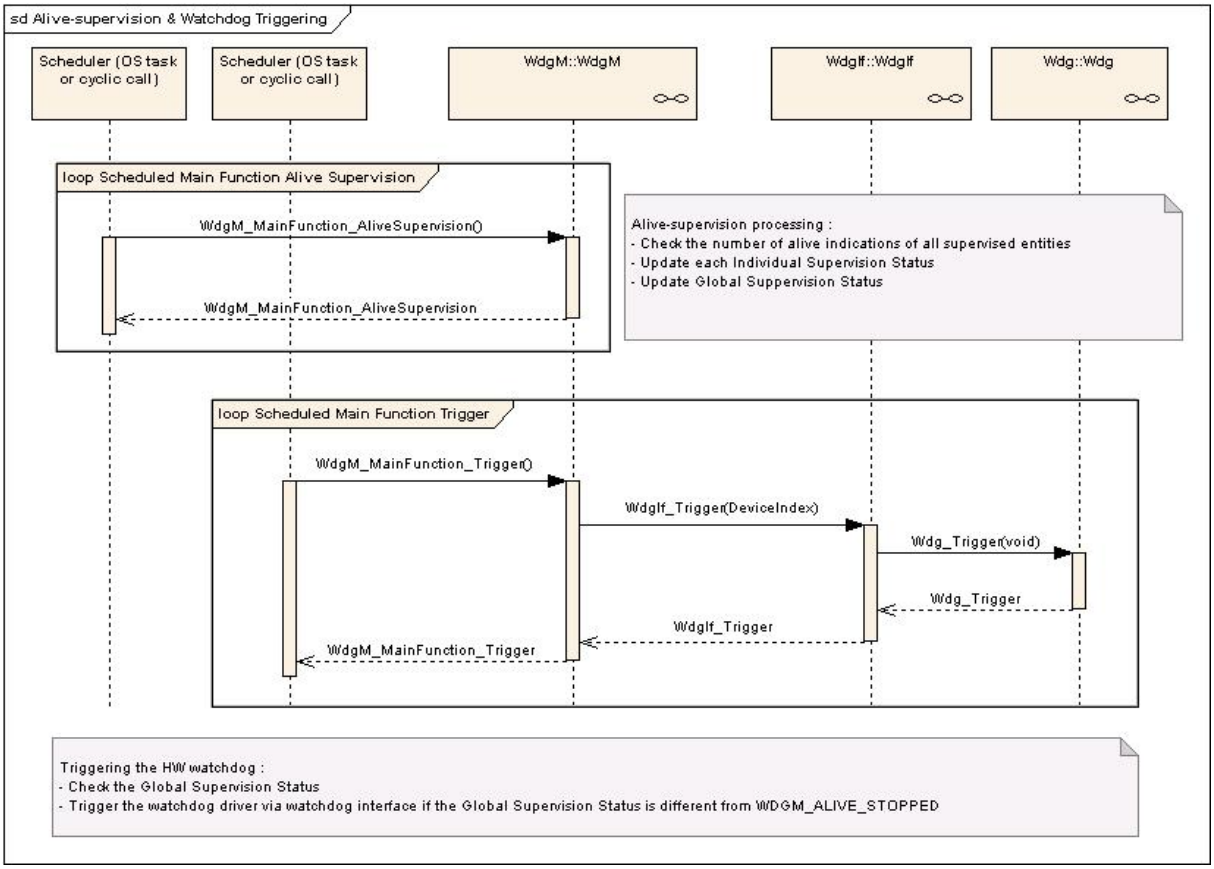
9 Sequence diagrams

9.1 Startup and Shutdown



Note: As mentioned in chapter 4.1, the Watchdog Manager doesn't encapsulate the watchdog driver initialization which needs to be performed early in the startup phase (before operating system initialization).

9.2 Alive supervision and watchdog triggering



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

10.2.2 Overview

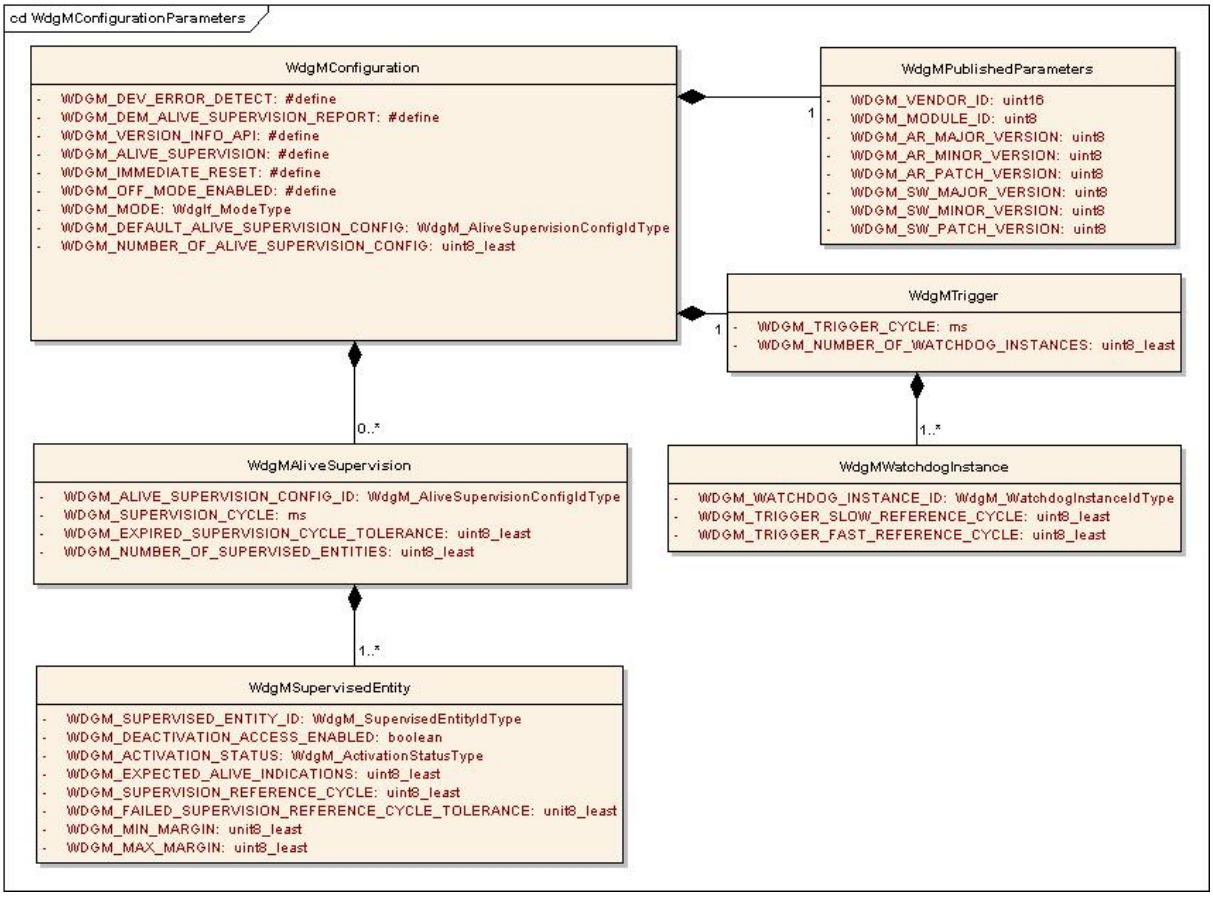


Figure 100-1: Watchdog Manager configuration parameters

10.2.3 Watchdog Manager Configuration Parameters

SWS Item	**WDGM032:**
Container Name	WdgMConfiguration
Description	Container for entire configuration of the Watchdog Manager.
Configuration Parameters	

Name	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		
Type or Unit	#define		
Range	ON	Development error detection is enabled	
	OFF	Development error detection is disabled	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	
	Post Build	--	
Scope	module		
Dependency	WDGM047, WDGM048		

Name	WDGM_DEM_ALIVE_SUPERVISION_REPORT		
Description	Preprocessor switch to enable/disable alive-supervision error reporting to DEM. Shall be used to remove unneeded code segments		
Type or Unit	#define		
Range	ON	Alive-supervision error reporting is enabled	
	OFF	Alive-supervision error reporting is disabled	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	
	Post Build	--	
Scope	module		
Dependency	WDGM129		

Name	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		
Type or Unit	#define		
Range	ON	API is enabled	
	OFF	API is disabled	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	
	Post Build	--	
Scope	module		
Dependency	WDGM111		

Name	WDGM_ALIVE_SUPERVISION	
Description	Preprocessor switch to enable/disable the alive-supervision mechanism. Shall be used to remove unneeded code segments regarding alive-supervision. Use Case : no supervised entity assigned (see chapter 1.1.3)	
Type or Unit	#define	
Range	ON	Alive-supervision mechanism is enabled
	OFF	Alive-supervision mechanism is disabled

Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	WDGM052		

Name	WDGM_IMMEDIATE_RESET		
Description	Preprocessor switch to enable/disable the immediate reset feature in case of alive-supervision failure. Shall be used to remove unneeded code segments regarding alive-supervision.		
Type or Unit	#define		
Range	ON	Immediate reset is enabled	
	OFF	Immediate reset is disabled	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	WDGM132		

Name	WDGM_OFF_MODE_ENABLED		
Description	Parameter to enable/disable the access of Off-Mode selection (Off mode of the watchdog driver)		
Type or Unit	#define		
Range	ON	Watchdog-HW could be switched off	
	OFF	Watchdog-HW could not be switched off	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	
	Post Build	--	
Scope	ECU		
Dependency	The parameter setting should be derived from the capabilities of watchdog-HW and the related Wdg-Driver modules.		

Name	WDGM_MODE		
Description	The parameter shall contain the initial mode of the Watchdog Manager.		
Type or Unit	WdgIf_ModeType		
Range	--		
	--		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	module		
Dependency	The initial mode must cooperate with the initial mode of the watchdog interface and watchdog driver(s).		

Name	WDGM_DEFAULT_ALIVE_SUPERVISION_CONFIG		
Description	The parameter shall contain the identifier of the alive-supervision configuration set to be used for the initialization of the Watchdog Manager.		
Type or Unit	WdgM_AliveSupervisionConfigIdType		
Range	--		
	--		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		

Dependency	

Name	WDGM_NUMBER_OF_ALIVE_SUPERVISION_CONFIG		
Description	This parameter shall contain the amount of alive-supervision configuration sets.		
Type or Unit	uint8_least		
Range	min : 0		
	max : --		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	--		

Container Name	***Multiplicity***	***Scope / Dependency***
WdgMAliveSupervision	0..*	Container for a management structure to configure the alive-supervision of the Watchdog Manager. Its multiplicity describes the number of alive-supervision configuration sets.
WdgMTrigger	1	Container for a management structure to configure the trigger service of the Watchdog Manager.

10.2.4 Watchdog Manager Alive Supervision Parameters

SWS Item	**WDGM035:**
Container Name	WdgMAliveSupervision
Description	This container collects all configuration parameters of Alive-Supervision
Configuration Parameters	

Name	WDGM_ALIVE_SUPERVISION_CONFIG_ID		
Description	This parameter shall contain the identifier of the alive-supervision configuration set		
Type or Unit	WdgM_AliveSupervisionConfigIdType		
Range	--		
	--		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	--		

Name	WDGM_SUPERVISION_CYCLE		
Description	This parameter defines the schedule period of the main function WdgM_MainFunction_AliveSupervision.		
Type or Unit	ms		
Range	Application specific		
	--		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	ECU (integration)		

Dependency	--

Name	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.		
Type or Unit	uint8_least		
Range	min : 0		
	max : --		
Configuration Class	***Pre-compile***	--	--
	Link time	--	--
	Post Build	X	Variant 1
Scope	ECU		
Dependency	WDGM118		

Name	WDGM_NUMBER_OF_SUPERVISED_ENTITIES		
Description	This parameter shall contain the amount of supervised entities for this alive-supervision configuration set.		
Type or Unit	uint8_least		
Range	min : 1		
	max : --		
Configuration Class	***Pre-compile***	X	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMSupervisedEntity	1..*	One set of parameter of this container is required per Supervised Entity to prepare the individual settings of each Supervised Entity.

10.2.5 Watchdog Manager Supervised Entity Parameters

SWS Item	**WDGM037:**
Container Name	WdgMSupervisedEntity
Description	This container collects all configuration parameters to define the individual settings of a Supervised Entity
Configuration Parameters	

Name	WDGM_SUPERVISED_ENTITY_ID	
Description	This parameter shall contain the identifier of the supervised entity	
Type or Unit	WdgM_SupervisedEntityIdType	
Range	--	
	--	

Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	--		

Name	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.		
Type or Unit	bool		
Range	TRUE	access is enabled	
	FALSE	access is denied	
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM079, WDGM082, WDGM093		

Name	WDGM_ACTIVATION_STATUS		
Description	This parameter shall contain the activation status of the corresponding SE		
Type or Unit	WdgM_ActivationStatusType		
Range	WDGM_SUPERVISION_ ENABLED	Alive Supervision of the corresponding SE is armed/activated right from the start of Alive Supervision	
	WDGM_SUPERVISION_ DISABLED	Alive Supervision of the corresponding SE needs to be armed/activated by the service *WdgM_ActivateAliveSupervision*	
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM096		

Name	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.		
Type or Unit	uint8_least		
Range	min : 1		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	Instance		
Dependency	WDGM074, WDGM090		

Name	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.		
Type or Unit	uint8_least		
Range	min : 1		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1

Scope	instance
Dependency	WDGM074, WDGM090

Name	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.		
Type or Unit	unit8_least		
Range	min : 0		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM095, WDGM097		

Name	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*.		
Type or Unit	unit8_least		
Range	min : 0		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM074, WDGM091		

Name	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*.		
Type or Unit	unit8_least		
Range	min : 0		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	Instance		
Dependency	WDGM074, WDGM091		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
none		

10.2.6 Watchdog Manager Trigger Parameters

SWS Item	**WDGM092:**
Container Name	WdgMTrigger
Description	This container collects all configuration parameters for the triggering of hardware watchdogs
Configuration Parameters	

Name	WDGM_TRIGGER_CYCLE		
Description	This parameter defines the schedule period of the main function WdgM_MainFunction_Trigger.		
Type or Unit	ms		
Range	Application specific		
	--		
Configuration Class	***Pre-compile***	X	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	ECU (integration)		
Dependency	--		

Name	WDGM_NUMBER_OF_WATCHDOG_INSTANCES		
Description	This parameter shall contain the amount of watchdog instances.		
Type or Unit	uint8_least		
Range	min : 1		
	max : --		
Configuration Class	***Pre-compile***	X	Variant 1
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgMWatchdogInstance	1..*	One set of parameter of this container is required per watchdog instance.

10.2.7 Watchdog Manager Watchdog Instance Parameters

SWS Item	**WDGM094:**
Container Name	WdgMWatchdogInstance
Description	This container collects all configuration parameters to define the individual settings of a watchdog instance
Configuration Parameters	

Name	WDGM_WATCHDOG_INSTANCE_ID	
Description	This parameter shall contain the identifier of the watchdog instance	
Type or Unit	WdgM_WatchdogInstanceIdType	
Range	--	

	--		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM003		

Name	WDGM_TRIGGER_SLOW_REFERENCE_CYCLE		
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 instance in slow mode.		
Type or Unit	uint8_least		
Range	min : 1		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM116		

Name	WDGM_TRIGGER_FAST_REFERENCE_CYCLE		
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 instance in fast mode.		
Type or Unit	uint8_least		
Range	min : 1		
	max : --		
Configuration Class	***Pre-compile***	--	
	Link time	--	
	Post Build	x	Variant 1
Scope	instance		
Dependency	WDGM116		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
None		

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).

SWS Item	**WDGM012:** The following table lists the parameters that shall minimally be provided in the module's header file `wdgM.h` for use by other modules and also in the module's description file.	
Information elements		
Information element name	***Type / Range***	***Information element description***
`WDGM_VENDOR_ID`	`#define/ uint16`	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
`WDGM_MODULE_ID`	`#define/ uint8`	Module ID of this module from Module List
`WDGM_AR_MAJOR_VERSION`	`#define/ uint8`	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
`WDGM_AR_MINOR_VERSION`	`#define/ uint8`	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
`WDGM_AR_PATCH_VERSION`	`#define/ uint8`	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
`WDGM_SW_MAJOR_VERSION`	`#define/ uint8`	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
`WDGM_SW_MINOR_VERSION`	`#define/ uint8`	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
`WDGM_SW_PATCH_VERSION`	`#define/ uint8`	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

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.