Safe Watchdog Manager

User Manual

Version:	3.3.1
Date:	22.05.2014
Document number:	D-MSP-M-70-001

TTTech Automotive GmbH
Schoenbrunner Str. 7, A-1040 Vienna, Austria, Tel. + 43 1 585 34 34-0, Fax +43 1 585 34 34-90, support@tttech-automotive.com

Table of Contents

1 Introduction 4

1.1 Architecture Overview 5

1.2 Use Cases 7

1.3 Safe Watchdog Manager Stack Content 8

2 Safe Watchdog Manager (S-WdgM) 9

2.1 File Structure 10

2.2 Basic Functionality of the S-WdgM 13

2.2.1 Supervised Entity and Program Flow Supervision 13

2.2.2 Deadline Monitoring 15

2.2.3 Alive Supervision 18

2.2.4 More Details on Checkpoints and Transitions 21

2.2.5 Global Transitions 22

2.2.6 Global Transitions and Program Flow 24

Example of an Incorrect Global Transition Split 24

Example of an Incorrect Program Split in the Middle of an Entity 25

2.2.7 S-WdgM Supervision Cycle 25

2.2.8 S-WdgM Stack Fault Reaction Time 27

2.2.9 Reset Path and Safe State 30

2.2.10 S-WdgM Local Entity State 31

2.2.11 S-WdgM Global State 33

2.3 Integration in AUTOSAR 3.1 and 4.0 Environments 33

2.4 Deviations from the AUTOSAR 4.0 r1 Watchdog Manager 34

2.4.1 Entities, Checkpoints and Transitions 34

2.4.2 Tolerances 36

2.4.3 Watchdog and Reset 36

2.4.4 API 36

2.5 Configuration Parameters for the S-WdgM 38

2.5.1 S-WdgM Global Preprocessor Settings 38

2.5.2 S-WdgM General Settings 49

2.5.3 S-WdgM Supervised Entity Options 57

2.5.4 S-WdgM Checkpoint Options 65

2.5.5 Alive Counter Options 65

2.5.6 S-WdgM Local Transition Options 67

2.5.7 S-WdgM Global Transition Options 68

2.5.8 S-WdgM Local and Global Deadline Options 69

2.6 ECU Description Configuration 72

2.6.1 Assumptions/Constraints 72

2.7 API Description 73

2.7.1 S-WdgM Type Definitions 73

2.7.2 S-WdgM Application Level API Functions 76

2.7.3 Callback Functions 84

2.7.4 S-WdgM System Level API Functions 84

2.7.5 Expected Interfaces 88

2.7.6 AUTOSAR 3.1 Compatibility Mode 90

User API 90

System API 92

3 Integration 94

3.1 Initialization of the S-WdgM 94

3.2 Memory Sections 95

3.3 Timing Setup 97

3.3.1 Deadline Measurement and Tick Counter 100

4 Configuration Generation 102

4.1 S-WdgM Configuration Generator 102

4.1.1 S-WdgM Configuration Verification 103

Installing the S-WdgM Verifier 104

4.2 Workflow 105

4.3 Output Files 107

4.4 Error Messages 108

4.4.1 Basic Errors 108

4.4.2 Semantic Errors 108

5 Appendix 114

5.1 Watchdog Manager Configuration Verifier Requirements 114

5.1.1 General Remarks 114

5.1.2 General Requirements 114

5.1.3 Deltas the S-WdgM Verifier Must Detect between the EDF and the Generated Configuration 114

5.1.4 Integrity Checks 118

5.1.5 Errors To Be Detected by the Verifier to Protect the Embedded Code 120

6 Abbreviations 122

7 Glossary 124

8 References 128

9 License Information 129

1 Introduction

The **Safe Watchdog Manager (S-WdgM) Stack** provides software modules to **monitor** the **correct functioning** of safety-relevant activities in systems with software **modules of mixed criticality**, such as

- newly developed safety-related functions,
- legacy functions, and
- basic software.

The **S-WdgM Stack** is designed to be used in automotive ECUs.

The **S-WdgM Stack** has three software modules

- **Safe Watchdog Manager (S-WdgM)**
- **Safe Watchdog Interface (S-WdgIf)**
- **Safe Watchdog Driver (S-Wdg)**

The S-WdgM can run on **single-core** and **multi-core** systems.

This user manual describes the **S-WdgM**, which is an AUTOSAR basic software module that is part of the AUTOSAR service layer. The **S-WdgM** checks the logical program flow and temporal behavior of the program flow of safety-relevant functions. Safety-relevant functions use **checkpoint calls** to send **life signs** to the **S-WdgM**. Internal or external watchdog hardware is used independently from the system CPU **to monitor**

- if the system is still **alive**,
- if the system **functions** properly, and
- if the system shows the **correct temporal behavior** and **logical program flow**.

The S-WdgM was developed according to **AUTOSAR version 4.0 r1** [1] 128. However, its functionality can be restricted to the functionality described by **AUTOSAR 3.1 r4** in the AUTOSAR 3.1 compatibility mode.

The S-WdgM is designed to be integrated into **AUTOSAR 3.1.4** or **4.0.1** compatible environments. However, it is not restricted to these AUTOSAR versions only. The software module can also be integrated into other versions of AUTOSAR and other system software architectures if the integration-related requirements listed in the *Safe Watchdog Manager Safety Manual* [5] 128 are met.

The S-WdgM is compatible with the **AUTOSAR 4.0 r1 Watchdog Manager**, but not fully compliant. For deviations from the AUTOSAR 4.0 r1 specification, see Section Deviations from the AUTOSAR 4.0 r1 Watchdog Manager 34.

This user manual does **not** cover safety-related topics. For safety-related requirements for integration and application of the S-WdgM, refer to the *Safe Watchdog Manager Safety Manual* [5] 128.

1.1 Architecture Overview

The **S-WdgM Stack** consists of the **hardware-independent** modules **Safe Watchdog Manager** and **Safe Watchdog Interface** and a **hardware-dependent** module, the **Safe Watchdog Driver**.

Figure 1 shows the S-WdgM Stack with its modules in an AUTOSAR environment.

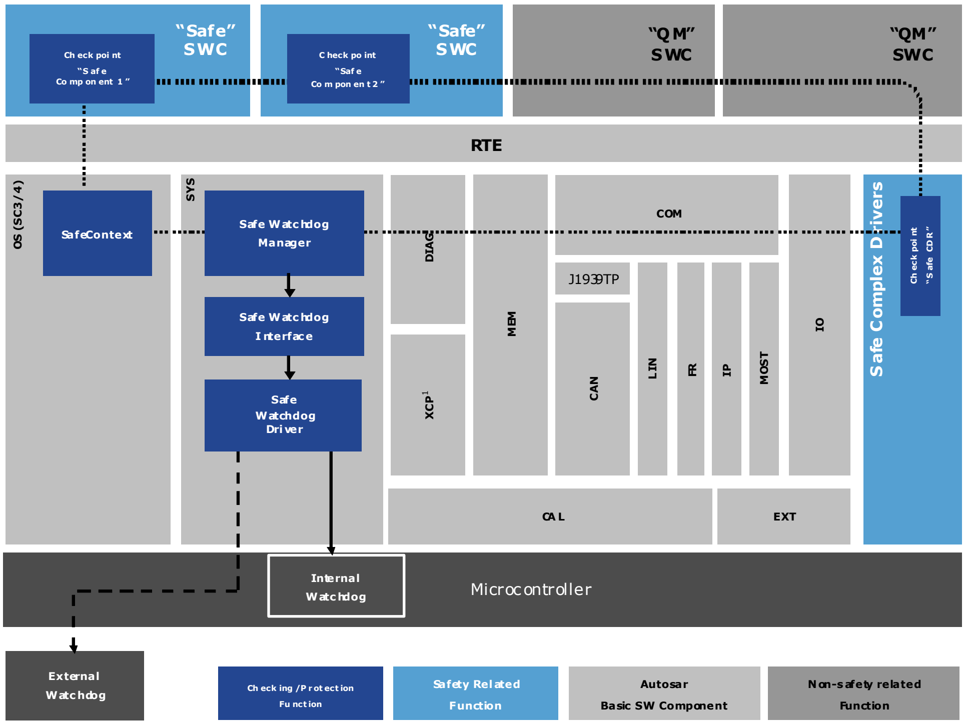


Fig 1: Safe Watchdog Manager Stack in an AUTOSAR environment

The **S-WdgM** controls, through the **S-WdgIf** and the **S-Wdg**, the hardware-implemented watchdog controller, which can be one or more internal watchdog controllers or external watchdog devices.

Note: A watchdog device requires a hardware-dependent S-Wdg driver.

Figure 2 shows the layered structure of the S-WdgM Stack. The attached watchdog device can be internal or external.

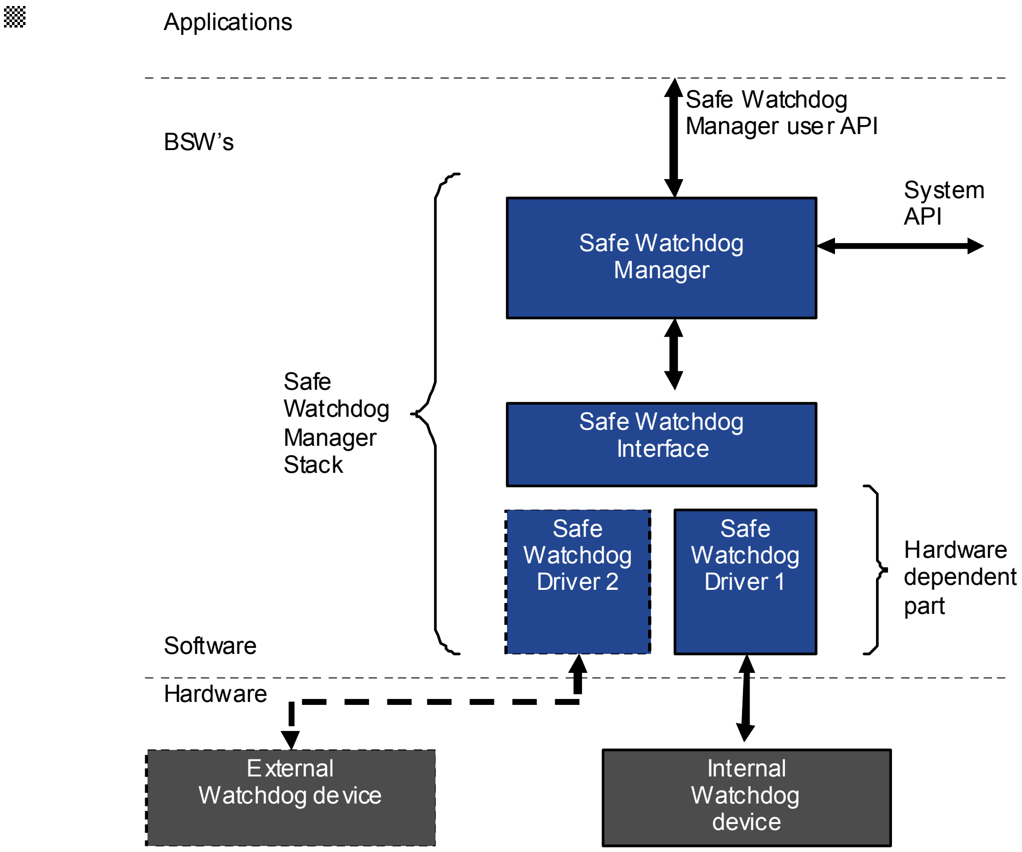


Fig. 2: Layered structure of the Safe Watchdog Manager

The **S-WdgM monitors** the **program flow** and **timing constraints** of so-called **supervised entities (SE)**. The SEs are software entities (like application software) that are supervised by the S-WdgM. When the S-WdgM detects a violation of the preconfigured program flow or the timing values, it takes a number of configurable actions to log that violation and/or go to a safe state (for details, see Section Safe Watchdog Manager (S-WdgM)[9]). The S-WdgM communicates with the system via the **Safe Watchdog Application Interface (API)** (see Section API Description)[73].

1.2 Use Cases

The S-WdgM monitors the user software at runtime and compares the preconfigured logical and temporal constraints with the actual logical and temporal behavior. The S-WdgM can monitor the following violations:

- timing violation (checked by **deadline monitoring** and **alive monitoring**)
- program flow violation (checked by **program flow monitoring**)

The S-WdgM periodically triggers the watchdog device through its **interface** (**S-WdgIf**) and **driver** layer (**S-Wdg**). When the S-WdgM detects a fault in the program flow or timing, then it stops the watchdog triggering, or it initiates a reset of the microcontroller immediately or after a delay, depending on the S-WdgM configuration.

The S-WdgM monitors the following **software** and **hardware faults**:

- The supervised entity is executed but the execution was not requested.
- The supervised entity was not executed but the execution was requested.
- The execution of the supervised entity started too early or too late.
- The execution time of a a supervised entity or part of a supervised entity or many supervised entities is longer or shorter than expected.
- The program flow of a a supervised entity or part of a supervised entity or many supervised entities differs from expected program flow.

The reaction of the S-WdgM to detected faults can be configured as follows:

- S-WdgM sends information about the detected fault.
- S-WdgM initiates a reset of the microcontroller after a watchdog timeout.
- S-WdgM initiates an immediate reset of the microcontroller.

1.3 Safe Watchdog Manager Stack Content

The **Safe Watchdog Manager Stack** consists of:

Embedded code:

- **Safe Watchdog Manager (S-WdgM)** software module
- **Safe Watchdog Interface (S-WdgIf)** software module
- **Safe Watchdog Driver (S-Wdg)** software modules

A part of the embedded code is generated out of a given ECU configuration.

S-WdgM Configuration Generators (which generate a part of the embedded code out of a given ECU configuration):

- Safe Watchdog Manager Generator
- Safe Watchdog Interface Generator
- Safe Watchdog Driver Generator
- Safe Watchdog Manager Configuration Verifier

Configuration example:

- An example of an ECU configuration and the generated code.

Documentation:

- **User Manuals** covering the
  - Safe Watchdog Manager,
  - Safe Watchdog Interface, and
  - Safe Watchdog Drivers
- **Safety Manuals** covering the
  - Safe Watchdog Manager
  - Safe Watchdog Interface
  - Safe Watchdog Drivers

2 Safe Watchdog Manager (S-WdgM)

The **S-WdgM** monitors safety-relevant applications on the ECU. The S-WdgM is a **basic software module** at the service layer of the standardized basic software architecture of **AUTOSAR**. The S-WdgM monitors the program flow of a configurable number of so-called **supervised entities (SE)**. When the S-WdgM detects a violation of the preconfigured temporal or logical constraints in the program flow, it takes a number of configurable actions to log the fault and to go to a safe state after a configurable time delay. The safe state is reached by resetting the watchdog or by omitting watchdog triggering.

Every supervised entity has a defined control flow. Significant points in this control flow are represented by **checkpoints (CP)**. This means the **control flow** can be modeled as a **graph**, with the **checkpoints** being the **nodes** and the pieces of **code** in between being the **transitions** (see Figure 4 for an example).

The S-WdgM configuration defines the allowed transitions between the checkpoints, and the timing constraints for these transitions

- within every supervised entity and
- between checkpoints of different supervised entities.

The supervised entities have to report to the S-WdgM when they have reached a checkpoint. Thus, the developer has to insert calls at the checkpoints that pass this information to the S-WdgM.

The S-WdgM functionality partially deviates from the AUTOSAR requirements. For details, refer to Section Deviations from the AUTOSAR 4.0 r1 Watchdog Manager 34.

2.1 File Structure

Figure 3 gives an overview of the **S-WdgM module**.

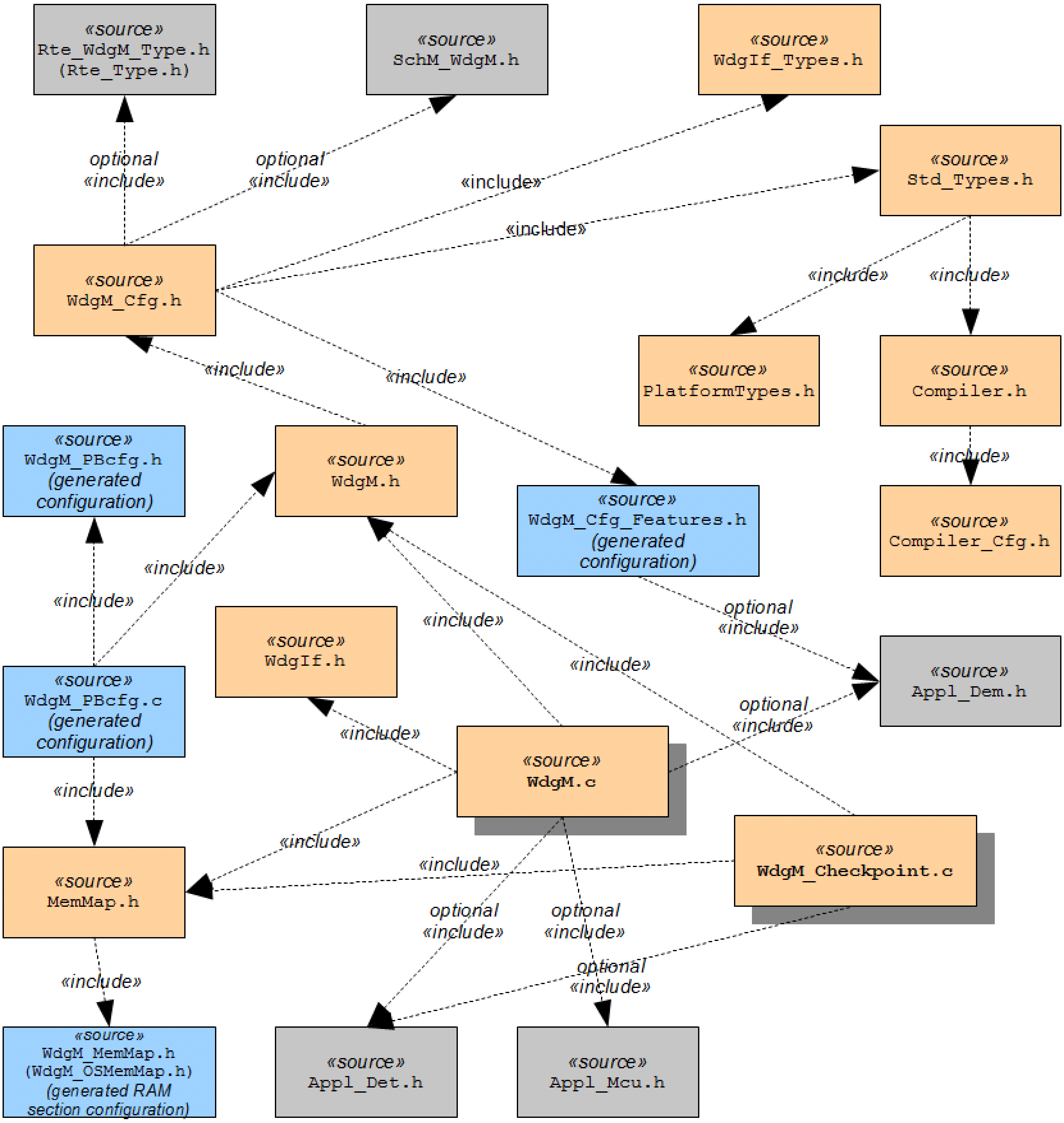


Fig. 3: File structure of the S-WdgM module

Note: The file structure shown in Figure 3 corresponds to the integration of the S-WdgM in an **AUTOSAR 3.1** environment. The differences between an **AUTOSAR 3.1** and an **AUTOSAR 4.0** environment are described below in the following two tables.

The following files are part of the S-WdgM:

File	Description
`WdgM.c`	Implementation of the S-WdgM, defines the API for the Service Layer of the BSW-Layer.
`WdgM_Checkpoint.c`	Implementation of the S-WdgM, defines the API for the Application Layer.
`WdgM.h`	Header file of the S-WdgM, provides API function declarations.
`WdgM_Cfg.h`	Provides defines and declarations for the S-WdgM configuration identifiers
`WdgM_MemMap.h` or **`WdgM_OSMemMap.h`**	The file is generated and contains defines for the memory management of the S-WdgM code and data. The integrator can place the status variables of every supervised entity in a separate RAM sector (see also Section Memory Sections 95). The file is included in the AUTOSAR **`MemMap.h`** file. **Note:** The name of this generated file is ▪ **`WdgM_MemMap.h`** in an **AUTOSAR 3.1** environment and ▪ **`WdgM_OSMemMap.h`** in an **AUTOSAR 4.0** environment.
`WdgM_Cfg_Features.h`	The file is generated and contains S-WdgM precompile directives.
`WdgM_PBcfg.h`	The file is generated and contains the declaration of the S-WdgM configuration.
`WdgM_PBcfg.c`	The file is generated and contains the S-WdgM configuration.

The following files are included by the S-WdgM, but are not part of the S-WdgM:

File	Description
`WdgIf_Types.h`	Provides the declaration of the S-WdgIf API.
`Std_Types.h`	AUTOSAR file
`Compiler.h`	AUTOSAR file

Compiler_Cfg	Contains compiler abstraction macros
PlatformTypes.h	AUTOSAR file
MemMap.h	AUTOSAR file. Includes **WdgM_MemMap.h**.
Appl_Det.h	Provides API to a wrapper function for Det_ReportError().*
Appl_Dem.h	Provides API to a wrapper function for Dem_ReportErrorStatus().* **Note:** In an **AUTOSAR 4.0** environment, this file is indirectly included by **WdgM.c**. It is included through the generated file **WdgM_Cfg_Features.h**.
Appl_Mcu.h	Provides API to a wrapper function for Mcu_PerformReset().*
Rte_Type.h or **Rte_WdgM_Type.h**	Provides generated RTE type definitions for the WdgM. **Note:** The name of this generated file is ▪ **Rte_Type.h** in an **AUTOSAR 3.1** environment and ▪ **Rte_WdgM_Type.h** in an **AUTOSAR 4.0** environment.
SchM_WdgM.h	Provides the API of the Schedule Manager for entering and exiting an exclusive area.

***)** The services

- Det_ReportError(),
- Dem_ReportErrorStatus() and
- Mcu_PerformReset()

may not meet the quality level required for the S-WdgM. These services must be wrapped by a wrapper service that has the same name as the corresponding AUTOSAR service with the prefix Appl_, which guarantees freedom from interference. The implementation of the wrapper service is not part of the S-WdgM. The *Safe Watchdog Manager Safety Manual* [5]128 provides a guideline on how to implement the wrapper.

NOTE: A wrapper could be just a direct call to the corresponding module, but that wrapper could also perform more complex operations such as switching the OS context before calling the service.

2.2 Basic Functionality of the S-WdgM

As described in *AUTOSAR* [1], the S-WdgM is a basic software module that monitors the program flow of **supervised entities (SE)**.

2.2.1 Supervised Entity and Program Flow Supervision

A **supervised entity** is a software part that is monitored by the S-WdgM. There is no fixed relationship between supervised entities and the architectural building blocks in AUTOSAR.

The **checkpoints** mark important steps during the execution of an algorithm. At the checkpoint, a supervised entity calls the function `WdgM_CheckpointReached()` 78 directly (if no runtime environment is present) or with a wrapper function (if a runtime environment is present), with that wrapper function being provided by the runtime environment. The checkpoints are connected by **transitions**. Local transitions bind Checkpoints to a **closed graph**. These graphs represent the program flow.

The S-WdgM knows which program flow is correct and decides if a supervised entity behaves as expected or violates the predefined rules.

The question of how to identify the checkpoints for an algorithm is a trade-off between performance and code block size per checkpoint:

- The more checkpoints an algorithm has, the better is the representation of the code structure. But this has an adverse effect on performance.
- However, if an algorithm has only a few checkpoints, then there are code segments and program flow branches that are not represented. In this case, performance will be better, but not everything will be monitored.

A supervised entity can represent an **algorithm**, a **function**, or – in the case of an operating system – an **entire task**. In the AUTOSAR definition, a supervised entity can be distributed over more than one task or application. There can be several supervised entities for the same task. However, the S-WdgM implementation does not support the distribution of one supervised entity over more than one task or application when they run in different contexts. The S-WdgM expects that at least one supervised entity and at least one checkpoint are defined.

Figure 4 shows the example of a simple supervised entity called `temperature_control`:

- Supervised entity `temperature_control` has six checkpoints (*illustrated by ovalboxes*), which are connected by directed transitions (*illustrated by arrows*).
- As can be seen in Figure 4, it is possible to reach the checkpoint `temperature_needs_correction` after the checkpoint `read_temperature`.
- However, reaching the checkpoint `heater_adjusted_successfully` after the checkpoint `read_temperature` would be a violation of the program flow.

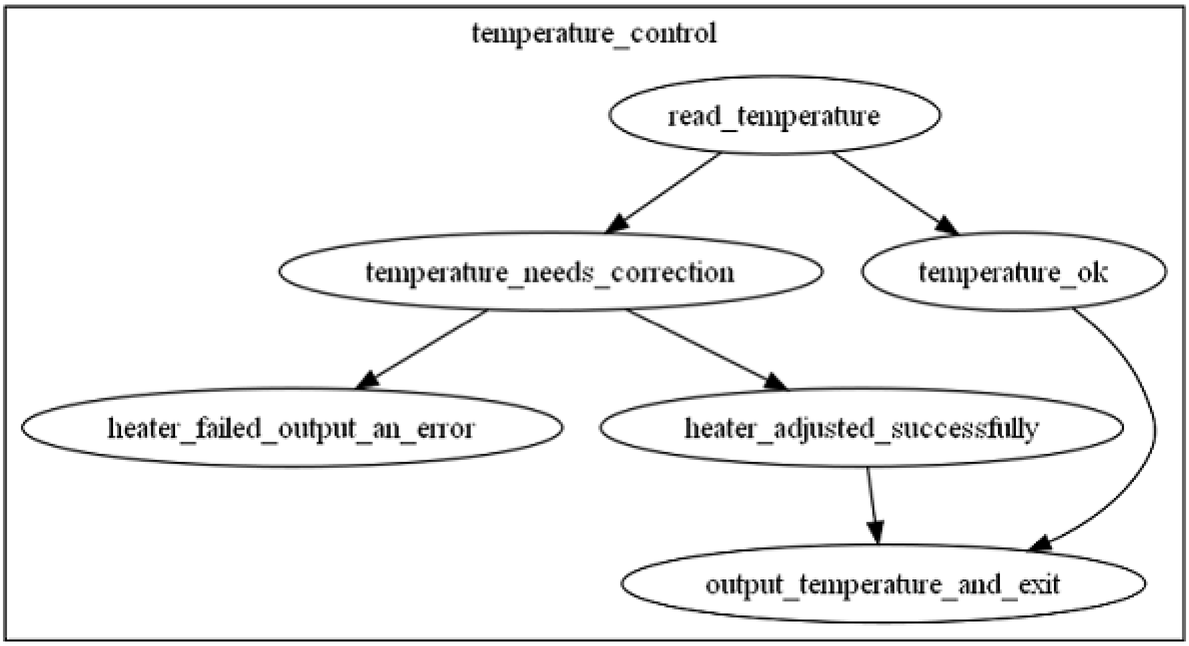

temperature_control
read_temperature
temperature_needs_correction
temperature_ok
heater_failed_output_an_error
heater_adjusted_successfully
output_temperature_and_exit

Fig. 4: Example of a simple supervised entity with a control flow

Use program flow monitoring

Control (program) flow monitoring is highly recommended by ISO 26262-6 (7.4.14). Apart from its main feature, which is to detect logical errors in the monitored algorithms, program flow monitoring increases the probability of detecting illegal program counter jumps within the whole system.

It is possible to tolerate **program flow violations** within a supervised entity for a **certain** amount of **monitoring cycles**. it is possible to define a **program flow reference cycle** (a multiple of the S-WdgM monitoring cycle) and a tolerance, which is a number of program flow reference cycles, during which program flow violations should be tolerated for the supervised entity. If a program flow violation is detected for more program flow reference cycles than the defined tolerance, then the supervised entity changes its status from `FAILED` to `EXPIRED`.

The necessary configuration parameters to tolerate program flow violations of a supervised entity are:

- `WdgMFailedProgramFlowRefCycleTol` [59]:This parameter contains the acceptable amount of program flow violations for this supervised entity.
- `WdgMProgramFlowReferenceCycle` [60]:This parameter contains the amount of supervision cycles to be used as reference by the program flow supervisions of this supervised entity.

Note: The program flow reference cycle for a supervised entity starts with the first detected program flow violation and not with the S-WdgM startup. Hence, the first program flow reference cycle starts with the transition of the supervised entity from status `OK` to `FAILED`. If no program flow violation is detected for a whole program flow reference cycle within the tolerance then the supervised entity recovers and changes its status from FAILED to OK. Otherwise, if the tolerance is exhausted and the program flow violations continue, then the supervised entity changes its status to EXPIRED. It can be said that the program flow reference cycle is processed only during the status FAILED – it starts with the first detected program flow violation. The program flow reference cycle is restarted with each following transition from OK to FAILED, and it is not processed during the status OK, EXPIRED or DEACTIVATED.

2.2.2 Deadline Monitoring

The main purpose of deadline monitoring is to check the **temporal**, **dynamic behavior** of the supervised entity. However, it would also strongly increase the probability of detecting random jumps or irregular updates of the timebase tick counter, which might otherwise degrade system integrity without being discovered.

The **temporal behavior** of the supervised entities can be monitored by assigning **deadlines** to **transitions**.

- A **deadline** is defined through a **maximum deadline** (parameter WdgMDeadlineMax 69) and a **minimum deadline** (parameter WdgMDeadlineMin 69). The destination checkpoint of a transition should not be reached before the minimum time or after the maximum time after which the source checkpoint of that transition was reached. Otherwise the S-WdgM will detect a deadline violation. Apart from a maximum deadline time it is strongly recommended to use a minimum deadline time as well, where applicable. This allows discovering timebase tick counter errors implicitly. **Deadlines** are good for **discovering** crashed tasks or infinite loops. If the **destination checkpoint** is never reached because the task ended with an error or is stuck in a loop, this would cause a deadline violation.
- A **transition** is considered to violate its deadline if the destination checkpoint is not hit within the configured deadline interval. A deadline is assigned to an already defined transition by specifying the same source and destination checkpoints as for the transition. The corresponding deadline parameters are WdgMDeadlineStartRef 70 and WdgMDeadlineStopRef 70.

Note: A **transition** should be defined either as a **local** or a **global** transition.

- As for **local transitions**, the source and destination checkpoints belong to the same supervised entity.
- As for **global transitions**, the source and destination checkpoints belong to different supervised entities.

An example of a supervised entity with deadlines defined for its transitions is given below.

Note: The first deadline is defined to have a **minimum** of **0** and a **maximum** of **2** (**seconds**). Hence, **CP1** must be reached no later than 2 seconds after **CP0**. The second deadline implies that **CP2** must be reached no earlier than **1** and no later than **3** seconds after **CP1**. Otherwise a deadline violation will be detected.

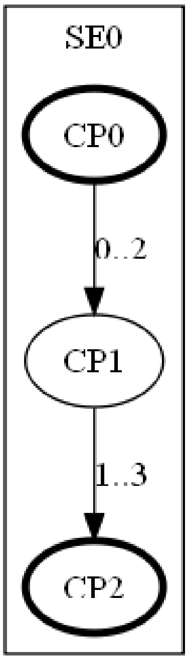


Fig. 5: Example of a simple supervised entity with deadlines

Note: Deadline violation is detected

- when the next checkpoint is reached outside the defined deadline or
- within the `WdgM_MainFunction()` 85 if the next checkpoint is not reached at all (or has not been reached yet) and the maximum deadline has already expired.

A slightly more complex situation is when several transitions go out of the same checkpoint. In this case, deadline violations are detected in the same manner when the next checkpoint is reached outside the defined deadlines. However, if none of the next checkpoints is reached, the `WdgM_MainFunction()` 85 detects a deadline violation only after the maximum of maximum deadlines of all outgoing transitions has elapsed, which is shown in Figure 6. If the program gets stuck after **CP0**, the deadline violation is detected within the next main function that is executed not earlier than **5 seconds** after reaching **CP0**.

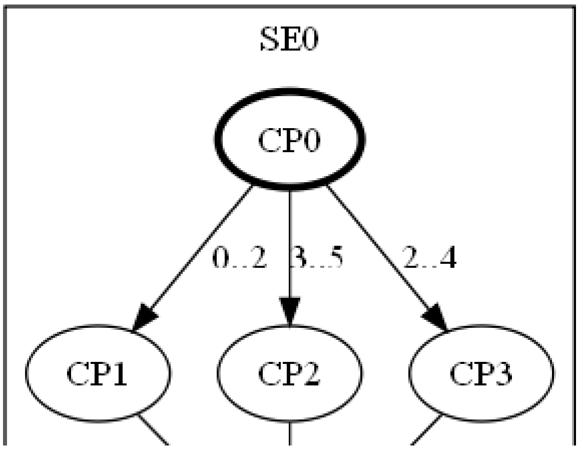


Fig. 6: Example of multiple outgoing transitions with deadlines

A special case is a hybrid situation when some of the outgoing transitions have deadlines and others do not. In this case, the main function detects a deadline violation if none of the next checkpoints is reached within the maximum of maximum deadlines in order to detect blocked supervised entities. No deadline violation will be detected after the maximum has expired, however, if the checkpoint without deadline is reached before the main function. If none of the **CP1**, **CP2** is reached after **CP0** (7), then the next WdgM_MainFunction() 85 (executed at least **2 seconds** after **CP0** is reached) detects a deadline violation. If, however, **CP1** is reached after **2 seconds**, but before the next WdgM_MainFunction() 85, no deadline violation would be detected.

Note: To avoid this ambiguous situation it is a good practice to define deadlines for all outgoing transitions of a checkpoint (or for none of them).

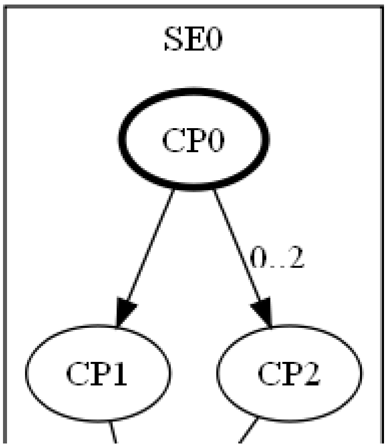


Fig. 7: Example of a the case where only one of several outgoing transitions has a deadline

The rules for deadline violation detection also apply to global transitions or to the case of local transitions mixed with global transitions at a checkpoint.

It is possible to tolerate **deadline violations** within a supervised entity for a certain amount of monitoring cycles. It is possible define a **deadline reference cycle** (a multiple of the S-WdgM monitoring cycle) and a tolerance, which is a number of deadline reference cycles, during which deadline violations should be tolerated for the supervised entity. If a deadline violation is detected for more deadline reference cycles than the defined tolerance, then the supervised entity changes its status from `FAILED` to `EXPIRED`.

The necessary configuration parameters to tolerate deadline violations of a supervised entity are:

- `WdgMFailedDeadlineRefCycleTol` 58: This parameter contains the acceptable amount of violated deadlines for this supervised entity.
- `WdgMDeadlineReferenceCycle` 59: This parameter contains the amount of supervision cycles to be used as reference by the deadline supervisions of this supervised entity.

Note: The deadline reference cycle for a supervised entity starts with the first detected deadline violation and not with the S-WdgM start up. Hence, the first deadline reference cycle starts with the transition of the supervised entity from the status `OK` to `FAILED`. If no deadline violation is detected for a whole deadline reference cycle within the tolerance, then the supervised entity recovers and changes its status from `FAILED` to `OK`. Otherwise, if the tolerance is exhausted and the deadline violations continue, then the supervised entity changes its status to `EXPIRED`. It can be said that the deadline reference cycle is processed only during the status `FAILED` – it starts with the first detected deadline violation. The deadline reference cycle is restarted with each following transition from `OK` to `FAILED`, and it is not processed during the status `OK`, `EXPIRED` or `DEACTIVATED`.

2.2.3 Alive Supervision

Aliveness monitors the **frequency of hits** of checkpoints. For example, the algorithm could expect a sensor to report its measurements on a regular basis, and a certain task needs to process this data periodically. If a task stops reporting (alive sign is lost or too infrequent) or starts reporting too often, then the aliveness of that task is violated.

Alive supervision is associated with a **checkpoint** in a **supervised entity**. If you need to monitor **only** the **frequency** with which a task is called, you can make it a supervised entity that contains **only one checkpoint** with the corresponding aliveness parameters.

Note: Irregular calls of the S-WdgM main function or the omission of calls of `WdgM_CheckPointReached()` 78 would most likely result in **aliveness violation**. When alive monitoring for a checkpoint is activated, then that checkpoint must be **regularly called** for the entire period during which the supervised entity is active, otherwise aliveness violation will be detected. In the first supervision cycle, the Alive counter evaluation can be suppressed by the parameter `WdgMFirstCycleAliveCounterReset` 48.

It is important to consider which aliveness parameters are better for a specific situation. The example below shows how to choose the **appropriate** alive supervision **parameters**.

- Let a supervised entity with one checkpoint monitor the **aliveness** of a **task**.
- The S-WdgM has a period of **20ms**, one S-WdgM tick is **1ms**.
- The task is periodic with a fixed period of **30ms**.
- The **aliveness parameters** that must be set are:
    - WdgMExpectedAliveIndications[65]:
      Defines how many alive indications (checkpoint reached calls) are expected within one supervision reference cycle.
    - WdgMSupervisionReferenceCycle[66]:
      Defines the supervision reference cycle length as a number of supervision cycles (WdgMSupervisionCycle[99]).
    - WdgMMinMargin[66]:
      Defines the lower tolerance of expected alive indications.
    - WdgMMaxMargin[66]:
      Defines the upper tolerance of expected alive indications.
    - Hence, the allowed number of indications is in the range
      `WdgMSupervisionReferenceCycle` is in the range
      `[WdgMExpectedAliveIndications - WdgMMinMargin,`
      `WdgMExpectedAliveIndications + WdgMMaxMargin]`

Note: In contrast to the deadline and program flow reference cycle the alive supervision cycle begins with the S-WdgM startup. The alive supervision in the very first cycle can be influenced by the parameter WdgMFirstCycleAliveCounterReset[48]. This is because each alive counter is evaluated once per supervision reference cycle. This means that the supervision reference cycle is processed from the system startup on and during the status `OK` and `FAILED` of the corresponding supervised entity. If the supervised entity is in the status `EXPIRED`, then the supervision reference cycle is not needed anymore. If the supervised entity is in the status `DEACTIVATED`, then the supervision reference cycle is frozen. It is restarted if the supervised entity is activated again.

There are several ways for monitoring the task given in the example above. Below, **one variant** is given:

Set

- `WdgMExpectedAliveIndications=1`
- `WdgMSupervisionReferenceCycle=1`
- `WdgMMinMargin=1`
- `WdgMMaxMargin=0`

This means the S-WdgM should expect **1** or **0** (`WdgMExpectedAliveIndications`

- WdgMMinMargin) occurrences within one supervised reference cycle, which is fixed to **20ms** (which is **one S-WdgM supervision cycle**).

Figure 8 illustrates this example.

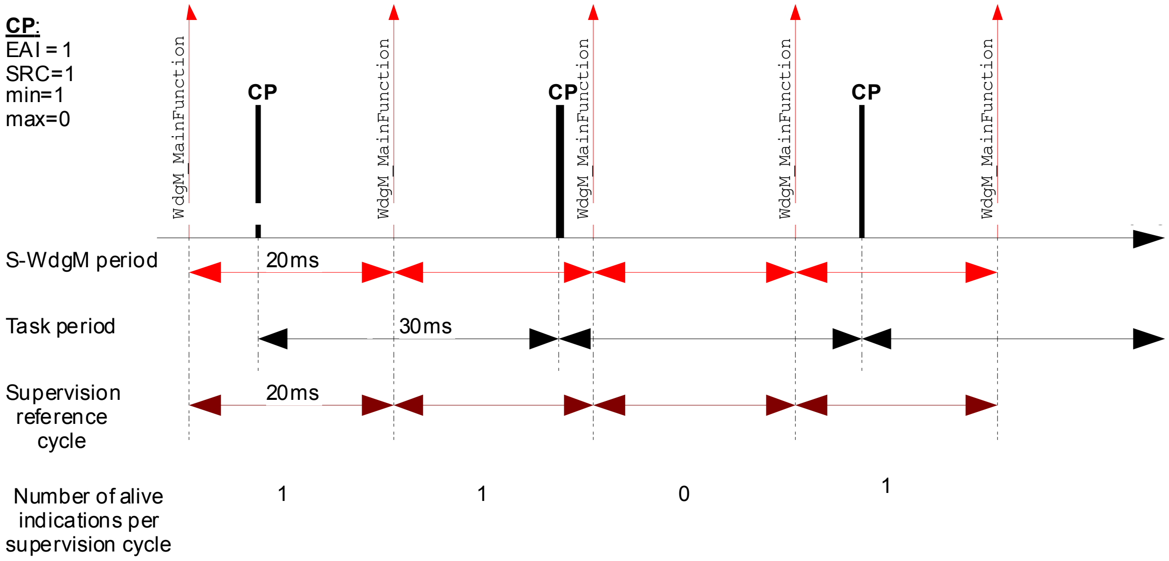


Fig. 8: A task being monitored during one S-WdgM supervision cycle (20ms)

However, if the task stops being executed it will not be detected, because **zero alive indications** per supervised reference cycle are **tolerated**. Therefore, this choice of setting aliveness parameters is not very good.

Below, a **second variant** is given:

Set

- WdgMExpectedAliveIndications=2
- WdgMSupervisionReferenceCycle=2
- WdgMMinMargin=1
- WdgMMaxMargin=0

This means the S-WdgM should expect **1** or **2** alive indications within one supervised reference cycle, which is fixed to **40ms** (and which is **two S-WdgM supervision cycles**).

Figure 9 illustrates this example.

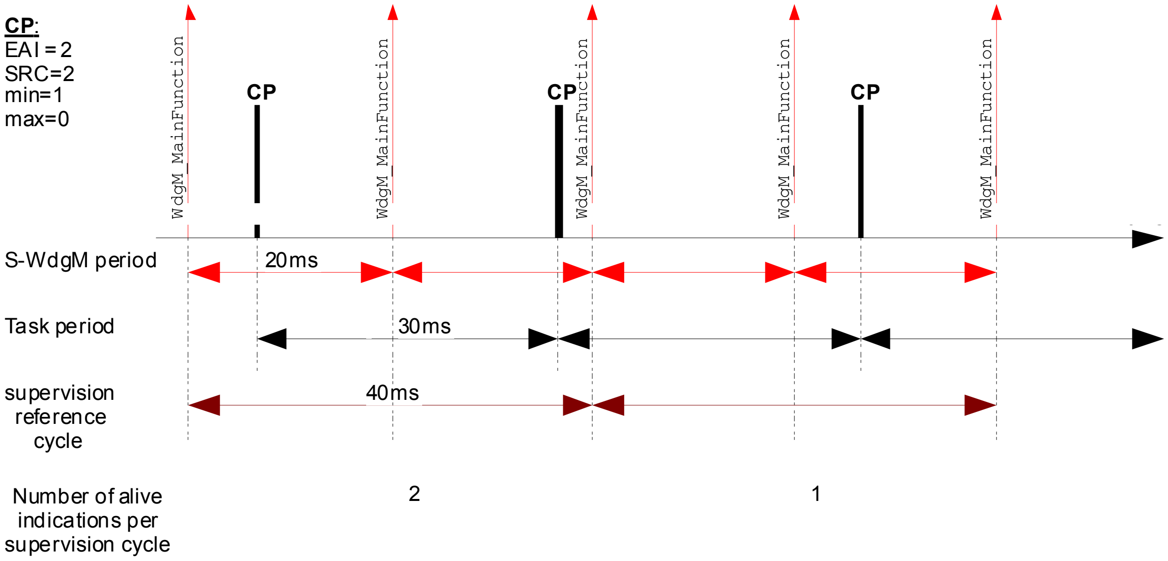


Fig. 9: A task being monitored during two S-WdgM supervision cycles (40ms)

This configuration solves the problem of detecting the disappearance of the task. However, the reaction time for error detection doubles from **20** to **40ms**.

A **third variant** would be to set the supervision reference cycle to the **least common multiple** of the **S-WdgM supervision cycle** and the **task period**. In the example given above this would be **60ms** (**three S-WdgM supervision cycles**). In this case, we expect exactly **2 alive indications**. Hence, the minimum and maximum margins are both `0`.

Note: The **task period** and the **S-WdgM supervision cycle** must be **synchronized** and started with an offset to each other (e.g., scheduled in an operating system).

2.2.4 More Details on Checkpoints and Transitions

Every supervised entity has one **initial checkpoint**. The number of **end checkpoints** can be zero, one or more than one. If the supervised entity contains only one single checkpoint, then it should be both an initial and an end checkpoint. **Local transitions** are defined by their **source** and **destination checkpoints**, which must belong to the same supervised entity. Those local transitions are specified in the parameters `WdgMLocalTransitionSourceRef` 68 and `WdgMLocalTransitionDestRef` 67.

After initialization of the S-WdgM, all supervised entites are passive.

Note: This has nothing to do with the supervised entity state `WDGM_LOCAL_STATUS_DEACTIVATED` 82.

A supervised entity becomes active when its local initial checkpoint has been called. In the example of the supervised entity `temperature_control` (see Section Supervised Entity and Program Flow Supervision 13 and Figure 4), the initial checkpoint is `read_temperature`. Only if the supervised entity is active, its checkpoints (other than the initial checkpoint) may be reached, otherwise a program flow violation occurs. Reaching an end checkpoint, the supervised entity is set to passive state, and it can be activated again only through the initial checkpoint.

Reaching the initial checkpoint again after the supervised entity has been activated is a program flow violation.

Local reflexive transitions (from a checkpoint to itself) are allowed only when configured. The reflexive transitions cannot be defined for local initial or local end checkpoints.

Local initial checkpoints are not allowed to have local incoming transitions.

Local end checkpoints are not allowed to have local outgoing transitions.

2.2.5 Global Transitions

It is possible to represent program flow dependencies between supervised entities by using so-called **global transitions**. Global transitions are defined for the S-WdgM configuration by their source and destination checkpoints, which must belong to different supervised entities and which are specified by the parameters `WdgMGlobalTransitionSourceRef` 69 and `WdgMGlobalTransitionDestRef` 68. The end checkpoint of an supervised entity is usually connected to the initial checkpoint of another supervised entity, expressing a logical dependency between them. However, global transitions are allowed between any two checkpoints of any two supervised entities.

One must keep in mind several things when defining a global transition between two arbitrary checkpoints:

- If the source of the global transition is not a local end checkpoint, then the source entity will remain active. Program flow violation would occur if its initial checkpoint were reached again.
- If the destination checkpoint of the global transition is not a local initial checkpoint., the destination entity may not be active. Program flow violation would occur if a non-initial checkpoint of an inactive supervised entity were reached.
- Exactly one global initial checkpoint must be defined. The first global transition passed must have that checkpoint as a source.
- It is possible to define one or several global end checkpoints or none. Once the global end checkpoint served as a destination checkpoint of a global transition, no more global transitions are allowed (unless they are started with the global initial checkpoint).

Figure 10 shows a global transition between two supervised entities:

- The `pressure_sensor_task` gets the pressure value.

- The `control_pressure_task` calculates a reaction and reacts to the measured pressure. However, it can start only after the first task (`pressure_sensor_task`) has finished and after the pressure value has been obtained. This relation is shown by a global transition (see *dotted arrow*).
- Some transitions in Figure 10 have comments that show deadlines in milliseconds.
- Deadlines can also be defined for global transitions (see *dotted arrow*), where **1..5ms** means that the second task (`control_pressure_task`) should start not later than **5ms**, but not earlier than **1ms** after the first task has finished.

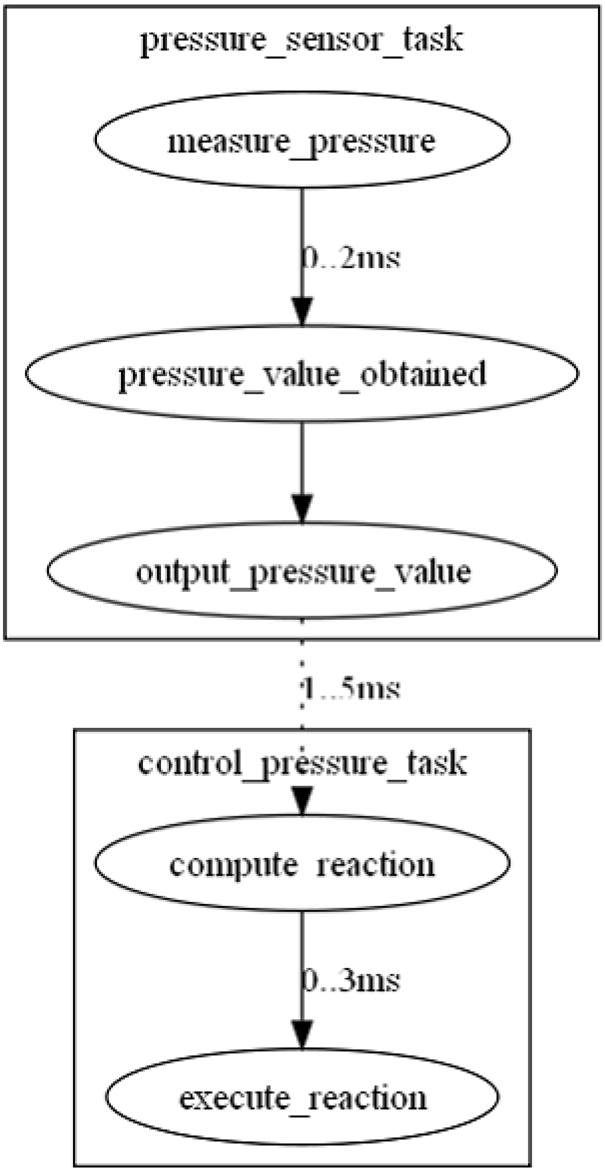


Fig. 10: Global transition between two supervised entities

2.2.6 Global Transitions and Program Flow

In general the, program flow does not differ between local and global transitions. But what seems intuitive for local transitions might not be so obvious for global transitions.. This section gives examples that show the usage of local and global transitions with a focus on program flow split.

From the perspective of the S-WdgM, the program flow is the consecutive reaching of checkpoints. The start of each program flow must be a local initial checkpoint. The program flow propagates through local transitions within the boundaries of a supervised entity and through global transitions within the boundaries of the whole system. The program flow might eventually come to an end at a local end checkpoint, or never come to an end if a program flow loop occurs.

A very important feature is that it is not allowed to split the program flow. This means that the program flow is allowed to take only one transition at each checkpoint from which more than one local or global transition comes out.

2.2.6.1 *Example of an Incorrect Global Transition Split*

Figure 11 shows that after checkpoint **cp0_1** the program flow must decide to take either the global transition **cp1_0** or **cp2_0**. Reaching **cp2_0** immediately after reaching **cp1_0** would result in a program flow violation.

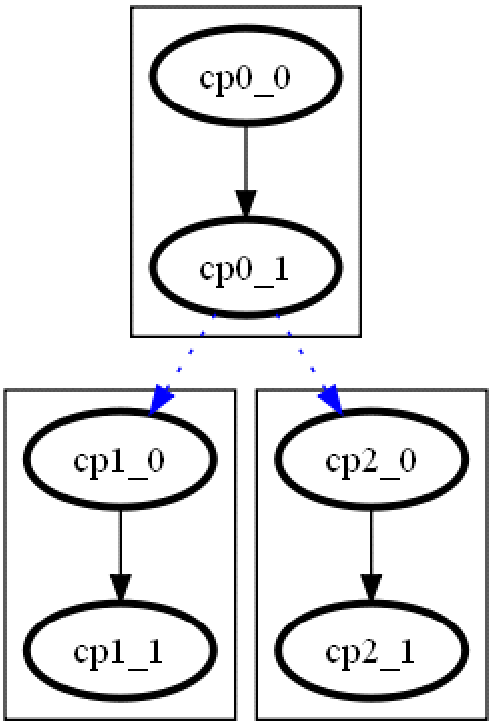


Fig. 11: Incorrect global transition split

2.2.6.2 *Example of an Incorrect Program Split in the Middle of an Entity*

Figure 12 shows another example. Let us assume that the program flow reaches **cp0_0** and then **cp0_1**. Afterward the program flow decides to take the global transition reaching **cp1_0** instead of taking the local transition. Now, if the local transition took place afterward (by reaching **cp0_2**), a program flow violation would occur. However, **cp0_2** can be reached via the global transition if the program flow comes from **cp1_1**.

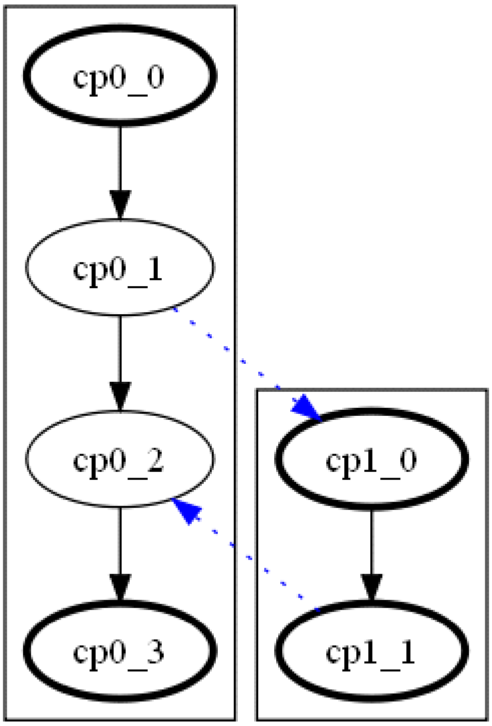


Fig. 12: Incorrect program split in the middle of an entity

Note: It is easy to create configurations with complex global transitions that do not make much sense in a real system. For example, if "jumping out" of a supervised entity from a checkpoint that is not a local end checkpoint, one must keep in mind that this supervised entity is still active (local activity flag is still `true`), and it cannot be restarted by reaching its local initial checkpoint again. Thus, it is recommended to use global transitions carefully and let them start only at local end checkpoints of a supervised entity and end at a local initial checkpoint of some other entity. Exceptions to this must be analyzed thoroughly, with respect to the program flow and the local activity of both supervised entities.

2.2.7 S-WdgM Supervision Cycle

The **supervision cycle** is the time period in which the cyclic supervision algorithm is executed. At the end of each supervision cycle, the **main function**, `WdgM_MainFunction()` 85, is called. This function evaluates the checkpoint data gathered in the previous period and triggers the Watchdog if no violation has been detected. Function `WdgM_MainFunction()` also checks for violations depending on the reference cycle defined for the respective monitoring feature.

Example: If `WdgMProgramFlowReferenceCycle` 60 `=3`, then the check for program flow violation is done in every third call of `WdgM_MainFunction()`.

The shorter this period and the reference cycles, the shorter the reaction time of the S-WdgM, but the more processor time is consumed.

Note: Aliveness supervision is strongly connected to this period. The expected number of **alive indications** for a certain checkpoint refers to the last supervision cycle (configurable for the checkpoint), which is expressed in the number of supervision cycles.

Figure 13 shows a time span with 3 supervision cycles. In each cycle, CP1 and CP2 are hit once. Once the S-WdgM main function is called, the window for the next watchdog trigger is defined by `WdgMTriggerWindowStart` 77 and `WdgMTriggerConditionValue` 77.

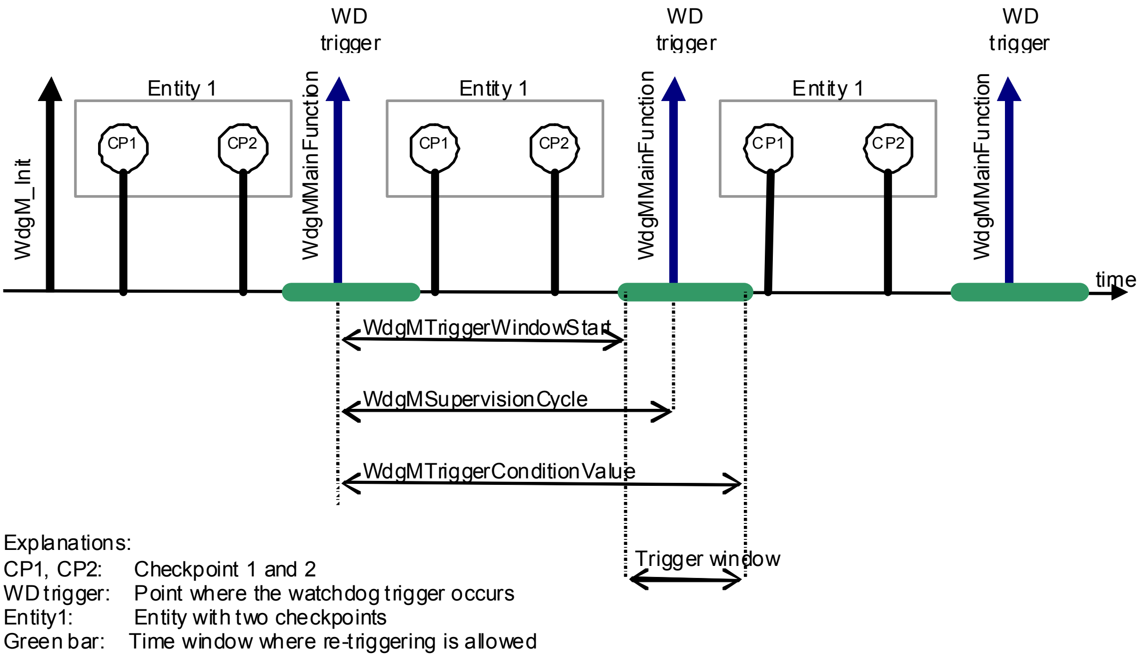

WD trigger
WD trigger
WD trigger
Entity 1
Entity 1
Entity 1
CP1
CP2
WdgM_Init
WdgMMainFunction
WdgMMainFunction
WdgMMainFunction
time
WdgMTriggerWindowStart
WdgMSupervisionCycle
WdgMTriggerConditionValue
Trigger window
Explanations:
CP1, CP2: Checkpoint 1 and 2
WD trigger: Point where the watchdog trigger occurs
Entity1: Entity with two checkpoints
Green bar: Time window where re-triggering is allowed

Fig. 13: S-WdgM supervision cycle

2.2.8 S-WdgM Stack Fault Reaction Time

The S-WdgM distinguishes between the **fault detection time** and the **fault reaction** time.

- The **fault detection time** spans from the **occurrence** of an **error** to the point in time when that error is detected and communicated to the system (via DET or callback functions).
- The **fault reaction time** spans from the **detection** of an **error** to the actual system reset.

If a **program flow violation** or a **deadline violation** occur, the source checkpoint **and** the destination checkpoint report to the S-WdgM when hit. At the end of the **current supervision cycle**, the S-WdgM main function, `WdgM_MainFunction()` ⇒85, is called and the violation is detected (ie. the configured destination checkpoint was hit too late or not at all) and communicated to the system.

If an **alive counter violation** occurs, it is also the S-WdgM main function that detects and communicates the violation at the **end** of the **supervision reference cycle** of the alive supervision.

Once a **violation** has been detected, the S-WdgM can (depending on the configuration)

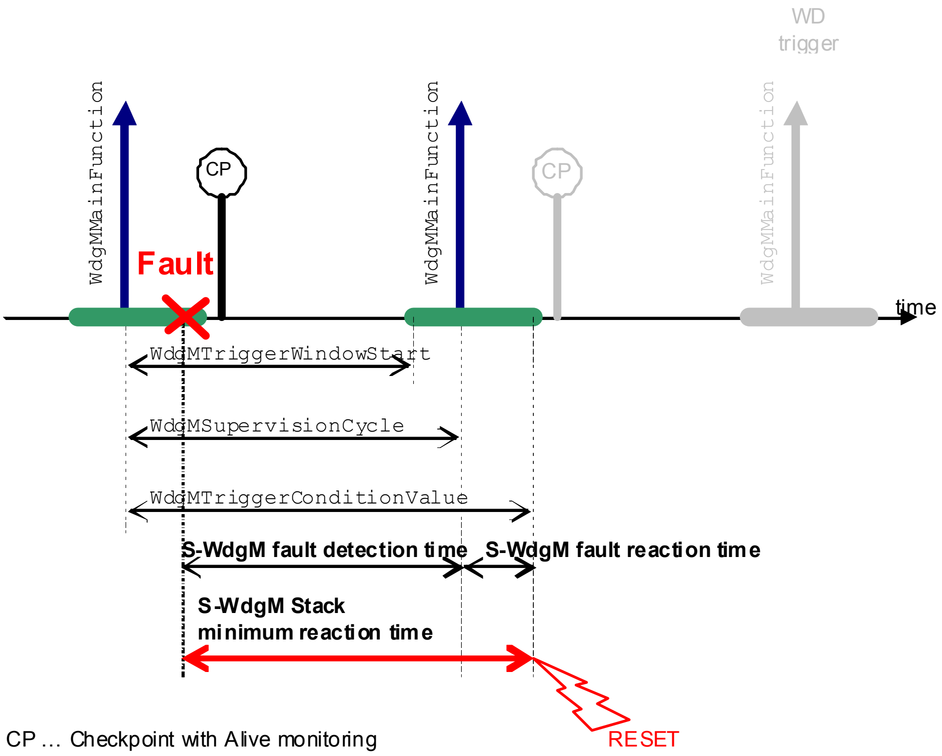
- immediately go to a **safe state** (ie. reset the WS or discontinue triggering the WD) or
- **allow** a configurable number of **violations** in a row and, hence, **delay** the **safe state** for this amount of supervision reference cycles.

The decision whether to trigger or reset the WD or not is made within the S-WdgM main function. This function also performs the trigger and reset.

The shortest fault detection and reaction time can be achieved by configuring an immediate reset. However, the time still depends on what occurs first in a supervision cycle, the fault or the hit of the checkpoint.

Figure 14 shows a scenario with a fault occurring first. The checkpoint registers the fault, and at the end of the current supervision cycle, the fault is detected, communicated, with the system being reset.

Note: For alive supervision, the detection is at the end of the current supervision reference cycle.

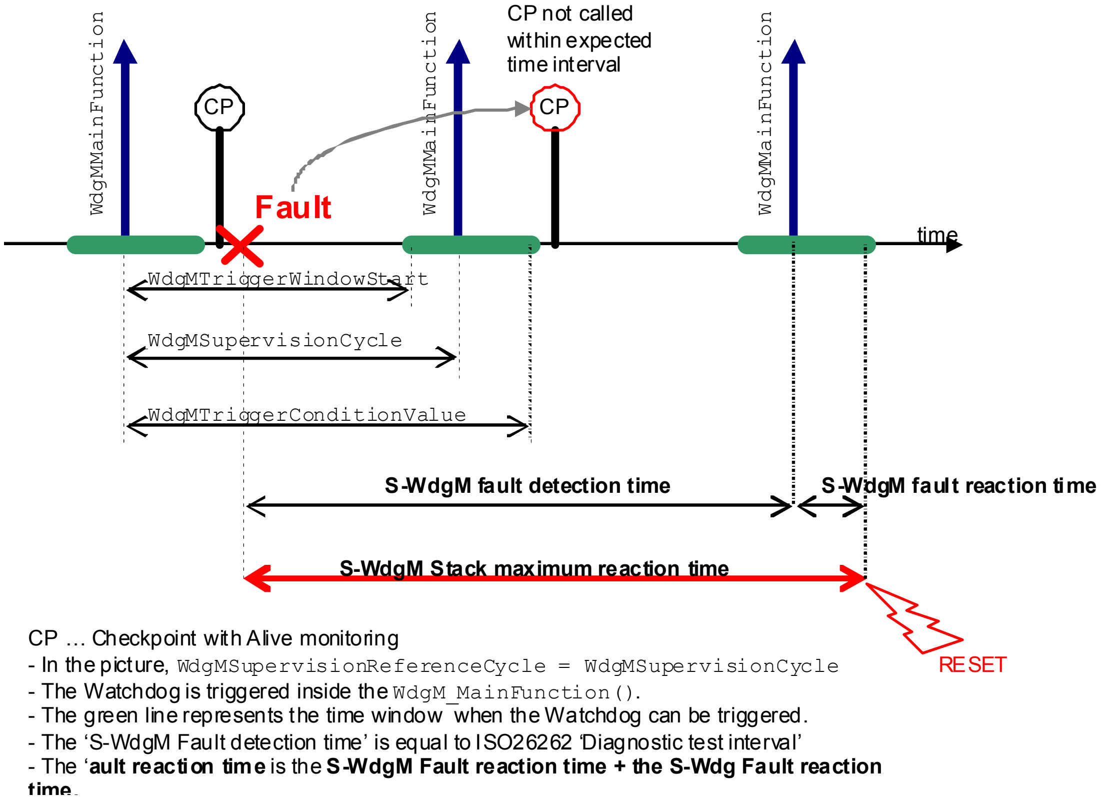


CP ... Checkpoint with Alive monitoring
- The `WdgMSupervisionReferenceCycle = WdgMSupervisionCycle`
- The Watchdog is triggered inside the `WdgM_MainFunction()`.
- The green line represents the time window when the Watchdog can be triggered.
- `WdgMImmediateReset = TRUE`

Fig. 14: The S-WdgM Stack minimum reaction time

Figure 15. shows a scenario with a checkpoint being hit first. The fault cannot be detected before the next checkpoint is hit, which is due to the subsequent supervision cycle. As a consequence, violation, detection, communication and system reset are done in the second following call of the S-WdgM main function.

Note: For alive supervision, the detection is at the end of the next supervision reference cycle for alive supervision.


WdgMMainFunction
CP
Fault
WdgMMainFunction
CP not called within expected time interval
CP
WdgMMainFunction
time
WdgMTriggerWindowStart
WdgMSupervisionCycle
WdgMTriggerConditionValue
S-WdgM fault detection time
S-WdgM fault reaction time
S-WdgM Stack maximum reaction time
RESET

CP … Checkpoint with Alive monitoring
- In the picture, `WdgMSupervisionReferenceCycle = WdgMSupervisionCycle`
- The Watchdog is triggered inside the `WdgM_MainFunction()`.
- The green line represents the time window when the Watchdog can be triggered.
- The ‘S-WdgM Fault detection time’ is equal to ISO26262 ‘Diagnostic test interval’
- The ‘**ault reaction time** is the **S-WdgM Fault reaction time + the S-Wdg Fault reaction time**.

Fig. 15: The S-WdgM Stack maximum reaction time

2.2.9 Reset Path and Safe State

The **safe state** is entered as a result of an **MCU reset**. The S-WdgM builds its functionality on a **reliable** and **robust reset path**. The S-WdgM **default reset path** uses the **Watchdog Device** itself through the S-WdgIF. The Watchdog Device can be either an external chip or an MCU-internal controller. The system integrator can additionally set a secondary path by adding the parameter `WDGM_SECOND_RESET_PATH = STD_ON`. The **secondary reset path** is used when the **Safe Watchdog Interface** returns an **error response**. This error response can be caused by communication errors to the external Watchdog device.

Figure 16 shows the **primary** and **secondary** reset path.

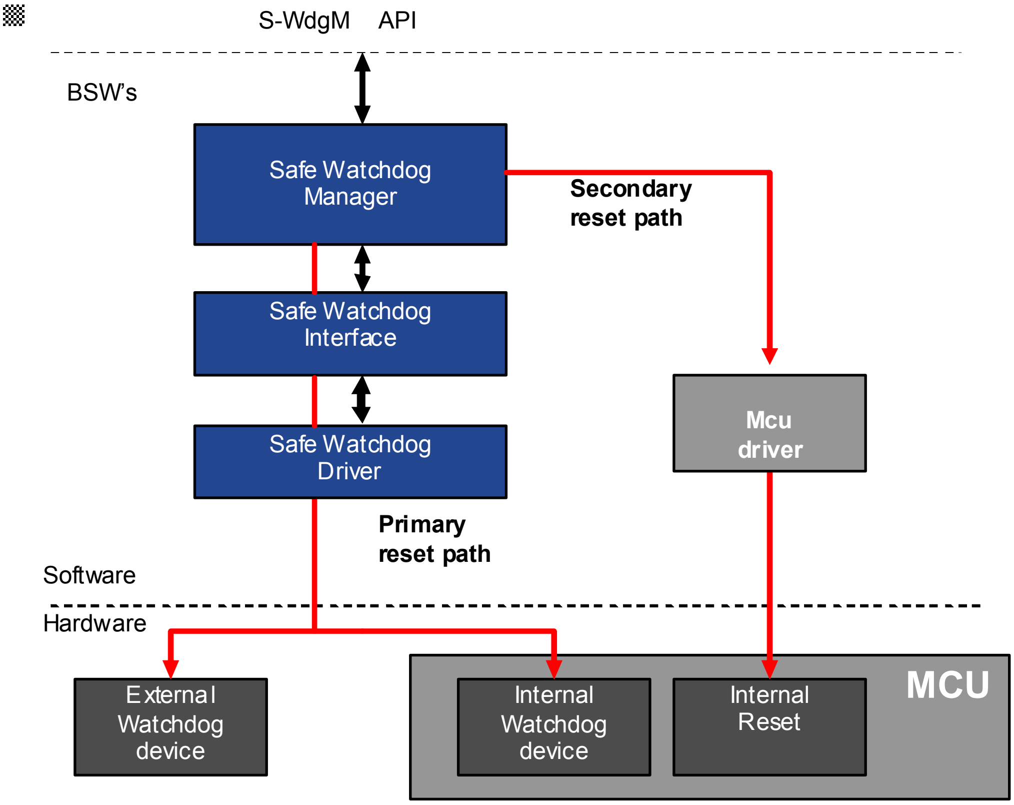


Fig. 16: Primary and secondary reset path of the S-WdgM

The S-WdgM uses the **primary reset path** for a **regular Watchdog-initiated reset** and also for an **immediate MCU reset**. The primary reset path is the **preferred** path, because it is part of the S-WdgM software and thus safe. The MCU driver with the AUTOSAR function `Appl_Mcu_PerformReset()` ↳89 must guarantee freedom from interference.

The **secondary reset path** is optional. It is used when the **primary** reset path signals a **fault**.

The S-WdgM **safe state** is the **MCU reset state**.

Note: The S-WdgM safe state is not necessarily the system safe state.

The S-WdgM can invoke the safe state in two ways:

- MCU reset after watchdog timeout by discontinuing watchdog triggering.
- Immediate MCU reset by an immediate watchdog reset. The immediate reset can be configured. See parameter `WDGM_IMMEDIATE_RESET` 39 in Section S-WdgM Global Preprocessor Settings 38.

2.2.10 S-WdgM Local Entity State

Every supervised entity has a **local state** that expresses the occurrence of detected violations:

State `OK` No violation has been detected

State `FAILED` A violation has been detected, the reset is pending within a delay time (maybe 0 ticks) and the violation repeats.

State `EXPIRED` A violation has repeated throughout the delay time. A reset is inevitable.

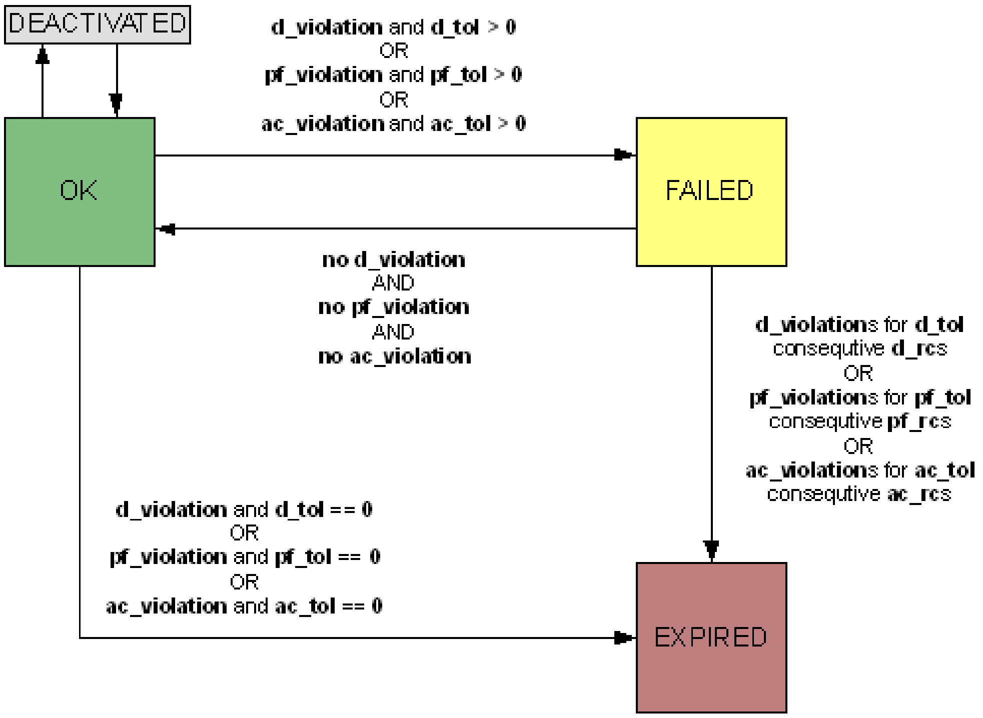
AUTOSAR allows configuring a tolerance delay after an alive counter violation has been detected. See [1] for detailed information. AUTOSAR does not allow configuring such tolerances for program flow and deadline violations. The S-WdgM allows configuring such tolerances for all three monitoring features described below:

- Once a violation has been detected, the S-WdgM changes its state from `OK` to `FAILED` and starts a so-called **tolerance time**, which is configured as follows:

The tolerance time is the **supervision reference cycle** (according to the monitoring feature) **multiplied** by a supervision reference cycle **tolerance value**.

- As long as the violation repeats within the tolerance time at least every supervision reference cycle, the S-WdgM stays in the state `FAILED`.
- If the violation does not occur in a supervision reference cycle within the tolerance delay, the S-WdgM returns to the state `OK` as if no violation had happened. Only the status change is logged.
- If the violation has repeated to the end of the tolerance time, the S-WdgM enters the state `EXPIRED`.

Figure 17 shows the state changes in dependence of the configured reference cycles and reference cycle tolerances.



d_tol	Deadline Reference Cycle Tolerance
d_rc	Deadline Reference Cycle
d_violation	at least one Deadline violation within the last **d_rc**
pf_tol	Program FlowReference Cycle Tolerance
pf_rc	Program FlowReference Cycle
pf_violation	at least one Program Flow violation within the last **pf_rc**
ac_tol	Alive Counter Reference Cycle Tolerance
ac_rc	Alive Counter Reference Cycle
ac_violation	at least one Alive Counter violation within the last **ac_rc**
Reference Cycle	reference cycle length given in number of WdgM cycles

Fig. 17: Modified state machine

Note: The AUTOSAR implementation can be simulated for deadline and program flow violations with

```
reference cycle = reference cycle tolerance = 0.
```

The exact names of the configuration fields for the tolerance delay are:

Monitoring	Reference Cycle	Reference Cycle Tolerance
Alive Supervision	WdgMSupervisionReferenceCycle[66]	WdgMFailedSupervisionRefCycleTol[57]
Program Flow Monitoring	WdgMProgramFlowReferenceCycle[60]	WdgMFailedProgramFlowRefCycleTol[59]
Deadline Monitoring	WdgMDeadlineReferenceCycle[59]	WdgMFailedDeadlineRefCycleTol[58]

Note:

- `WdgMProgramFlowReferenceCycle`[60] and `WdgMFailedProgramFlowRefCycleTol`[59] must both be **0** or **unequal to 0**.
- `WdgMDeadlineReferenceCycle`[59] and `WdgMFailedDeadlineRefCycleTol`[58] must both be **0** or **unequal to 0**.

2.2.11 S-WdgM Global State

The local states are periodically summarized in an **S-WdgM global state**. If all supervised entities have the state `OK`, then the global state is `OK`. When at least one supervised entity changes to the state `FAILED`, then the global state becomes `FAILED`. When at least one supervised entity changes to the state `EXPIRED`, the global state becomes `EXPIRED`. Once the global state is `EXPIRED`, the S-WdgM continues the delay until it enters the state `STOPPED`. This is when the S-WdgM stops triggering the Watchdog (or resets it). The **delay** is the **supervision cycle multiplied** by the configurable **expired supervision cycle tolerance** (parameter `WdgMExpiredSupervisionCycleTol`[53]).

Once in the state `STOPPED`, the S-WdgM brings the system to the safe state by performing a system reset through the **S-WdgIf module** and, thus, through the watchdog(s) in the system. If the preprocessor option `WDGM_SECOND_RESET_PATH`[45] is set to `STD_ON` and the S-WdgIf reports a failure, then the system goes into a safe state through the MCU module (see Section S-WdgM Global Preprocessor Settings[38])
.

2.3 Integration in AUTOSAR 3.1 and 4.0 Environments

The S-WdgM implements functionality described in **AUTOSAR 4.0r1**. However, the S-WdgM can be integrated in **AUTOSAR 3.1** and **AUTOSAR 4.0** environments. To this end, a special **preprocessor switch** is automatically generated by the **configuration generator**. That preprocessor switch cannot be altered manually. This is `WDGM_AUTOSAR_4_x (STD_ON / STD_OFF)`, which is placed in the generated file `WdgM_Cfg_Features.h`. The value of the preprocessor switch is determined by the configuration generator according to the provided **ECUC file**, more specifically according to the **XML default name space** of the ECUC file (attribute **`xmlns`**).

- For **AUTOSAR 3.1**: `WDGM_AUTOSAR_4_x` is generated to `STD_OFF`, which prepares the embedded code for a compilation in an AUTOSAR 3.1 environment. If the AUTOSAR version is not 3.1, but any other 3.x, the configuration generator additionally outputs a warning during this process.
- For **AUTOSAR 4.0**: `WDGM_AUTOSAR_4_x` is generated to `STD_ON`, which prepares the embedded code for a compilation in an AUTOSAR 4.0 environment. If the AUTOSAR version is not 4.0, but any other 4.x, the configuration generator additionally outputs a warning during this process.
- For any **other AUTOSAR version** (smaller than 3 or greater than 4), the configuration generator generates no code and exits with an error message.

Note: The integration of the S-WdgM in an **AUTOSAR 3.1 environment** must be differentiated from the **AUTOSAR 3.1 compatibility mode** described in this document. The integration into an AUTOSAR environment refers only to the software environment in which the S-WdgM interacts, whereas the AUTOSAR 3.1 **compatibility mode** is a **special operation mode** of the module itself selected at pre-compile time. In this special mode, the functionality is reduced to the functionality described by the AUTOSAR 3.1. For more information refer to **AUTOSAR version 3.1 r1** [7] 128.

2.4 Deviations from the AUTOSAR 4.0 r1 Watchdog Manager

The S-WdgM is compatible with the **AUTOSAR 4.0 r1 Watchdog Manager**, but not fully compliant. This has the following reasons:

- The AUTOSAR specification does not define functionality comprehensively and precisely enough for implementation (e.g., global transitions).
- The AUTOSAR specification does not contain certain functionality (e.g., program flow, deadline monitoring recovering).
- The AUTOSAR specification defines an approach that is very complex to be handled by the user or consumes too much run time (S-WdgM mode switching).
- The AUTOSAR specification does not fully consider safety requirements (e.g., windowed Watchdog Trigger).

Below you can find the **deviations** from the AUTOSAR 4.0 r1 Watchdog Manager **in detail**:

2.4.1 Entities, Checkpoints and Transitions

- For periodical watchdog triggering at least one supervised entity and one checkpoint should be defined.
- In contrast to AUTOSAR, local activity flags of the supervised entities are set back to

FALSE every time an end checkpoint of this supervised entity is reached. Analogously, the global activity flag is set back to FALSE as soon as a global end checkpoint is reached.

- Local initial checkpoints cannot have incoming local transitions, but they can have incoming global transitions.
- Local end checkpoints cannot have outgoing local transitions, but they can have outgoing global transitions.
- If global transitions are used, then there must be exactly one global initial checkpoint.
- The global initial checkpoint should be called before any other global checkpoint is invoked.
- If a non-initial checkpoint of an supervised entity is reached and this supervised entity is not active, then this is considered to be a program flow violation in this supervised entity.
- If a checkpoint is the source for a local and a global transition, then only one of the two transitions can occur. The other one is considered a program flow violation. This is because the program flow cannot split into 2 paths. If, for example, a new task is started from a **CP1** (global transition to **CPnew**) and the original task continues (local transition to **CP2**), then the sequence following the sequences of checkpoint hits is not allowed:

  ○ **CP->CPnew->CP2** and

  ○ **CP->CP2->CPnew**.
- If a local initial checkpoint is the destination checkpoint for a global transition, then the checkpoint must be hit by following the global transition. There is a dilemma, though: If several supervised entities form a cycle of transitions, with each supervised entity entered via a global transition from the previous supervised entity, then there is no way to start the cycle, because no local initial checkpoint is allowed to be hit in a way other than via the global transition. The solution is an exception in the S-WdgM: A local initial checkpoint can be hit, not coming through the global transition, if it is also the global initial checkpoint.
- As in AUTOSAR, the S-WdgM needs a time source in order to measure transition deadlines. Whereas AUTOSAR does not define the source for ticks, the S-WdgM allows the user to choose between three Tick sources:
  ○ Internal software source,

  ○ Internal hardware source,

  ○ External tick source

  For details see Section Deadline Measurement and Tick Counter 100 and the description of parameter WdgMTimebaseSource 44 in Section S-WdgM Global Preprocessor Settings 38.
- The checkpoint and entity identifiers are zero-based and increase the list of integer numbers without gaps.

- Deadline monitoring is bound to program flow. Only if program flow transitions are configured, it is possible to configure transition deadlines.
- The local/global end checkpoint does not need to be defined.
- Currently only one checkpoint with an alive counter is supported per supervised entity. This is recommended in the AUTOSAR 4.0 r1 Watchdog Manager specification, since the functionality is not consistently described.

2.4.2 Tolerances

- The S-WdgM allows **tolerance delay** for all three monitoring features. In AUTOSAR, this is restricted to alive supervision. Tolerance delay allows recovering from program flow and deadline violations as well as from alive counter violations.
- The interpretation of the **AUTOSAR** parameter `WdgMExpiredSupervisionCycleTol` 53 implements a delay of **`(WdgMExpiredSupervisionCycleTol + 2)` supervision cycles**. The **S-WdgM** implements a delay of **`WdgMExpiredSupervisionCycleTol` supervision cycles**. This allows configuring no delay, with the tolerance value set to **0**.

2.4.3 Watchdog and Reset

- The AUTOSAR Watchdog Manager supports several watchdog drivers and several watchdog devices per watchdog driver. However, the TTTech S-WdgM Stack supports only one watchdog driver and only one watchdog device per watchdog driver.
- For safety reasons, the S-WdgM uses the primary watchdog reset as an immediate reset (`WDGM_IMMEDIATE_RESET = STD_ON`) . In contrast, the AUTOSAR Watchdog Manager uses the external function `Appl_Mcu_PerformReset()`.
- The S-WdgM does not support a **partition reset** with `BswM_WdgM_RequestPartitionReset()`.

2.4.4 API

- The S-WdgM function `WdgM_SetMode()` switches the **trigger mode** only. This relates to the fields
  - `WdgMTriggerConditionValue` 56
  - `WdgMTriggerWindowStart` 56
  - `WdgMWatchdogMode` 55.

  It does not change the set of supervised entities. This can be simulated by activating and deactivating different sets of supervised entities for different modes. **Note:** Full support of the function is too time expensive at runtime and too complex (not safe) to implement and to configure.

- For safety and complexity reasons, the function `WdgM_DeInit()` is not implemented.
- The S-WdgM provides the functions `WdgM_DeactivateSupervisionEntity()` 81 and `WdgM_ActivateSupervisionEntity()` 82 for deactivating and activating of the SE. These functions are not AUTOSAR 4.0 r1 compatible.
- The S-WdgM uses only direct callback notification for a local and global state change. The RTE notification is not implemented.
- Due to implementation complexity and verification difficulty, the S-WdgM does not support RTE Mode Ports.
- The S-WdgM checks the configuration independently of the `WdgMDevErrorDetect` 38 parameter. This parameter enables/disables the DET calls only.

The ECU Description Configuration constraints are described in Section Assumptions/ Constraints 72.

2.5 Configuration Parameters for the S-WdgM

This Section contains a brief description of the configuration parameters for the S-WdgM, sorted according to their functionality. The path to each parameter or option is the exact ECU description file path. The parameters are placed inside the ECU description file. The S-WdgM Configuration Generator[102] uses the parameters to generate **configuration structures**.

The list includes functions defined in AUTOSAR 4.0 r1 and functions added by TTTech. For AUTOSAR 3.1 functions and a comparison of AUTOSAR 4.0 r1 and AUTOSAR 3.1 functions, see Section AUTOSAR 3.1 Compatibility[90].

2.5.1 S-WdgM Global Preprocessor Settings

Parameter Name	`WdgMDevErrorDetect`
Parameter Name **(Embedded Code)**	`WDGM_DEV_ERROR_DETECT`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch enables/disables development error detection and reporting. This parameter must be used to remove unneeded code segments regarding DET features. `true`: Development error detection is enabled. `false`: Development error detection is disabled.

Parameter Name	`WdgMDemReport`
Parameter Name **(Embedded Code)**	`WDGM_DEM_REPORT`

Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch enables/disables calls to DEM in case of production error detection. `true`: DEM calls enabled in case of production errors. `false`: DEM calls disabled in case of production errors.

Parameter Name	`WdgMImmediateReset`
Parameter Name (Embedded Code)	`WDGM_IMMEDIATE_RESET`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch enables/disables the **immediate watchdog reset** feature in case of *alive*, *deadline* or *program flow* fault. When it is enabled and the S-WdgM recognizes a fault (i.e., the S-WdgM global state changes to `WDGM_GLOBAL_STATUS_STOPPED`), then the S-WdgM does not wait for the watchdog device timeout, but invokes the reset immediately. The parameter can be configured to perform an MCU reset if the immediate reset fails. **Note:** Not all hardware platforms can invoke an immediate reset.

	true: Perform an immediate watchdog reset. false: Discontinue watchdog trigger and wait for watchdog timeout.

Parameter Name	WdgMOffModeEnabled
Parameter Name (Embedded Code)	WDGM_OFF_MODE_ENABLED
Path	WdgM/WdgMGeneral/
Group	Preprocessor
Type	Boolean
Range	false/true
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch enables/disables the selection of WDGIF_MODE_OFF for the watchdog mode. When enabled, the watchdog device can be deactivated. **Note:** On the same hardware platform, the watchdog cannot be deactivated once it has been activated. true: WDGIF_MODE_OFF is allowed. false: WDGIF_MODE_OFF is disallowed.

Parameter Name	WdgMVersionInfoApi
Parameter Name (Embedded Code)	WDGM_VERSION_INFO_API
Path	WdgM/WdgMGeneral/
Group	Preprocessor

Type	Boolean
Range	`false/true`
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch enables/disables the API function `WdgM_GetVersionInfo()` 92. **Note:** `WdgM_GetVersionInfo()` is a macro. `true`: Version API is enabled. `false`: Version API is disabled.

Parameter Name	`WdgMDefensiveBehavior`
Parameter Name (Embedded Code)	`WDGM_DEFENSIVE_BEHAVIOR`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch enables/disables the defensive behavior of the Watchdog Manager module. ▪ `WdgM_SetMode()` 76 checks whether the caller is authorized. ▪ `WdgM_MainFunction()` 85 checks if the S-WdgM has been initialized.

Parameter Name	`WdgMUseRte`
Parameter Name **(Embedded Code)**	`WDGM_USE_RTE`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	TTTech
Description	This preprocessor switch instructs the S-WdgM to use the defines and typedefs generated by the RTE. The RTE-generated defines and typedefs save S-WdgM configuration RAM. **Note:** Section S-WdgM Type Definitions[73] covers the types and defines that can be imported from the RTE. `true`: The S-WdgM uses the RTE-generated defines and typedefs. `false`: The S-WdgM uses its own defines and typedefs.

Parameter Name	`WdgMDemSupervisionReport`
Parameter Name **(Embedded Code)**	`WDGM_DEM_SUPERVISION_REPORT`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`

Compatibility	AUTOSAR 4.0 r1 (renamed from WdgMDemAliveSupervisionReport)
Description	This preprocessor switch enables/disables the call to DEM if the S-WdgM has reached the state WDGM_GLOBAL_STATE_STOPPED. true: The DEM call is performed. false: The DEM call is not performed.

Parameter Name	WdgMUseOsSuspendInterrupt
Parameter Name **(Embedded Code)**	WDGM_USE_OS_SUSPEND_INTERRUPT
Path	WdgM/WdgMGeneral/
Group	Preprocessor
Type	Boolean
Range	false/true
Compatibility	AUTOSAR 4.0 r1
Description	This preprocessor switch controls how interrupts are suspended and resumed within the S-WdgM. true: For **AUTOSAR 3.1** (WDGM_AUTOSAR_4_x is STD_OFF), the S-WdgM uses - function SchM_Enter_WdgM() to suspend interrupts, - function SchM_Exit_WdgM() to resume interrupts. For **AUTOSAR 4.0** (WDGM_AUTOSAR_4_x is STD_ON), the S-WdgM uses - function SchM_Enter_WdgM_WDGM_EXCLUSIVE_ARE _0() to suspend interrupts, - function SchM_Exit_WdgM_WDGM_EXCLUSIVE_AREA

	`0()` to resume interrupts. `false:` The user must define - function `GlobalSuspendInterrupts()` to suspend interrupts, - function `GlobalRestoreInterrupts()` to resume interrupts.

Parameter Name	`WdgMTimebaseSource`
Parameter Name **(Embedded Code)**	`WDGM_TIMEBASE_SOURCE`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	integer
Range	▪ `WDGM_EXTERNAL_TICK` ▪ `WDGM_INTERNAL_SOFTWARE_TICK` ▪ `WDGM_INTERNAL_HARDWARE_TICK`
Compatibility	TTTech
Description	This preprocessor switch defines the source for the S-WdgM *Tick*. **Note:** ▪ The precision of the transition deadline measurement is based on this *Tick*. ▪ When the deadline measurement is not used, the S-WdgM *Tick* counter is internally not used, and it need not be incremented. In this case, to save run-time resources, the parameter `WdgMTimebaseSource` should be set to `WdgMInternalSoftwareTick`, which is the default value. See also parameter `WdgMTicksPerSecond`. The parameters: ▪ `WDGM_EXTERNAL_TICK:`

	An external clock source (through the API function `WdgM_UpdateTickCount()`[87]). The S-WdgM tick counter is incremented every time this function is called by the system. ▪ `WDGM_INTERNAL_SOFTWARE_TICK:` The S-WdgM *Tick* Counter is incremented every time `WdgM_MainFunction()`[85] is called. ▪ `WDGM_INTERNAL_HARDWARE_TICK:` The Tick source is the MCU hardware counter. The frequency of the MCU hardware counter is given by the parameter `WdgMTicksPerSecond`. The tick is queried by the S-WdgM through the S-WdgIf API. **Note:** Not all hardware platforms support this feature. For details, refer to the S-Wdg Driver documentation.

Parameter Name	`WdgMSecondResetPath`
Parameter Name **(Embedded Code)**	`WDGM_SECOND_RESET_PATH`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	TTTech
Description	This preprocessor switch allows an MCU reset if a WD command (trigger or reset) fails. This second reset path is performed by calling `Appl_Mcu_PerformReset()`. **Note:** `Appl_Mcu_PerformReset()` itself calls `Mcu_PerformReset()`, which triggers the reset. `true`: The MCU is reset with `Appl_Mcu_PerformReset()` when the primary reset path signals an error.

	false: The MCU is not reset.

Parameter Name	WdgMTickOverrunCorrection
Parameter Name **(Embedded Code)**	WDGM_TICK_OVERRUN_CORRECTION
Path	WdgM/WdgMGeneral/
Group	Preprocessor
Type	Boolean
Range	false/true
Compatibility	TTTech
Description	This preprocessor switch enables/disables the 32-bit S-WdgM *Tick* Counter overflow detection and correction. ▪ true: The *Tick* counter overflow is corrected. ▪ false: The *Tick* counter overflow is not corrected. **Note:** Depending on the frequency with which the *Tick* Counter is incremented, the counter can overflow or not. See parameter WdgMTimebaseSource 44 for additional information. The Tick Counter overflow detection and correction is only used when WDGM_TIMEBASE_SOURCE = WDGM_EXTERNAL_TICK. If not set to true, the check of the tick counter for jumps and jitter may be incorrect. The parameter must be set to true when the external Tick source is used and the Tick counter (32bit) can overflow. **Example**: The tick counter is incremented every millisecond. Then the overflow happens after 49 days.

Parameter Name	`WdgMEntityDeactivationEnabled`
Parameter Name (Embedded Code)	`WDGM_ENTITY_DEACTIVATION_ENABLED`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean
Range	`false/true`
Compatibility	TTTech
Description	This preprocessor switch enables entity deactivation. This functionality is not specified in AUTOSAR 4.0 r1 and can violate system safety (see the *Safe Watchdog Manager Safety Manual* [5]→128, parts `WdgM_DeactivateSupervisionEntity()`→81 and `WdgM_ActivateSupervisionEntity()`→82). See also parameter `WdgMEnableEntityDeactivation`→61. ▪ `true`: An entity can be deactivated. ▪ `false`: An entity cannot be deactivated. The default value is `false`.

Parameter Name	`WdgMStateChangeNotification`
Parameter Name (Embedded Code)	`WDGM_STATE_CHANGE_NOTIFICATION`
Path	`WdgM/WdgMGeneral/`
Group	Preprocessor
Type	Boolean

Range	false/true
Compatibility	TTTech
Description	This preprocessor switch enables *local* and *global state* change callback notifications. There are different callbacks for *local* and *global state* notifications. true: Any *local* or *global state* change invokes a callback. false: No callbacks are performed. See also the parameters WdgMGlobalStateChangeCbk 49 and WdgMLocalStateChangeCbk 63.

Parameter Name	WdgMCallerId
Path	WdgM/WdgMGeneral/WdgMCallerIds/
Group	General
Type	Integer
Range	0...65535
Compatibility	AUTOSAR 4.0 r1
Description	This parameter defines one valid CallerId for the callers that have permission to call the function WdgM_SetMode().

Parameter Name	WdgMFirstCycleAliveCounterReset
Parameter Name **(Embedded Code)**	WDGM_FIRSTCYCLE_ALIVECOUNTER_RESET
Path	WdgM/WdgMGeneral/
Group	General
Type	Boolean

Range	`false/true`
Compatibility	TTTech
Description	This parameter decides if the Alive counters are evaluated in the first supervision cycle. ▪ `true`: The Alive counters are not evaluated in the first supervision cycle ▪ `false`: The Alive counters are evaluated in the first supervision cycle

2.5.2 S-WdgM General Settings

Parameter Name	`WdgMGlobalStateChangeCbk`
Path	`WdgM/WdgMGeneral/`
Group	General
Type	Reference
Compatibility	TTTech
Description	This is the parameter for a callback function for notifying the system of the S-WdgM global state change. The S-WdgM has only one callback function for the global state. In a safety-relevant environment, the callback function can cause safety degradation. For details, refer to the *Safe Watchdog Manager Safety Manual* [5] 128.

Parameter Name	`WdgMGlobalMemoryAppTaskRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/`
Group	General
Type	Reference
Multiplicity	`0, 1`
Compatibility	TTTech

Description	This is the parameter for a reference to an OS application or task where the S-WdgM is running. **Note:** When OS SC3 (OS with memory protection) is used, the global variables of the S-WdgM should be placed in the same memory segment where the S-WdgM context is running. **Example:** The application name is incorporated into the corresponding `MemMap` defines in the file `WdgM_MemMap.h` in an AUTOSAR 3.1 environment or `WdgM_OSMemMap.h` in an AUTOSAR 4.0 environment.

Parameter Name	`WdgMModeId`
Path	`WdgM/WdgMConfigSet/WdgMMode/`
Group	General
Type	Integer
Range	`0...255`
Compatibility	AUTOSAR 4.0 r1 / TTTech
Description	This is the parameter for the S-WdgM mode. The S-WdgM, in contrast to the AUTOSAR WdgM, uses only one mode. This parameter is kept for compatibility reasons only, and it is not used by the S-WdgM.

Parameter Name	`WdgMInitialTriggerModeId`
Path	`WdgM/WdgMConfigSet/WdgMMode/`
Group	General
Type	Integer
Range	`0...255`

Compatibility	TTTech
Description	This is the parameter for the S-WdgM initial trigger mode. The S-WdgM trigger mode is a restricted version of the AUTOSAR mode. It only sets the fields: ▪ `WdgMTriggerConditionValue` ▪ `WdgMTriggerWindowStart` ▪ `WdgMWatchdogMode` When more than one Watchdog device is used, then this parameter addresses the first Watchdog only. For details, refer to the function `WdgM_SetMode()`.

Parameter Name (ECU)	`WdgMTriggerModeId`
Path (ECU)	`WdgM/WdgMConfigSet/WdgMMode/WdgMTrigger/`
Group	Watchdog trigger
Type	Integer
Range	`0...254`
Compatibility	TTTech
Description	This parameter contains a unique identifier of the trigger mode.

Parameter Name	`WdgMTicksPerSecond`
Path	`WdgM/WdgMConfigSet/WdgMMode/`
Group	General
Type	Float
Unit	`Hz`

Compatibility	TTTech
Description	This parameter defines the number of S-WdgM *Ticks* per second. It is the rate by which the S-WdgM *Tick* Counter is incremented. This parameter is used in two ways: 1. The system environment that periodically calls the function `WdgM_UpdateTickCount()` for deadline monitoring. See also parameter `WdgMTimebaseSource`. 2. The S-WdgM Configuration Generator that calculates min and max parameters for the transition deadlines. **Note:** • When the S-WdgM *Tick* source is `WDGM_INTERNAL_SOFTWARE_TICK`, then the following relation must be obeyed: `(1 / WdgMTicksPerSecond [Hz])` `= WdgMSupervisionCycle [s]` • For the *Tick* sources `WDGM_INTERNAL_HARDWARE_TICK` and `WDGM_EXTERNAL_TICK`, the following relation must be obeyed: `(1 / WdgMTicksPerSecond [Hz])` `<= WdgMSupervisionCycle [s]` • The parameter `WdgMTicksPerSecond` **must not** be **zero**.

Parameter Name	`WdgMSupervisionCycle`
Path	`WdgM/WdgMConfigSet/WdgMMode/`
Group	General
Type	Float
Range	`0 < WdgMSupervisionCycle`
Unit	`second`
Compatibility	AUTOSAR 4.0 r1

Description	This parameter defines the schedule period of the main function, `WdgM_MainFunction()`. It is the time period in which the S-WdgM performs cyclic supervision, and also the watchdog trigger period. The parameter is important for the system that calls the function `WdgM_MainFunction()`.

Parameter Name	`WdgMExpiredSupervisionCycleTol`
Path	`WdgM/WdgMConfigSet/WdgMMode/`
Group	General
Type	Integer
Range	`0...65535`
Compatibility	AUTOSAR 4.0 r1
Description	This parameter defines a further delay of the violation escalation to the Watchdog after the S-WdgM reached the status `WDGM_LOCAL_STATUS_EXPIRED` (in numbers of supervision cycles).

Parameter Name	`WdgMGlobalCheckpointFinalRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/` `WdgMProgramFlowSupervision/`
Group	General
Type	Reference
Multiplicity	`0...65535`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the final global checkpoint. **Note:** There might be no, one or several global end

	checkpoints.

Parameter Name	`WdgMGlobalCheckpointInitialRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/` `WdgMProgramFlowSupervision/`
Group	General
Type	Reference
Multiplicity	`0, 1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the global initial checkpoint. **Note:** If global transitions are defined, then exactly one global initial checkpoint must be defined.

Parameter Name	`WdgMWatchdogName`
Path	`WdgM/WdgMGeneral/WdgMWatchdog/`
Group	Watchdog device
Type	String
Range	N/A
Compatibility	AUTOSAR 4.0 r1
Description	This parameter is a symbolic name of the Watchdog. It is used as a comment only.

Parameter Name	`WdgIfDeviceRef`
Path	`WdgM/WdgMGeneral/WdgMWatchdog/`
Group	Watchdog device
Type	Reference
Multiplicity	`1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to a device container (`WdgIfDevice`) of the S-WdgIf. This container contains data and a reference that represents the connection of the S-WdgM to the Watchdog device through the S-WdgIf.

Parameter Name	`WdgMWatchdogMode`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMTrigger/`
Group	Watchdog trigger
Type	Enumeration
Range	▪ `WDGIF_FAST_MODE` ▪ `WDGIF_OFF_MODE` ▪ `WDGIF_SLOW_MODE`
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the watchdog mode for a referenced watchdog in the S-WdgM. Implementation type: `WdgIf_ModeType`. **Note:** Not all hardware platforms support all watchdog modes. For details, see the User Manual of the respective S-Wdg Driver. **Note**: Do not confuse this parameter with the S-WdgM

	Trigger Mode (WdgMModeID 50).

Parameter Name	WdgMTriggerConditionValue
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMTrigger/
Group	Watchdog trigger
Type	Integer
Range	1...65535
Unit	ms
Compatibility	AUTOSAR 4.0 r1
Description	This parameter defines the latest possible time where the next watchdog trigger is accepted (window end). **Note:** Not all hardware platforms allow changing this parameter during runtime. For details, see the User Manual of the respective S-Wdg Driver.

Parameter Name	WdgMTriggerWindowStart
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMTrigger/
Group	Watchdog trigger
Type	Integer
Range	0...65535
Unit	ms
Compatibility	TTTech
Description	This parameter defines the earliest time after which the next watchdog trigger is accepted (window start).

	Note: Not all hardware platforms allow changing this parameter during runtime. On some platforms, this parameter is not avaliable or set to zero. For details, see the User Manual of the respective S-Wdg Driver.

Parameter Name	`WdgMTriggerWatchdogRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMTrigger/`
Group	Watchdog trigger
Type	Reference
Multiplicity	`0...255`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the configured watchdog.

2.5.3 S-WdgM Supervised Entity Options

Parameter Name	`WdgMFailedSupervisionRefCycleTol`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/`
Group	Supervised entity
Type	Integer
Range	`0...65534`
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the acceptable number of failed alive indications for this supervised entity in a row (i.e., at least one violation per supervision reference cycle in a row). **Note:** This parameter should be set to **0** if no alive counter is configured for this supervised entity, because nothing can

	be tolerated. If there is an alive counter in this supervised entity, then the parameter can be **0** (no alive counter violations tolerated) or **positive**.

Parameter Name	`WdgMSupervisedEntityInitialMode`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/`
Group	Supervised entity
Type	Enumeration
Range	▪ `WDGM_LOCAL_STATUS_DEACTIVATED,` ▪ `WDGM_LOCAL_STATUS_OK,` ▪ `WDGM_LOCAL_STATUS_FAILED`
Compatibility	TTTech
Description	This is the initial local monitoring status of the supervised entity.

Parameter Name	`WdgMFailedDeadlineRefCycleTol`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/`
Group	Supervised entity
Type	Integer
Range	`0...65534`
Compatibility	TTTech
Description	This parameter contains the acceptable number of violated deadlines for this supervised entity in a row (i.e., at least one violation per `WdgMDeadlineReferenceCycle` in a row). **Note:** If a positive tolerance for deadline violations is entered, then the user must enter a positive reference cycle

	for the violations (WdgMDeadlineReferenceCycle), because the tolerance is defined in terms of reference cycles. The tolerance can also be **0**. In this case a positive reference cycle would make no sense, because there is no reference cycle if no violations are tolerated.

Parameter Name	WdgMDeadlineReferenceCycle
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/
Group	Supervised entity
Type	Integer
Range	0...65535
Compatibility	TTTech
Description	This parameter contains the number of supervision cycles that define a cycle for the *deadline monitoring* of this supervised entity. **Note:** If the deadline reference cycle tolerance (WdgMFailedDeadlineRefCycleTol) is set to **0**, then this parameter must be **0** as well. This is because the first detected violation would cause the supervised entity to change its status to EXPIRED and then no reference cycle could exist. If the deadline reference cycle tolerance is positive, then this parameter must be positive as well, because the tolerance is defined as a number of reference cycles which cannot be of zero duration.

Parameter Name	WdgMFailedProgramFlowRefCycleTol
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/
Group	Supervised entity
Type	Integer
Range	0...65534

Compatibility	TTTech
Description	This parameter contains the acceptable number of program flow violations for this supervised entity in a row (i.e., at least one violation per `WdgMProgramFlowReferenceCycle` in a row). **Note:** If a positive tolerance for program flow violations is entered, then the user must enter a positive reference cycle for the violations (`WdgMProgramFlowReferenceCycle`), because the tolerance is defined in terms of reference cycles. The tolerance can also be **0**. In this case a positive reference cycle would make no sense, because there is no reference cycle if no violations are tolerated.

Parameter Name	`WdgMProgramFlowReferenceCycle`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/`
Group	Supervised entity
Type	Integer
Range	`0...65535`
Compatibility	TTTech
Description	This parameter contains the number of supervision cycles that define a cycle for the *program flow monitoring* of this supervised entity. **Note:** If the program flow reference cycle tolerance (`WdgMFailedProgramFlowRefCycleTol`) is set to **0**, then this parameter must be **0** as well. This is because the first detected violation would cause the supervised entity to change its status to `EXPIRED` and then no reference cycle could exist. If the deadline reference cycle tolerance is positive, then this parameter must be positive as well, because tolerance is defined as a number of reference cycles which cannot be of zero duration.

Parameter Name	`WdgMLocalStatusSupervisedEntityRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams/`
Group	Supervised entity
Type	Reference
Multiplicity	`1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the supervised entity for which the parameters of this container are set.

Parameter Name	`WdgMSupervisedEntityId`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/`
Group	Supervised entity
Type	Integer
Range	`0...65534`
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the identifier of the supervised entity for which the parameters of this container are set.

Parameter Name	`WdgMEnableEntityDeactivation`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/`
Group	Supervised entity
Type	Boolean

Range	`false/true`
Compatibility	TTTech
Description	This parameter enables the deactivation and activation of this *supervised entity*. See also the preprocessor switch `WdgMEntityDeactivationEnabled`47. This functionality is not specified in AUTOSAR 4.0 r1 and can violate system safety (see the *Safe Watchdog Manager Safety Manual* [5]128, parts `WdgM_DeactivateSupervisionEntity()` and `WdgM_ActivateSupervisionEntity()`). ▪ `true`: *Supervised entity* deactivation and activation is enabled. - For activation, function `WdgM_ActivateSupervisionEntity()`82 must be used. - For deactivation, function `WdgM_DeactivateSupervisionEntity()`81 must be used ▪ `false`: Entity deactivation and activation for this supervised entity is disabled.

Parameter Name	`WdgMSupportedAutosarAPI`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/`
Group	Supervised entity
Type	Enumeration
Range	`API_4_0` `API_3_1`
Compatibility	TTTech
Description	This parameter defines the S-WdgM API compatibility. ▪ `API_4_0`: The AUTOSAR 4.0 r1 API is selected.

	▪ API_3_1: The AUTOSAR 3.1 API is selected. The system can be either AUTOSAR4.0 r1 or AUTOSAR 3.1. Mixed variants are not allowed. When one supervised entity in a system is AUTOSAR 3.1 then all the other supervised entities must be AUTOSAR 3.1 as well. For details, refer to Section AUTOSAR 3.1 Compatibility 90.

Parameter Name	WdgMLocalStateChangeCbk
Path	WdgM/WdgMGeneral/WdgMSupervisedEntity/
Group	Supervised entity
Type	Function name
Multiplicity	0, 1
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a callback function used to inform about a local state change of a supervised entity. The S-WdgM has one callback function for every supervised entity. **Note:** In a safety-relevant environment, the callback function can cause safety degradation. For details, see the *Safe Watchdog Manager Safety Manual* [5] 128.

Parameter Name	WdgMLocalCheckpointFinalRef
Path	WdgM/WdgMGeneral/WdgMSupervisedEntity/
Group	Supervised entity
Type	Reference
Multiplicity	0...65535
Compatibility	AUTOSAR 4.0 r1

Description	This is the reference to an end checkpoint for this supervised entity.

Parameter Name	`WdgMLocalCheckpointInitialRef`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/`
Group	Supervised entity
Type	Reference
Multiplicity	`1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the reference to the initial checkpoint for this supervised entity.

Parameter Name	`WdgMAppTaskRef`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/`
Group	Supervised entity
Type	Reference
Multiplicity	`0, 1`
Compatibility	TTTech
Description	This is the reference to an OS application (task) to which this supervised entity belongs. In case of OS SC3, the local data of the supervised entity must be placed in the same memory segment as the application (task) of which this supervised entity is a part. The S-WdgM Configuration Generator▷102 enables memory mapping of the supervised entity local data so that it can be put into the memory segment of the referred task or application (task) using memory mapping.

2.5.4 S-WdgM Checkpoint Options

Parameter Name	`WdgMCheckpointId`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/WdgMCheckpoint/`
Group	Checkpoint
Type	Integer
Range	`0...65534`
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the identifier of the checkpoint that is unique over the supervised entity.

2.5.5 Alive Counter Options

Parameter Name	`WdgMExpectedAliveIndications`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMAliveSupervision/`
Group	Alive counter
Type	Integer
Range	`0...65535`
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the number of expected *alive* indications within a supervision reference cycle, according to the corresponding supervised entity.

Parameter Name	WdgMMaxMargin
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMAliveSupervision/
Group	Alive counter
Type	Integer
Range	0...65535
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the number of *alive* indications that are acceptable in addition to the expected indications (WdgMExpectedAliveIndications) within the corresponding supervision reference cycle.

Parameter Name	WdgMMinMargin
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMAliveSupervision/
Group	Alive counter
Type	Integer
Range	0...65535
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the number of *alive* indications that are acceptable to be missing from the expected indications (WdgMExpectedAliveIndications) within the corresponding supervision reference cycle.

Parameter Name	WdgMSupervisionReferenceCycle
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMAliveSupervision/

Group	Alive counter
Type	Integer
Range	1...65535
Compatibility	AUTOSAR 4.0 r1
Description	This parameter defines the supervision reference cycle length as a number of supervision cycles (WdgMSupervisionCycle).

Parameter Name	WdgMAliveSupervisionCheckpointRef
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMAliveSupervision/
Group	Alive counter
Type	Reference
Multiplicity	1
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the checkpoint for which this alive supervision is configured.

2.5.6 S-WdgM Local Transition Options

Parameter Name	WdgMLocalTransitionDestRef
Path	WdgM/WdgMGeneral/WdgMSupervisedEntity/ WdgMLocalTransition/
Group	Local transition
Type	Reference
Multiplicity	1

Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the destination checkpoint of a local transition within this *supervised entity.*

Parameter Name	`WdgMLocalTransitionSourceRef`
Path	`WdgM/WdgMGeneral/WdgMSupervisedEntity/` `WdgMLocalTransition/`
Group	Local transition
Type	Reference
Multiplicity	`1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the source checkpoint of a local transition within this *supervised entity*.

2.5.7 S-WdgM Global Transition Options

Parameter Name	`WdgMGlobalTransitionDestRef`

Path	`WdgM/WdgMConfigSet/WdgMMode/`
	`WdgMProgramFlowSupervision`
	`/WdgMGlobalTransition/`

Group	Global transition
Type	Reference
Multiplicity	`1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the destination checkpoint of a global transition.

Parameter Name	WdgMGlobalTransitionSourceRef
Path	WdgM/WdgMConfigSet/WdgMMode/ WdgMProgramFlowSupervision /WdgMGlobalTransition/
Group	Global transition
Type	Reference
Multiplicity	1
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the source checkpoint of a global transition.

2.5.8 S-WdgM Local and Global Deadline Options

Parameter Name	WdgMDeadlineMax
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMDeadlineSupervision/
Group	Local or global deadline
Type	Float
Range	0.0...((1/WdgMTicksPerSecond) * 65535) seconds
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the longest time span after which the deadline is still considered to be met. **Note:** The time span is counted from the point in time when the source checkpoint of the transition is reached.

Parameter Name	WdgMDeadlineMin
Path	WdgM/WdgMConfigSet/WdgMMode/WdgMDeadlineSupervision/

Group	Local or global deadline
Type	Float
Range	`0.0...((1/WdgMTicksPerSecond) * 65535)` seconds
Compatibility	AUTOSAR 4.0 r1
Description	This parameter contains the shortest time span after which the deadline is considered to be met. **Note:** The time span is counted from the point in time when the source checkpoint of the transition is reached.

Parameter Name	`WdgMDeadlineStartRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMDeadlineSupervision/`
Group	Local or global deadline
Type	Reference
Multiplicity	1
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the *source checkpoint* for deadline monitoring. **Note:** The start and stop references of a deadline must match an existing local or global transition.

Parameter Name	`WdgMDeadlineStopRef`
Path	`WdgM/WdgMConfigSet/WdgMMode/WdgMDeadlineSupervision/`
Group	Local or global deadline
Type	Reference

Multiplicity	`1`
Compatibility	AUTOSAR 4.0 r1
Description	This is the parameter for a reference to the *destination checkpoint* for deadline monitoring. **Note:** The start and stop references of a deadline must match an existing local or global transition.

2.6 ECU Description Configuration

2.6.1 Assumptions/Constraints

- There is a `WdgMTrigger` element for every `WdgMWatchdog` element; i.e., the former `WdgMTriggerWatchdogRef` always "points" to an existing `WdgMWatchdog` element.
- For the purpose of navigating within the ECU description file, we assume that every referenced element is identified by its `SHORT-NAME` element.

  Example: A `WdgMTrigger` element `WdgMTriggerWatchdogRef` attribute is a reference to a `WdgMWatchdog SHORT-NAME` element and not to its `WdgMWatchdogName` element.
- We expect the Checkpoint IDs to create a zero-based, monotonically increasing sequence of integers with no gaps.
- We expect that every `WdgMMode` element has a maximum of one `WdgMProgramFlowSupervision` subelement, which in turn has exactly one `WdgMGlobalCheckpointInitialRef` subelement.
- We expect that the `WdgMSupervisedEntityId` attribute of all `SupervisedEntity` instances in one ECU description file builds a zero-based, monotonically increasing sequence of integers with no gaps. This is a requirement because the embedded code uses the **Entity ID** as an array index when accessing `WdgMSupervisedEntity`.
- The ECU description files to be used for configuring the Watchdog Manager must belong to the **XML namespace** `"http://autosar.org/3.1.4".`

2.7 API Description

The **S-WdgM software module** is the top level layer of the **Safe Watchdog Manager Stack**. The S-WdgM software module contains the core functionality with supervised entity state machines and calculation of the S-WdgM global state. The S-WdgM communicates on one side through its **user API** with the **Application Layer** (optionally using RTE) and through its **system API** with the **Basic Software Components (BSW)** and, on the other side, with the S-WdgIf layer.

2.7.1 S-WdgM Type Definitions

This Section describes the **types of parameters** passed to the API functions of the **S-WdgM**.

Name	`WdgM_ConfigType`
Type	`Structure`
Range	N/A
Description	This is the type for the S-WdgM configuration structure. This structure is generated by the S-WdgM Configuration Generator 102.

Name	`WdgM_SupervisedEntityIdType`
Type	`uint16`
Range	`0...65534`
Description	This is the type for an individual supervised entity for the Safe Watchdog Manager. **Note:** If configuration parameter `WDGM_USE_RTE` is set to `STD_ON`, then this type is imported, otherwise it is generated.

Name	`WdgM_CheckpointIdType`
Type	`uint16`
Range	`0...65534`
Description	This is the type for a checkpoint in the context of a supervised entity for the S-WdgM. **Note:** If configuration parameter `WDGM_USE_RTE` is set to `STD_ON`, then this type is imported, otherwise it is generated..

Name	`WdgM_ModeType`
Type	`uint8`
Range	`0...255`
Description	This is the type for the ID of a trigger mode that was configured for the S-WgM. The current trigger mode can be retrieved with `WdgM_GetMode()`. **Note:** If configuration parameter `WDGM_USE_RTE` is set to `STD_ON`, then this type is imported, otherwise it is generated..

Name	`WdgM_LocalStatusType`
Type	`uint8`
Range	▪ `WDGM_LOCAL_STATUS_OK = 0` ▪ `WDGM_LOCAL_STATUS_FAILED = 1` ▪ `WDGM_LOCAL_STATUS_EXPIRED = 2` ▪ `WDGM_LOCAL_STATUS_DEACTIVATED = 4`
Description	This is the type for the local monitoring state of a supervised entity."The current local state of a supervised entity can be retrieved with `WdgM_GetLocalStatus()`.

	Note: If configuration parameter WDGM_USE_RTE is set to STD_ON, then this type is imported, otherwise it is generated..

Name	WdgM_GlobalStatusType
Type	uint8
Range	▪ WDGM_GLOBAL_STATUS_OK = 0, ▪ WDGM_GLOBAL_STATUS_FAILED = 1, ▪ WDGM_GLOBAL_STATUS_EXPIRED = 2, ▪ WDGM_GLOBAL_STATUS_STOPPED = 3, ▪ WDGM_GLOBAL_STATUS_DEACTIVATED = 4
Description	This is the type for the global monitoring state. It summarizes the local states of all supervised entities. The current global state can be retrieved with WdgM_GetGlobalStatus(). **Note**: If configuration parameter WDGM_USE_RTE is set to STD_ON, then this type is imported, otherwise it is generated..

Name	WdgM_TimeBaseTickType
Type	uint32
Range	0...2^{32-1}
Description	This is the type for the Timebase Tick.

Name	`Std_VersionInfoType`
Type	Structure
Range	N/A
Description	This is the parameter type of function `WdgM_GetVersionInfo()` 92.

2.7.2 S-WdgM Application Level API Functions

This Section describes the **S-WdgM API functions** that are imported or provided by the S-WdgM software module.

Syntax	`Std_ReturnType WdgM_SetMode`
	`(WdgM_ModeType Mode, uint16 CallerID)`

Service ID[hex]	`0x03`
Sync/Async	Synchronous
Reentrant?	Yes
Parameters (in)	`Mode`: The ID of the Trigger Mode to which the S-WdgM must be set. `CallerID:` ID of the caller allowed to call the function `WdgM_SetMode().` The allowed caller is defined in the configuration. The caller ID is checked if `WdgMDefensiveBehavior` is true.
Parameters (in/out)	None
Parameters (out)	None
Return value	`Std_ReturnType:` `E_OK:` The new Trigger Mode has been successfully set. `E_NOT_OK:` The setting of the new Trigger Mode failed.
Compatibility	AUTOSAR 4.0 r1 / TTTech
Description	This functions sets the *Trigger Mode* of the S-WdgM. The S-

	WdgM *Trigger Mode* is a set of Watchdog trigger times and Watchdog mode. The S-WdgM can have one or more *Trigger Modes* for every watchdog. In contrast to AUTOSAR, where the `Mode` represents a set of entities with all entity-specific parameters, the S-WdgM Trigger Mode only sets the following parameters: ▪ `WdgMTriggerConditionValue` ▪ `WdgMTriggerWindowStart` ▪ `WdgMWatchdogMode` **Note:** A change to trigger mode with `ID Mode` sets all configured watchdogs to the trigger mode with `ID Mode`. As a consequence, all watchdogs must have configured the same number of Trigger Modes. This function can be used to increase the S-WdgM supervision cycle in an MCU sleep mode.

Syntax	`Std_ReturnType WdgM_GetMode(WdgM_ModeType* Mode)`
Service ID[hex]	`0x0b`
Sync/Async	Synchronous
Reentrant?	Yes
Parameters (in)	None
Parameters (in/out)	None
Parameters (out)	`Mode`: Pointer to the current Trigger Mode ID of the Watchdog Manager
Return value	`Std_ReturnType`: ▪ `E_OK`: Current Trigger Mode successfully returned. ▪ `E_NOT_OK`: Returning current Trigger Mode failed.
Compatibility	AUTOSAR 4.0 r1/TTTech

Description	Returns the current Trigger Mode of the S-WdgM. The S-WdgM Trigger Mode represents one Watchdog trigger time and mode setting.

Syntax	`Std_ReturnType WdgM_CheckpointReached (WdgM_SupervisedEntityIdType SEID, WdgM_CheckpointIdType CheckpointID)`
Service ID[hex]	`0x0e`
Sync/Async	Synchronous
Reentrant?	Yes, reentrant in the context of a different supervised entity.
Parameters (in)	▪ `SEID`: Identifier of the supervised entity that reports a checkpoint. ▪ `CheckpointID`: Identifier of the checkpoint within a supervised entity that has been reached.
Parameters (in/ out)	None
Parameters (out)	None
Return value	`Std_ReturnType`: ▪ `E_OK`: Checkpoint monitoring successful. ▪ `E_NOT_OK`: Checkpoint monitoring fault. Returned in the following cases ○ `WDGM_E_NO_INIT`: Uninitialized S-WdgM (DET code `0x10`) ○ `WDGM_E_PARAM_SEID`: Wrong Id number of the supervised entity (DET code `0x13`) ○ `WDGM_E_CPID`: Invalid checkpoint ID number (DET code `0x16`) ○ `WDGM_E_PARAM_STATE`: Invalid S-WdgM state. Reset will be invoked (DET code `0x29`).
Compatibility	AUTOSAR 4.0 r1

Description	Indicates to the S-WdgM that a checkpoint within a supervised entity has been reached.

Syntax	`Std_ReturnType WdgM_GetLocalStatus (WdgM_SupervisedEntityIdType SEID, WdgM_LocalStatusType* Status)`
Service ID[hex]	`0x0c`
Sync/Async	Synchronous
Reentrant?	Yes
Parameters (in)	`SEID`: Identifier of the supervised entity whose monitoring state is returned.
Parameters (in/ out)	None
Parameters (out)	`Status`: Pointer to the local monitoring state of the given supervised entity.
Return value	`Std_ReturnType`: ▪ `E_OK`: Current monitoring state successfully returned. ▪ `E_NOT_OK`: Returning the current monitoring state failed.
Compatibility	AUTOSAR 4.0 r1
Description	Returns the monitoring state of the given supervised entity. **Note:** The S-WdgM updates the state inside the `WdgM_MainFunction()` every supervision cycle.

Syntax	`Std_ReturnType WdgM_GetGlobalStatus (WdgM_GlobalStatusType* Status)`
Service ID[hex]	`0x0d`

Sync/Async	Synchronous
Reentrant?	Yes
Parameters (in)	None
Parameters (in/ out)	None
Parameters (out)	`Status`: Pointer to global monitoring state of the S-WdgM.
Return value	`Std_ReturnType`: ▪ `E_OK`: Current global monitoring state successfully returned. ▪ `E_NOT_OK`: Watchdog reset failed.
Compatibility	AUTOSAR 4.0 r1
Description	Returns the global monitoring state of the S-WdgM. **Note:** The S-WdgM updates the state inside the `WdgM_MainFunction()` every supervision cycle.

Syntax	`Std_ReturnType WdgM_PerformReset(void)`
Service ID[hex]	`0x0f`
Sync/Async	Synchronous
Reentrant?	No
Parameters (in)	None
Parameters (in/ out)	None
Parameters (out)	None
Return value	`Std_ReturnType`: ▪ `E_OK`: This value will not be returned because the reset is activated, and the routine does not return.

	▪ `E_NOT_OK`: The function has failed.
Compatibility	AUTOSAR 4.0 r1
Description	Instructs the S-WdgM to cause an immediate watchdog reset. **Note:** ▪ This function is hardware-dependent. Some watchdogs do not support an immediate reset. Check the S-Wdg Driver documentation. ▪ This function can may direct access to hardware registers. Access to hardware registers can be dependent on hardware platforms and software architectures. Hence, the application that calls `WdgM_PerformReset()` must have the corresponding access rights.

Syntax	`Std_ReturnType WdgM_DeactivateSupervisionEntity`
	`(WdgM_SupervisedEntityIdType SEID)`

Re-entrant?	Yes
Parameters (in)	`SEID`: ID of the supervised entity to be deactivated. Range `[0...N]`
Parameters (in/out)	None
Parameters (out)	None
Return value	`Std_ReturnType`: ▪ `E_OK`: Marking the supervised entity for deactivation was successful. ▪ `E_NOT_OK`: Marking the supervised entity for deactivation failed.
Compatibility	TTTech, AUTOSAR 3.1 **Note:** Defined in the AUTOSAR 3.1 specification. This function is no longer available in the AUTOSAR 4.0 r1 specification.
Description	The function marks an entity for deactivation. An entity can only be deactivated when its local state is `WDGM_LOCAL_STATUS_OK` or `WDGM_LOCAL_STATUS_FAILED`. The deactivation itself

	happens at the end of the supervision cycle inside the `WdgM_MainFunction()`. When an entity is deactivated then its checkpoints are not evaluated anymore and the entity local state is `WDGM_LOCAL_STATUS_DEACTIVATED`. **Note:** ▪ When an entity is deactivated, the global transitions to this entity are not evaluated. ▪ Using this function can degrade system safety. The deactivation of entity supervision in safety-related products needs special attention to avoid unintended supervised entity deactivation. ▪ The function `WdgM_DeactivateSupervisionEntity()` can deactivate a supervised entity only before its **initial checkpoint** was passed or **after** its **end checkpoint** was passed. The focus here is on entities that are spread over more than one supervision cycles. **Note:** The local program flow of a supervised entity may span over more than one supervision cycle. Those active entities cannot be deactivated while running. Deactivating active SEs leads to a DEM error report. ▪ In the same call of `WdgM_MainFunction()`, first the supervised entity is deactivated, then the local states of all supervised entities and the global state are set. ▪ After SE deactivation the function `WdgM_GetLocalStatus()` can be used to check the SE local state. ▪ This function is only available if the preprocessor switch `WdgMEntityDeactivationEnabled` is set to `true` and if the entity option `WdgMEnableEntityDeactivation`[61] is set to `true`.

Syntax	`Std_ReturnType WdgM_ActivateSupervisionEntity (WdgM_SupervisedEntityIdType SEID)`
Parameters (in)	`SEID`: Supervised entity identifier.
Parameters (in/out)	None
Parameters (out)	None
Return value	`Std_ReturnType`:

	- `E_OK`: Marking the supervised entity for activation was successful. - `E_NOT_OK`: Marking the supervised entity for activation failed.
Compatibility	TTTech, AUTOSAR 3.1 **Note:** Defined in the AUTOSAR 3.1 specification, this function is no longer available in the AUTOSAR 4.0 r1 specification.
Description	The function marks an entity for activation. An entity can only be activated when its local state is `WDGM_LOCAL_STATUS_DEACTIVATED`. The activation itself happens at the end of the supervision cycle inside the `WdgM_MainFunction()`. **Note:** - This function can degrade system safety. The activation of entity supervision in safety-related products needs special attention to avoid unintended supervised entity deactivation. - In the same call of `WdgM_MainFunction()`, first the local states of all supervised entities and the global state are set, then the supervised entity is activated. - After SE activation the function `WdgM_GetLocalStatus()` can be used to check the SE local state. - This function is only available if the preprocessor switch `WdgMEntityDeactivationEnabled` is set to `true` and if the entity option `WdgMEnableEntityDeactivation` is set to `true`.

2.7.3 Callback Functions

Global state callback

When WDGM_STATE_CHANGE_NOTIFICATION == STD_ON and the S-WdgM global state changes, then the callback routine defined by the parameter WdgMGlobalStateChangeCbk 49 is called.

Local state callback

When WDGM_STATE_CHANGE_NOTIFICATION == STD_ON and the local state of a supervised entity changes, then the callback routine defined by the parameter WdgMLocalStateChangeCbk 63 is called.

2.7.4 S-WdgM System Level API Functions

This section describes the **function definitions** of the S-WdgM system level interface. The system level interface functions are not visible in the AUTOSAR application layer. The system functions are directly invoked by the BSW modules. The RTE does not generate interfaces for these functions.

Syntax	void WdgM_Init(const WdgM_ConfigType* ConfigPtr)
Service ID[hex]	0x00
Sync/Async	Synchronous
Reentrant?	No
Parameters (in)	ConfigPtr: Pointer to post-build configuration data
Parameters (in/ out)	None
Parameters (out)	None
Return value	None
Compatibility	AUTOSAR 4.0 r1
Description	The WdgM_Init() function initializes the S-WdgM. After the execution of this function, monitoring is activated according to the configuration of ConfigPtr. This function can be used during monitoring, too, but note that all pending violations are lost.

Syntax	`void WdgM_GetVersionInfo (Std_VersionInfoType* VersionInfo)`
Service ID[hex]	`0x02`
Sync/Async	Synchronous
Reentrant?	Yes
Parameters (in)	None
Parameters (in/ out)	None
Parameters (out)	`VersionInfo`: Pointer to where to store the version information of the S-WdgM module.
Return value	None
Compatibility	AUTOSAR 4.0 r1
Description	The `WdgM_GetVersionInfo()` function returns information about the version of this module. This includes the module ID, the vendor ID, and the vendor-specific version number.

Syntax	`void WdgM_MainFunction(void)`
Service ID[hex]	`0x08`
Timing	`FIXED_CYCLIC`
Reentrant?	No
Parameters (in)	None
Parameters (in/ out)	None
Parameters (out)	None
Return value	None

Compatibility	AUTOSAR 4.0 r1
Description	This function evaluates *monitoring data* gathered from the hit checkpoints in all supervised entities during the supervision cycle. Depending on the violation found (if there is any), the ▪ local state of the supervised entities and ▪ the S-WdgM global state are evaluated again. Depending on the resulting global state: ▪ the WD is triggered, or ▪ the WD trigger discontinues (safe state), or ▪ the WD is reset (safe state). The function must run at the end of every supervision cycle. It may be called by the *Basic Software Scheduler* or a task with a fixed period time. The `WdgM_MainFunction()` function is not reentrant. To prevent data inconsistency when it is interrupted by itself (e.g., due to schedule overload), the function checks if it is executed concurrently. If this function is started before its last instance has finished, it raises a development error. **Note:** ▪ Alive counter violations are detected at the end of every alive supervision reference cycle, ▪ program flow violations are detected at the end of every supervision cycle, ▪ continued program flow violations are detected at the end of every program flow supervision cycle. ▪ deadline violations are detected at the end of every supervision cycle, ▪ continued of deadline violations are detected at the end of every deadline supervision cycle. See also the *Safe Watchdog Manager Safety Manual* [5]→128.

Syntax	`void WdgM_UpdateTickCount(void)`
Service ID[hex]	None
Timing	`FIXED_CYCLIC`
Reentrant?	No
Parameters (in)	None
Parameters (in/ out)	None
Parameters (out)	None
Return value	None
Compatibility	TTTech
Description	This function increments the S-WdgM *Timebase Tick Counter* by one. When the precompile configuration parameter `WdgMTimebaseSource`44 is set to `WDGM_EXTERNAL_TICK`, then this function needs to be called periodically from outside the S-WdgM. The *Timebase Tick Counter* delivers the time base for deadline monitoring. In the AUTOSAR environment, this function can be called, for example, from a task with fixed time period and high priority.

2.7.5 Expected Interfaces

This section describes the **expected interfaces** to external modules used by the S-WdgM at BSW level (see Figure 18) and describes how to use the external interfaces with regard to safety (for detailed requirements on how to use external interfaces, see the *Safe Watchdog Manager Safety Manual* [5] 128).

Note: The external modules are AUTOSAR-defined modules.

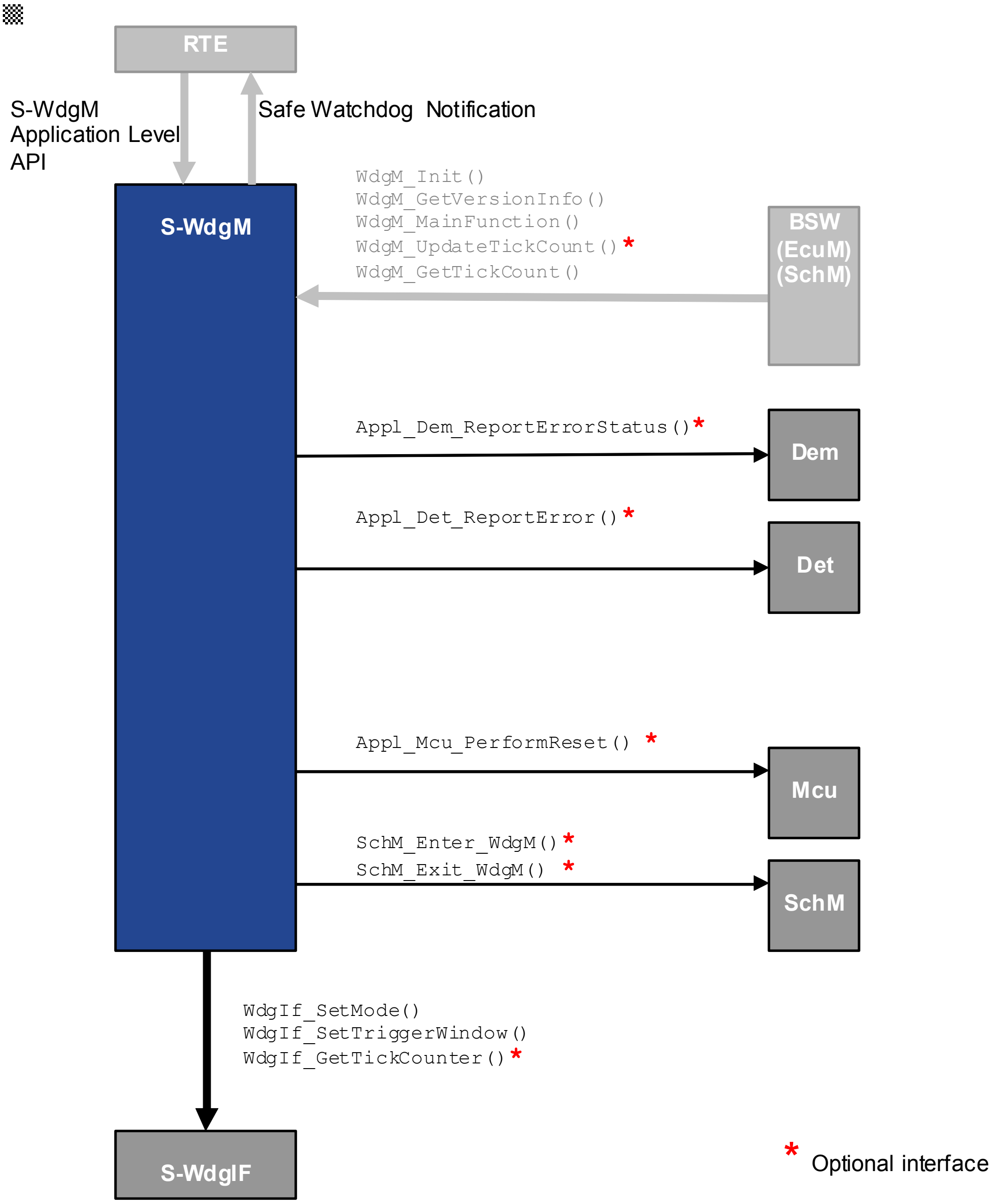


Fig. 18: Expected interfaces to external modules

Function	Description
Appl_Dem_ReportErrorStatus()	If the precompiler switch WdgMDemReport is set to STD_ON, the S-WdgM calls the function Dem_ReportErrorStatus() through the wrapper Appl_Dem_ReportErrorStatus(). **Safety aspect:** The DEM module may not meet the required quality level. The wrapper is implemented to guarantee freedom from interference. See the *Safe Watchdog Manager Safety Manual* [5] 128 for more information.
Appl_Det_ReportError()	If the precompiler switch WdgMDevErrorDetect is set to STD_ON, the S-WdgM calls the function Det_ReportError() through the wrapper Appl_Det_ReportError(). **Safety aspect:** The DET module may not meet the required quality level. The wrapper is implemented to guarantee freedom from interference. See the *Safe Watchdog Manager Safety Manual* [5] 128 for more information.
Appl_Mcu_PerformReset()	If the precompiler switch WDGM_SECOND_RESET_PATH is STD_ON, the S-WdgM calls the function Mcu_PerformReset() through the wrapper Appl_Mcu_PerformReset(). **Safety aspect:** The MCU module may not meet the required quality level. The wrapper is implemented to guarantee freedom from interference. See the *Safe Watchdog Manager Safety Manual* [5] 128 for more information.
SchM_Enter_WdgM() and SchM_Exit_WdgM()	If the precompiler switch WdgMUseOsSuspendInterrupt is set to STD_ON, the S-WdgM calls the functions SchM_Enter_WdgM() and SchM_Exit_WdgM(). **Safety aspect:** The SCHM module may not meet the required quality level. See the *Safe Watchdog Manager Safety Manual* [5] 128 for more information.

Note: If the precompiler switches

- WdgMDevErrorDetect,
- WdgMDemReport,
- WdgMUseOsSuspendInterrupt,
- WdgMImmediateReset and
- WDGM_SECOND_RESET_PATH

are set to false, the S-WdgM module does not call the corresponding function(s).

Note: The functions listed in the table above may not meet the required quality level and, thus, must be wrapped in order to ensure freedom from interference with the S-WdgM. The integrator must implement the `Appl_...()` functions according to the requirements specified in the *Safe Watchdog Manager Safety Manual* [5] ⇢128.

Note: The system integrator must revise the necessity of the expected interfaces. A called external function may degrade the quality level of the S-WdgM below the required quality level.

2.7.6 AUTOSAR 3.1 Compatibility Mode

If the parameter `WdgMSupportedAutosarAPI` ⇢62 is set to `API_3_1` , the S-WdgM is compiled in the AUTOSAR 3.1 compatibility mode. This means that its functionality is reduced to the functionality described by AUTOSAR 3.1.

The AUTOSAR 3.1 compatibility mode has the following configuration restrictions:

- Exactly one checkpoint must be defined for a supervised entity.
- The checkpoint must have an **initial attribute** and an **end attribute**.
- An Alive counter must be defined for the checkpoint.
- Local and global transitions are not allowed.
- The AUTOSAR 4.0 r1 supervised entities are not allowed.

Note: the AUTOSAR 3.1 compatibility mode must be differentiated from the AUTOSAR environment version in which the S-WdgM is integrated. The compatibility mode is related only to the functionality of the module.

2.7.6.1 *User API*

If the parameter `WdgMSupportedAutosarAPI` ⇢62 is set to `API_3_1` (embedded macro `WDGM_AUTOSAR_3_1_X_COMPATIBILITY = STD_ON`), then the S-WdgM provides the AUTOSAR 3.1 functions described in the table below. The table also shows the internal mapping of the AUTOSAR 3.1 to the AUTOSAR 4.0 r1 functions:

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_SetMode(Mode)`
Native S-WdgM function	`WdgM_SetMode(Mode, CallerID)`
Note	The `CallerID = 0` is added in the S-WdgM embedded code.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_GetMode(*Mode)`
Native S-WdgM function	`WdgM_GetMode(*Mode)`
Note	The function signature is the same.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_UpdateAliveCounter(SEID)`
Native S-WdgM function	`WdgM_CheckpointReached(SEID, CPID)`
Note	The `CPID = 0` is added in the S-WdgM embedded code

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_GetAliveSupervisionStatus(SEID, *status)`
Native S-WdgM function	`WdgM_GetLocalStatus(SEID, *status)`
Note	The function name is redefined in the file `WdgM_swc.arxml`.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_GetGlobalStatus(*status)`
Native S-WdgM function	`WdgM_GetGlobalStatus(*status)`
Note	The function signature is the same.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_ActivateAliveSupervision(SEID)`
Native S-WdgM function	`WdgM_ActivateSupervisionEntity(SEID)`
Note	The function name is redefined in the file `WdgM_swc.arxml`.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_DeactivateAliveSupervision(SEID)`
Native S-WdgM function	`WdgM_DeactivateSupervisionEntity(SEID)`
Note	The function name is redefined in the file `WdgM_swc.arxml`.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_GssChangeCbk(status)`
Native S-WdgM function	`WdgM_GlobalStateChangeCbk(status)`
Note	The function name is redefined in the file `WdgM_swc.arxml`.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_IssChangeCbk(status)`
Native S-WdgM function	`WdgM_LocalStateChangeCbk(status)`
Note	The function name is redefined in the file `WdgM_swc.arxml`.

2.7.6.2 *System API*

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_Init(&Config)`
Native S-WdgM function	`WdgM_Init(&Config)`
Note	The function signature is the same.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_GetVersionInfo(&versioninfo)`
Native S-WdgM function	`WdgM_GetVersionInfo(&versioninfo)`
Note	The function signature is the same.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_Cbk_GptNotification()`
Native S-WdgM function	`WdgM_UpdateTickCount()`
Note	In the AUTOSAR 3.1 environment the `WdgM_UpdateTickCount()` function is not used, because it is used in the AUTOSAR 4.0 r1 deadline monitoring only.

S-WdgM in AUTOSAR 3.1 compatibility mode	`WdgM_MainFunction_AliveSupervision()`
Native S-WdgM function	`WdgM_MainFunction()`
Note	In the AUTOSAR 3.1 and AUTOSAR 4.0 r1 environment, the native S-WdgM function (`WdgM_MainFunction()` 85) must be called.

3 Integration

3.1 Initialization of the S-WdgM

In a safety-related system, the initialization of the Watchdog device should be done as soon as possible after system start (at least before a QM task may compromise the initialization process). The Watchdog device starts the counter for the next expected trigger.

Note: The ways how the Watchdog device is initialized, configured, and how it reacts are platform-dependent and can be different. See the corresponding *Safe Watchdog Driver User Manual*.

The **time** between the **initialization** of the S-Wdg and the **first triggering** in function `WdgM_MainFunction()` (**Supervision cycle 0**) must match the Watchdog requirements. This time can be adapted in the S-Wdg configuration by changing the initial S-Wdg trigger window to meet the operating system start time requirements (see Figure 19).

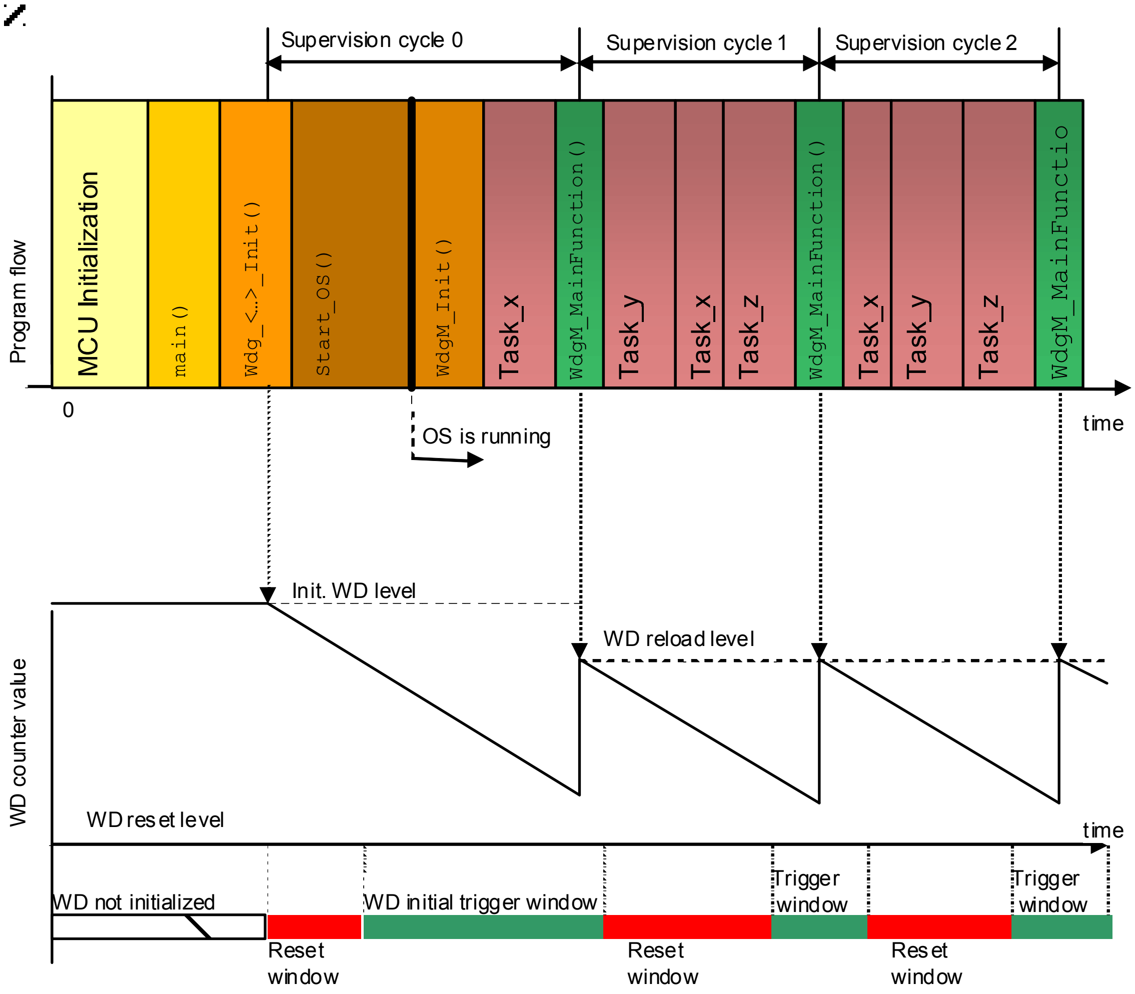


Fig. 19: Start phase of the S-WdgM

The y-axis in Figure 21 shows the **WD counter value**, which is reset after each trigger. Then the countdown runs until the S-Wdg is triggered again (within the **WD initial trigger window** or **Trigger window**) or **0** (**WD Reset level**) is reached (i.e., the window has been missed) so that a reset is performed.

Notes:

- Not all hardware platforms can configure a different trigger time for the first supervision cycle (cycle 0).
- In the first supervision cycle, the Alive counter evaluation can be suppressed by the parameter `WDGM_FIRSTCYCLE_ALIVECOUNTER_RESET` 48.
- The functions `WdgM_Init()` 84 and `WdgM_MainFunction()` 85 functions can be placed inside a task, too.
- The function `Wdg_<...>_Init()` can be placed before `main()`.

For safety reasons the S-WdgM uses **windowed triggering mode**. This means that watchdog triggering outside the defined window time causes a reset.

After the execution of function `WdgM_Init()` the supervision of configured entities is activated and the checkpoints can be executed (called).

3.2 Memory Sections

Memory segmentation into sections is especially important when memory protection is used in the system.

The S-WdgM uses three basic RAM data sections:

1. **Memory sections for local data of every SE:** This section contains local information about every supervised entity and, if defined, also the Alive counters. These variables are used by the `WdgM_CheckpointReached()` function and are part of the private SWC (task, application) memory and written only in the context of this SWC.

   Note: The S-WdgM does not protect this memory section.

2. **Memory sections for global data:** This section contains the S-WdgM global data such as S-WdgM global status and Timebase Tick counter. It is a S-WdgM private memory.

   Note: In the AUTOSAR environment, where QM and Safety-related modules are used together, the S-WdgM global data should be placed in a so-called **trusted memory section** to guarantee its safety and integrity.

3. **Memory sections for global shared data:** This section contains data such as the last active entity. This memory must be writable for all SWCs using the `WdgM_CheckpointReached()` function and for the `WdgM_Init()` function. As this is a memory where all the QM SWCs could write, the S-WdgM variables are

protected (stored double-inverted) by the S-WdgM itself. The S-WdgM checks the correctness of these variables with read operations. If a fault is detected, the S-WdgM initiates a reset.

Figure 20 shows the memory usage of the S-WdgM.

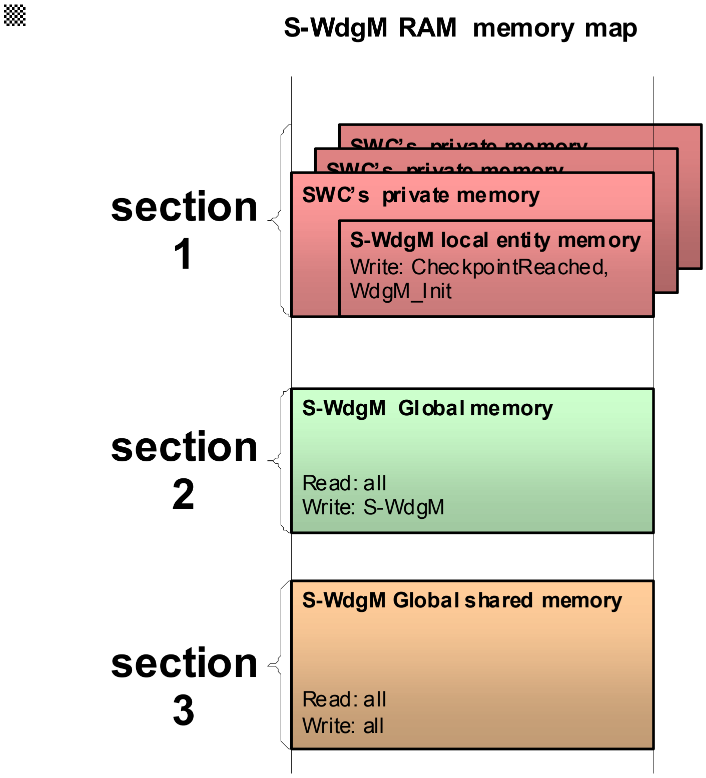


Fig. 20: Memory usage of the S-WdgM

Local entity memory: **Local entity data** is supervised entity private data. This is the data where the function `WdgM_CheckpointReached()` ⇾78 writes.

The S-WdgM Configuration Generator ⇾102 provides defines so that the status variables of every supervised entity can be placed in a separate RAM section. The declaration of every entity starts with the defines `WDGM_SE`*`i`*`_START_SEC_VAR_*` and ends with `WDGM_SE`*`i`*`_STOP_SEC_VAR_*`, where *i* is the ID of the supervised entity.

Theses defines are in the generated file `WdgM_MemMap.h` in an AUTOSAR 3.1 environment or `WdgM_OSMemMap.h` in an AUTOSAR 4.0 environment. Hence, it must be included in the file `MemMap.h`.

If the entity is linked to an OS task (through its ECU description parameter `WdgMAppTaskRef` ⇾64), then the supervised entity data is placed in a section embedded in *`appl_name`*`_START_SEC_VAR_*` and *`appl_name`*`_STOP_SEC_VAR_*`, where *`appl_name`* is the name of the application. In this case, the integrator must make sure to include the file `Os_MemMap.h` after the file `WdgM_MemMap.h` in file `MemMap.h`.

Global memory: **Global data** are **private** S-WdgM variables. The memory mapping defines are WDGM_GLOBAL_START_SEC_VAR_* and WDGM_GLOBAL_STOP_SEC_VAR_*.

This section can be mapped to an OS application (through its ECU description parameter WdgMGlobalMemoryAppTaskRef 49). For this mapping, the defines appl_name_START_SEC_VAR_* and appl_name_STOP_SEC_VAR_* are used, where appl_name is the name of the application. In this case, the integrator must make sure to include the file Os_MemMap.h after the file WdgM_MemMap.h in file MemMap.h.

As this section is internally not protected by the S-WdgM, it should be in a memory area where it cannot be corrupted.

Global shared memory: **Global shared data** should be placed in a **RAM section** where all tasks can read and write to that data.

The memory mapping defines are WDGM_GLOBAL_SHARED_START_SEC_VAR_* and WDGM_GLOBAL_SHARED_STOP_SEC_VAR_*. These variables are internally protected by the S-WdgM.

3.3 Timing Setup

The **timing** of the S-WdgM is defined by

- the **calling period** of function WdgM_MainFunction() 85 and,
- the **count period** of the S-WdgM **Tick Counter** (for **Deadline Monitoring**).

Every time when the function WdgM_MainFunction() 85 is invoked,

- the **Alive counters** are evaluated,
- running deadlines are checked for violations,
- checkpoint **fault indications** are **evaluated** and, finally,
- the S-WdgM **global status** of all supervised entities is **calculated**.

Note: The time period during which the function WdgM_MainFunction() 85 is called, is the **S-WdgM supervision cycle**. This cycle time is also used for the periodic triggering of the Watchdog device. The period of this cycle determines the shortest S-WdgM reaction time. For example: If the S-WdgM reaction time should be not more than 10 ms, the supervision cycle time should be set to 10 ms or shorter.

Figure 21 shows the S-WdgM timing configuration parameters. The parameters can be set by a Configuration Tool.

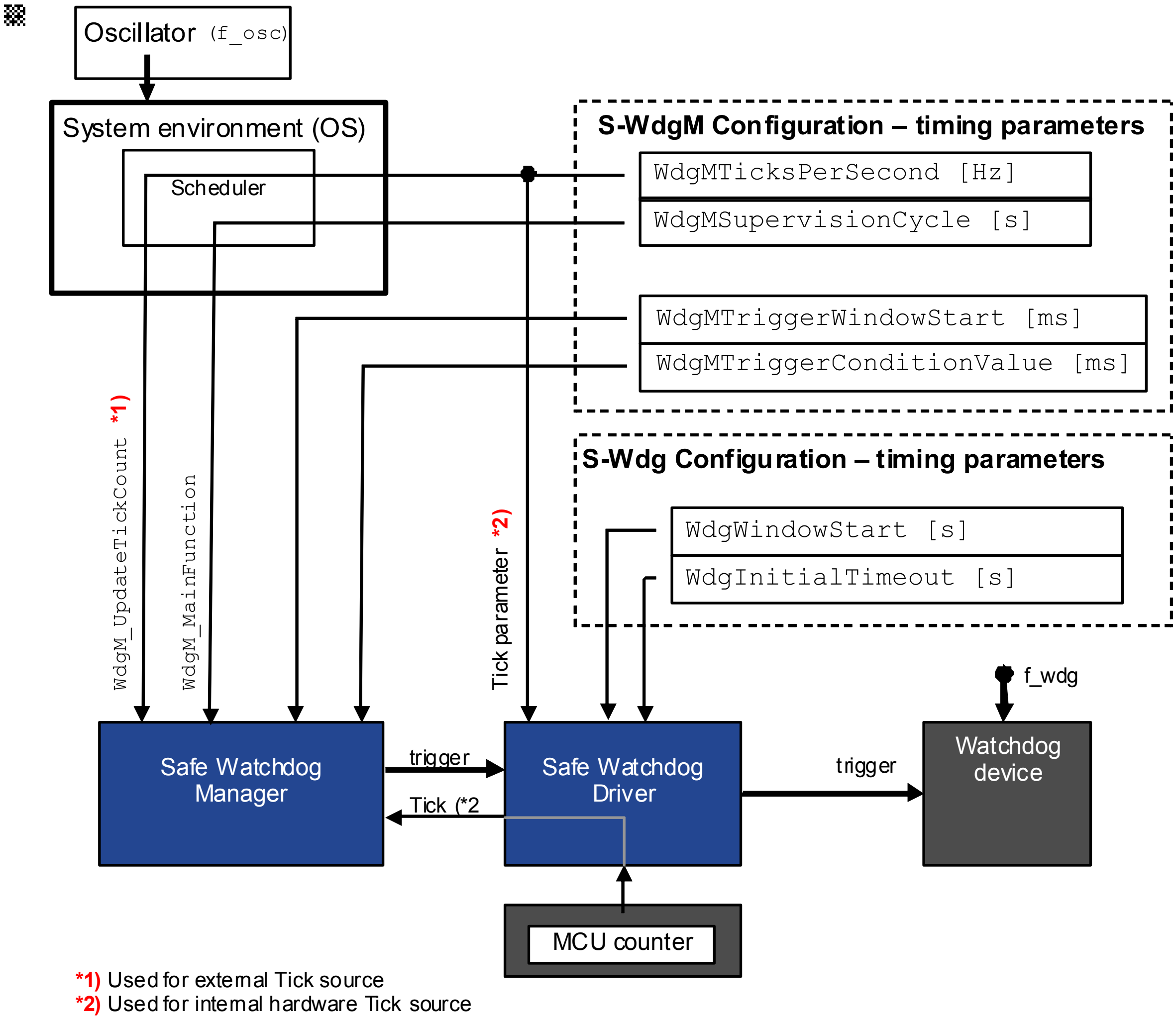


Fig. 21: Time base of the S-WdgM

Two configuration **parameters** shown in Figure 21 are used by the **System Environment** only. The Scheduler uses these parameters and periodically calls

- function WdgM_MainFunction() 85 and,
- if defined, also function WdgM_UpdateTickCounter().

All the other parameters are used by the S-WdgM and S-Wdg.

Configuration Parameter	Description
WdgMSupervisionCycle	This parameter defines the **time period** in which the S-WdgM performs **cyclic supervision**. This is the time period in which function WdgM_MainFunction() 85 is called. The user of this parameter is the system environment that periodically calls function WdgM_MainFunction(). The Watchdog device is triggered with every call of WdgM_MainFunction().
WdgMTicksPerSecond	This parameter defines the **frequency** by which the S-WdgM Tick counter is incremented. • If the **external Tick** counter is selected, the user of this parameter is the system environment that periodically calls function WdgM_UpdateTickCount() 87. • If the **internal hardware Tick** counter is selected, this parameter configures the frequency of the MCU counter. • The parameter WdgMTicksPerSecond 51 must not be zero.
WdgMTriggerWindowStart	This parameter defines, for **all** supervision cycles (except for the first), the lower limit of the Watchdog trigger window. If the Watchdog triggered before, a reset is caused. This parameter is in **milliseconds**. The user is the S-WdgM.
WdgMTriggerConditionValue	This parameter defines, for **all** supervision cycles (except for the first), the **upper limit** of the Watchdog trigger window. If the Watchdog is not triggered in time, a reset is caused. This parameter is in **milliseconds**. The user is the S-WdgM.
WdgWindowStart	This parameter defines, for the **first** supervision cycle, the **minimum window time** after which watchdog triggering is allowed. This parameter is used by the Safe Watchdog Driver only.
WdgInitialTimeout	This parameter defines, for the **first** supervision cycle, the **upper limit** of the Watchdog trigger window. If the Watchdog is not triggered in time, a reset is caused. This parameter is used by the Safe Watchdog Driver only (see the corresponding *Safe Watchdog Driver*

	User Manual).

3.3.1 Deadline Measurement and Tick Counter

The **transition time** between two checkpoints is measured in **Ticks**. The Tick Counter delivers a **time base** for **Deadline Monitoring**. The Tick counter is the smallest deadline time unit for the S-WdgM. There are three possible **Tick sources** (see Figure 22):

- **Internal hardware Tick source:** The tick source is an S-WdgM internal source derived from the MCU hardware counter. If the *internal hardware Tick source* is selected, the frequency is set by the parameter `WdgMTicksPerSecond`.
- **Internal software Tick source:** The Tick source is software-based where the internal counter is incremented every time the **S-WdgM main function** (`WdgM_MainFunction()`) is called. If the *internal software Tick source* is selected, the frequency is the same as `WdgM_MainFunction()` is called.
- **External Tick source:** The Tick must be counted externally by calling the S-WdgM function `WdgM_UpdateTickCount()`. If the *external Tick source* is selected, the system integrator is responsible for calling the function on a regular basis. The S-WdgM internally checks if the number of Ticks corresponds with the Supervision Cycle.

Note: The Tick source can be selected by setting the parameter `WdgMTimebaseSource`. The default parameter value is `WDGM_INTERNAL_SOFTWARE_TICK`.

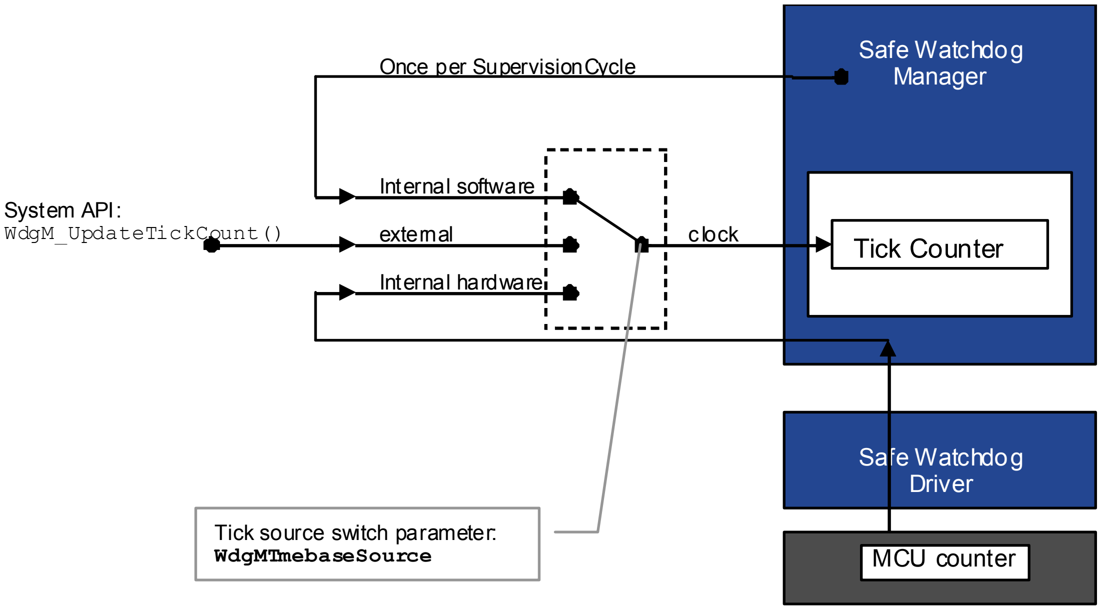


Fig. 22: S-WdgM Tick source selection for deadline monitoring

The **Ticks per second** must be configured for the S-WdgM to translate the monitored deadlines from **seconds** (as stored in the AUTOSAR ECU description files) to **S-**

WdgM ticks. This conversion is done during configuration generation for the S-WdgM, with the deadlines being stored in the generated configuration as S-WdgM ticks.

Note:

- Non-integer ticks are not allowed. If a deadline cannot be converted into an integer number of S-WdgM ticks, the S-WdgM Configuration Generator will report an error.
- For an **internal software Tick source** and an **external Tick source** the internal Tick counter is initialized to **`1`**.

Examples

- Let a S-WdgM Tick be **2 ms**. If there is a deadline of **3 ms**, it cannot be converted to S-WdgM ticks without loss of accuracy. It will be between 1 and 2 S-WdgM ticks.
- Let a S-WdgM Tick be **1 ms** (i.e., the parameter `WdgMTicksPerSecond` is set to `1000`). A **deadline of `0.002s=2ms`** is then translated to **2 S-WdgM ticks**. But a deadline of `0.0005s=0.5ms` cannot be translated to an integer number of S-WdgM ticks.

Note: There is a trade-off between the S-WdgM Tick resolution and performance. The shorter the Tick length, the finer the deadlines that can be monitored. However, the performance gets worse due to more frequent calls to the `WdgM_UpdateTickCount()` function.

4 Configuration Generation

4.1 S-WdgM Configuration Generator

The **S-WdgM Configuration Generator** is a Microsoft Windows console application that can be launched from a **command prompt** window by entering the command `Wdg_Mgr_Cfg_Gen.exe`. The S-WdgM Configuration Generator **reads** the S-WdgM **module information** from the AUTOSAR **ECU description file** (`*.arxml`) and generates configuration structures for the S-WdgM.

Note: Safety requirements must be considered for the generation process. These requirements are listed in the *Safe Watchdog Manager Safety Manual* [5]128, which also gives a detailed description of a verification process for the generated files using a separate tool. This verification process is mandatory for safety-related systems.

To use the S-WdgM Configuration Generator, enter the following command in a command prompt window:

```
Wdg_Mgr_Cfg_Gen.exe [options] <ECU-DESC-FILE> <OUTPUT-DIR>
```

`[options]`	**Description**
`--version`	Shows the application version number and license information, and then exits.
`-h/--help`	Shows this help message and exits.

Parameter	**Description**
`<ECU-DESC-FILE>`	The ECU description file (`*.arxml`). It is generated by a tool like the DaVinci Configurator, for example.
`<OUTPUT-DIR>`	The destination folder for the generated output. You must specify this parameter.

The S-WdgM Configuration Generator was developed and tested for **MS Windows 7** and can be integrated into a graphical configuration environment. The following **DLL**s must be present in the system:

- `OLEAUT32.dll`
- `USER32.dll`
- `POWRPROF.dll`
- `SHELL32.dll`

- `ole32.dll`
- `WSOCK32.dll`
- `ADVAPI32.dll`
- `WS2_32.dll`
- `VERSION.dll`
- `KERNEL32.dll`

The installer for this **DLL** is available at the Microsoft Download Center.

4.1.1 S-WdgM Configuration Verification

The **S-WdgM Verifier** is a TTTech tool for the **verification** of the generated **S-WdgM configuration**. The S-WdgM Verifier is delivered as a **DLL** (`wdgm_verifier.dll`) that must be compiled with the **configuration files** produced by the **generator** and the files produced by the **XSLT Processor**. The compilation result is a Windows `Verifier.exe` program. Running the Verifier generates a report file (`verifier_report.txt`) that contains the result of the verification.

Figure 23 shows the workflow of the S-WdgM Verifier build. For details, refer to the *Safe Watchdog Manager Safety Manual* [5] 128.

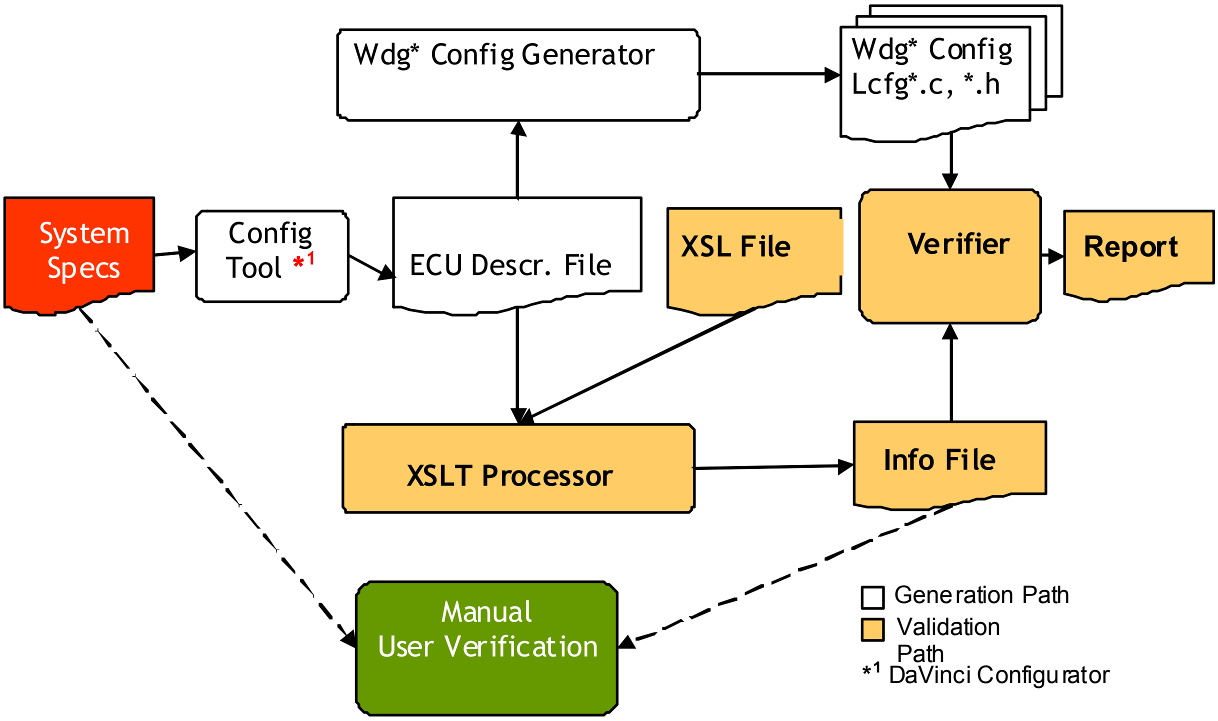


Fig. 23: Workflow of the S-WdgM Configuration Verifier build

4.1.1.1 *Installing the S-WdgM Verifier*

To run the **S-WdgM Verifier** an **XSLT Processor** and a working **gcc** environment are required.

The **XSLT Processor** can be installed by installing the **Configuration Tool** (DaVinci Configurator), and it consists of following files:

- `iconv.dll`,
- `libexslt.dll`,
- `libxml2.dll`,
- `libxslt.dll`,
- `zlib1.dll`,
- `xsltproc.exe`.

The recommended way to install **gcc** is to install the **MinGW** environment with the provided installer program (`MinGW-5.1.6.exe`) for Windows 7. To install gcc proceed as follows:

1. Start the installer program, accept the license terms and click **Next** until you are prompted to select a configuration.
2. When prompted, select **Minimal configuration**. There is no need to select any check boxes.
3. Complete the installation process after accepting the **default settings**.
4. Having installed **gcc**, add the `c:\MinGW\bin` directory to your search path by entering the command `set PATH=%PATH%;c:\MinGW\bin` in a command prompt window. Alternatively you can edit **Environment Variables** in the **System Properties** dialog (**Start** > **Control Panel** > **System**).

To verify that **gcc** is working, open a new command prompt window and enter `gcc --version` to let gcc show its version number.

4.2 Workflow

Figure 24 shows the workflow of how to generate and apply a configuration for the S-WdgM.

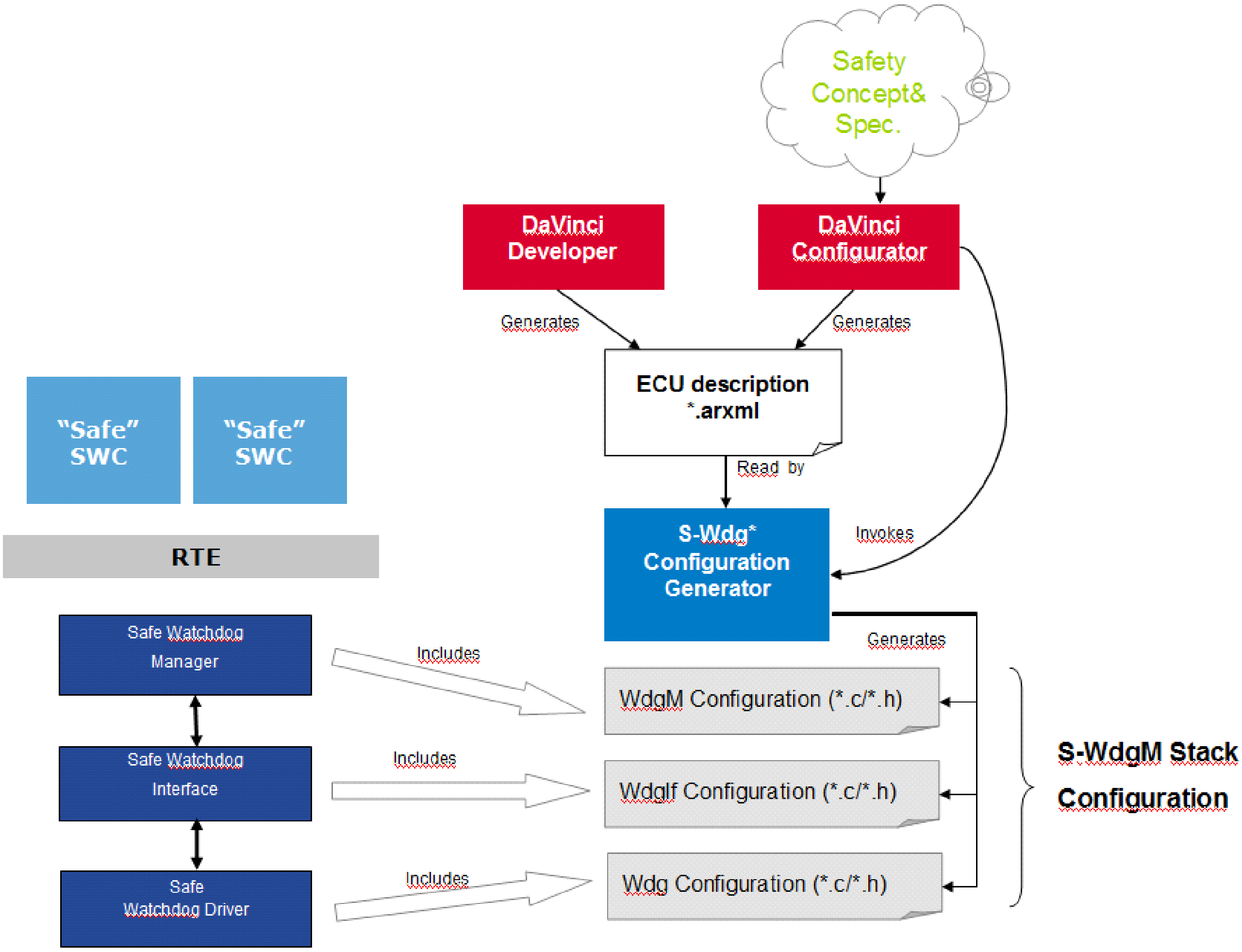


Fig. 24: Workflow of configuration generation and application for the S-WdgM

The **S-WdgM Configuration Generator** is the application that generates the configuration for the **S-WdgM**. The input used to generate a configuration is an **ECU description file (`*.arxml`)**. The ECU description file contains the configured **AUTOSAR WdgM**, **WdgIf** and **Wdg** modules. The S-WdgM configuration can be created and configured in several ways.

If you use the Vector tools **DaVinci Configurator Pro (DVC)** and **DaVinci Developer**, the workflow to generate the configuration is as follows:

- DVC is configured such that it uses the external generator S-WdgM Configuration Generator to generate the configuration for the modules S-WdgM, S-WdgIf and S-Wdg.

- During configuration generation, the S-WdgM Configuration Generator is automatically invoked and produces the configuration `*.c` and `*.h` files.
- If necessary, **DaVinci Developer** can be used to configure the **runtime environment (RTE)** for the **S-WdgM**. You can configure the software components that need to call **S-WdgM functions**, with the tool generating the respective RTE configuration files.

If you do not use the Vector tools mentioned above the workflow to create a configuration is as follows:

Start a **command prompt** window and enter the following command:

```
Wdg_Mgr_Cfg_Gen.exe <ecu_descr_file> <output_directory>
```

where `<ecu_descr_file>` is the name of the respective ECU description file (`*.arxml`) and

`<output_directory>` is the directory where to create the respective `*.c` and `*.h` files.

The S-WdgM code generator generates a configuration file, `WdgM_PBcfg.c` (see Section Configuration Generation 102), where all S-WdgM variables are defined and assigned to various memory sections (see Section Memory Sections 95).

The S-WdgM code generator also generates the file `WdgM_MemMap.h` in an AUTOSAR 3.1 environment or `WdgM_OSMemMap.h` in an AUTOSAR 4.0 environment, where the S-WdgM memory sections defined in the `WdgM_PBcfg.c` file are assigned to user-defined application sections or other system sections. The relation between memory sections and applications can be defined with a tool such as DaVinci Configurator using the parameters `WdgMAppTaskRef` and `WdgMGlobalMemoryAppTaskRef`.

The following example of a **`WdgM_MemMap.h`** file places the status variables for a supervised entity with index **1** to the application memory section called `Application_1_START_SEC_VAR_NOINIT_UNSPECIFIED` and `Application_1_STOP_SEC_VAR_NOINIT_UNSPECIFIED`:

```
/* Supervised Entity SE1 */
#ifdef WDGM_SE1_START_SEC_VAR_NOINIT_UNSPECIFIED
#undef WDGM_SE1_START_SEC_VAR_NOINIT_UNSPECIFIED
#define Application_1_START_SEC_VAR_NOINIT_UNSPECIFIED
#endif
#ifdef WDGM_SE1_STOP_SEC_VAR_NOINIT_UNSPECIFIED
#undef WDGM_SE1_STOP_SEC_VAR_NOINIT_UNSPECIFIED
#define Application_1_STOP_SEC_VAR_NOINIT_UNSPECIFIED
#endif
```

If no application is assigned with the parameters `WdgMAppTaskRef` or

WdgMGlobalMemoryAppTaskRef, then the prefix is WDGM_ instead of the application name.

All **global shared data** used by the S-WdgM are protected by S-WdgM against corruption.

4.3 Output Files

The following output files are generated for the respective **platform type** (<platform>), where <platform> is the respective **hardware platform** used, e.g., MPC5604 or TMS570LS3xx:

- WdgM_PBCfg.c
- WdgM_PBCfg.h
- WdgM_MemMap.h (in an AUTOSAR 3.1 environment) or WdgM_OSMemMap.h (in an AUTOSAR 4.0 environment)
- WdgM_Cfg_Features.h

The file WdgM_PBCfg.c contains the main configuration structure with the default name WdgMConfig_Mode0. This configuration name should be used by the initialization function, i.e., by call WdgM_Init(&WdgMConfig_Mode0). If necessary, the non-standard AUTOSAR name WdgMConfig_Mode0 can be renamed to WdgMConfigSet in the Configuration Tool (e.g, DaVinci).

Since the S-WdgM Configuration Generator is not trusted, the generated code must be verified. For details on the configuration verification process, refer to the *Safe Watchdog Manager Safety Manual* [5] ⇒128.

4.4 Error Messages

The generator will show an **error message** in the command prompt window and quit if something goes wrong during configuration generation.

4.4.1 Basic Errors

Error No.	Error Message
1	`Bad call syntax.`
2	``Cannot open ECU description file `%s`.``
3	``Cannot convert float parameter `%s/%s` to an Watchdog ticks.``
4	``Cannot convert `%s` to a numerical value.``
5	`Fatal error.`
6	``Method `%s` must be implementd by subclass of `%s`.``
7	`Missing WdgM data.`

4.4.2 Semantic Errors

Error No.	Error Message
1001	``Checkpoint IDs belonging to Supervised Entity `%s` are not a zero-based list of increasing integers without gaps.``
1002	`No WdgMMode elements found.`
1003	``Supervised Entity `%s`: local transition starts at Checkpoint with an ID %d.``
1004	`No WdgMMode element with WdgMModeId %d found.`
1005	``ECU Description File has no `WdgM` element.``
1006	``Referencing non-existing checkpoint `%s`.``
1015	``No value found for parameter defined by `%s` in `Element `%s`.``
1016	``Supervised Entity `%s` has no checkpoints.``
1017	``Supervised Entity `%s` defines local transitions for alien checkpoint(s) `%s`.``
1018	``Local Transition `%s` references alien checkpoint `%s`.``
1019	``Local Transition `%s` references wrong destination entity `%s`.``
1020	``Local Transition `%s` references wrong source entity `%s`.``
1021	``Cannot convert float parameter `%s/%s` (%.6f [s]) to an integral number of Watchdog ticks. (Using %.2f ticks per second).``

1025	Ignoring `WdgMGeneral/WdgMNumberOfSupervisedEntities`.
1026	Found more than one `WdgMMode` elements; generating code for mode with ID %d.
1027	Cannot find top level element %s.
1028	No value found for `#define %s`. Verify element defined by `.../%s`.
1029	No `.../WdgM/WdgMGeneral/WdgMWatchdog` elements found.
1031	No transition found for WdgMDeadlineSupervision `%s` between Supervised Entity `%s`, Checkpoint `%s` and Supervised Entity `%s`, Checkpoint `%s`.
1034	Found a `REFERENCE-VALUE` element defined by `%s` without a `VALUE-REF` child element.
1035	Cannot find `REFERENCE-VALUE` element defined by `%s`.
1036	Checkpoint `%s` has no ID.
1037	Checkpoint `%s` has no `VALUE` element for its ID.
1038	Missing `SHORT-NAME` element.
1039	No global initial Supervised Entity found.
1040	Program Flow Supervision has no checkpoint defined by %s.
1043	Watchdog `%s` has no `WdgMTrigger` element assigned to it.
1044	Cannot identify driver.
1045	No `WdgMLocalStatusParams` element found.
1048	Cannot find checkpoint ID for `%s/%s`.
1049	Cannot find checkpoint ID for `%s/%s`.
1050	Cannot find checkpoint ID for `%s`.
1051	`%s` is an AUTOSAR 3.1 Supervised Entity and therefore shall have exactly one checkpoint and this checkpoint shall have its ID set to 0.
1052	Supervised Entity `%s`: `WdgMFailedProgramFlowRefCycleTol` is positive (%d) but `WdgMProgramFlowRefCycle` is not (%d).
1053	Supervised Entity `%s`: Zero tolerance for program flow violations - `WdgMProgramFlowRefCycle` set to %d and `WdgMFailedProgramFlowRefCycleTol` set to zero.
1054	Supervised Entity `%s`: `WdgMFailedDeadlineRefCycleTol` is

	positive (%d) but `WdgMDeadlineReferenceCycle` is not (%d).
1055	Supervised Entity `%s`: Zero tolerance for dealine violations - `WdgMDeadlineReferenceCycle` set to %d and `WdgMFailedDeadlineRefCycleTol` set to zero.
1056	WdgMAliveSupervision `%s` (checkpoint `%s`): `WdgMSupervisionReferenceCycle` (%d) must be a positive value.
1057	Supervised Entity `%s`: `WdgMFailedSupervisionRefCycleTol` set to a positive value (%d) but there is no alive counter attached to any of its checkpoints.
1058	Mandatory `LocalStatusParams` data is missing.
1059	Shortest maximum deadline (%s: %f seconds) is shorter than `WdgMSupervisionCycle` (%f seconds).
1060	Mode with ID `%d` (`WdgMTicksPerSecond`: %d; `WdgMSupervisionCycle`: %f) fails to meet timing requirement `(1 / WdgMTicksPerSecond) <= WdgMSupervisionCycle`.
1061	Watchdog `%s`, trigger mode ID %s: the requirement `WdgMTriggerWindowStart <= WdgMSupervisionCycle <= WdgMTriggerConditionValue` is not fulfilled
1062	Verify that every Supervised Entity has a unique ID.
1063	No local incoming transitions defined for checkpoint `%s` in Supervised Entity `%s`. Reaching `%s` will trigger a Program Flow violation.
1064	Supervised Entity `%s` has no initial checkpoint.
1065	Callback function(s) `%s` will never be executed because `WDGM_STATE_CHANGE_NOTIFICATION` is turned off.
1066	`WDGM_STATE_CHANGE_NOTIFICATION` is turned on but there is no callback function defined. Verify the `WdgMGlobalStateChangeCbk` and `WdgMLocalStateChangeCbk` values
1068	Ensure that Supervised Entities have callback functions with a unique name.
1069	Local end checkpoint %s/%s must not be the source of a local transition.
1070	Local init checkpoint %s/%s must not be the destination of a local transition.

1071	The Supervised Entity IDs are not a zero-based list of integers without gaps.
1072	The watchdog driver is called in the context of the watchdog manager and its global variables must be placed in the same section as the watchdog manager's global variables in the presence of memory protection! (The watchdog driver global variables are placed in \`%s\` and the watchdog manager global variables are placed in \`%s\`).
1073	This driver configuration generator supports %s -- %s is not supported.
1075	The targeted precision (%d ticks per second) is too high; please lower the resolution (\`.../WdgMMode/WdgMTicksPerSecond\`).
1076	There is no WdgMTrigger element associated to Watchdog \`%s\`.
1081	No drivers found
1082	No Watchdog Interface devices found
1083	Watchdog IF device \`%s\` references non-existing Watchdog \`%s\`
1084	Watchdog \`%s\` references non-existing Watchdog IF device \`%s\`
1085	\`WdgMTicksPerSecond\` must not be zero if the Watchdog Manager uses an external tick counter source for deadline monitoring.
1086	Supervised Entity \`%s\` contains more than one checkpoint having an alive counter
1090	No Supervised Entities found!
1091	Transition \`%s\` references non existing checkpoint \`%s\` in entity \`%s\`.
1092	ECU Description File references non-existing checkpoint \`%s\` in Supervised Entity \`%s\`.
1093	Supervised Entity \`%s\` contains references to non-existing checkpoint(s) \`%s\`.
1094	Global Transition \`%s\` has non-existing Entity \`%s\` as source.
1095	Global Transition \`%s\` has non-existing Entity \`%s\` as destination.
1096	WdgMDeadlineSupervision \`%s\`: \`WdgMDeadlineMin\` (%s) is greater than \`WdgMDeadlineMax\` (%s).
1097	The \`%s\` value (%s [s]) of \`%s\` must not be greater than %s [s].
1098	For the INTERNAL_SOFTWARE_TICK the \`(1 / WdgMTicksPerSecond[Hz])

	= WdgMSupervisionCycle[s]` relation must be kept; the configured values for `WdgMTicksPerSecond` (%s) and `WdgMSupervisionCycle` (%s) do not fulfill this requirement.
1099	This ECU Description File's AUTOSAR version (%s) is not compatible with the version supported by this configuration generator (%s)
1100	This ECU Description File's AUTOSAR version (%s) has a different minor number than the version supported by this configuration generator (%s)
1101	Watchdog Driver `%s` is configured to have an active tick counter but the Watchdog Manager is not configured to have an internal HW timebase.
1102	The Watchdog Manager is configured to use an internal HW counter but the Watchdog Interface is not.
1103	The Watchdog Interface is configured to use an internal HW counter but the Watchdog Manager is not
1104	The Watchdog Manager is configured to use an internal HW tick counter but the Watchdog driver `%s` has no active tick counter.
1105	Error while reading list of `WdgMCallerIds`
1106	The Watchdog Manager is configured to use an internal HW tick counter but the Watchdog Interface does not reference any Watchdog Driver at all.
1107	The Watchdog Manager is not configured to use an internal HW tick counter but the Watchdog Interface has a reference to a Watchdog Driver with an internal tick counter.
1108	Every `WdgWatchdog` has to have the same number (either %d or %d) of associated `WdgMTrigger` elements.
1109	Verify that the Trigger Modes belonging to each trigger have IDs building a zero-based integer sequence without any gaps
1110	Invalid `WdgMInitialTriggerModeId` value (%d).
1111	The `SafeTcore` platform requires `WdgWindowStart` = 0 [ms]. (Current value: %s)
1112	`WdgMWatchdogMode` is set to `WDGIF_OFF_MODE`: `WdgMTriggerConditionValue` and `WdgMTriggerWindowStart` will be ignored

1113	Ticks per second must be greater than zero
1114	Multiple `WdgMDeadlineSupervision` elements defined for the transition from %s/%s to %s/%s
1115	OS partition reset is currently not supported.
1116	The current version supports only configurations having only one Watchdog, one IF device and one driver.
1119	The value 65535; e.g., 2^16 -1, must not be assigned to any of these elements: `WdgMFailedDeadlineRefCycleTol`, `WdgMFailedProgramFlowRefCycleTol` and `WdgMFailedSupervisionRefCycleTol`.
1120	Cannot find a VALUE element for `...WdgMConfigSet/WdgMMode/WdgMInitialTriggerModeId’
1121	Cannot find a VALUE element for ...WdgMConfigSet/WdgMMode/WdgMTrigger/WdgMTriggerModeId`
1122	Global transition connecting checkpoints `%s` and `%s` in the same entity `%s` is not allowed.
1123	`WdgMSupervisionCycle` (%s) is not greater than zero
1124	Watchdog `%s`, trigger mode ID %s: `WdgMTriggerConditionValue` is not greater than zero.

5 Appendix

List of Generator and Verifier checks.

5.1 Watchdog Manager Configuration Verifier Requirements

5.1.1 General Remarks

The verifier detects **three** kinds of errors:

1. deltas between the **ECU Description File (EDF)** and the generated configuration,
2. **errors** in the **configuration** which might have a negative impact on the embedded code (worst case could be to make it crash),
3. **integrity checks** already required to be implemented by the generator.

5.1.2 General Requirements

The S-WdgM Verifier must handle a (broken) configuration with **no supervised entities** at all (even though the S-WdgM Configuration Generator would not generate a configuration out of an EDF with no supervised entities at all).

The S-WdgM Verifier must handle a (broken) configuration with **no checkpoints** at all (even though the S-WdgM Configuration Generator would not generate a configuration out of an EDF with no checkpoints at all).

5.1.3 Deltas the S-WdgM Verifier Must Detect between the EDF and the Generated Configuration

Test No.	Requirement
Test 1	The number of CPs according to the EDF and the number of CPs referenced by SEs entities must match.
Test 2	The number of CPs according to the EDF and the number of CPs stored in the `NrOfAllCheckpoints` member of the main structure must match.
Test 3	The number of local transitions according to the EDF must match the number of local transitions referenced by CPs according to the corresponding `NrOfLocalTransitions` members.
Test 4	The number of global transitions according to the EDF must match the number of global transitions referenced by CPs according to the corresponding `NrOfGlobalTransitions` members.

Test 5	The number of SEs according to the EDF must match the value of the `NrOfSupervisedEntities` member of the main structure.
Test 17	The `NrOfStartedGlobalTransitions` value of a CP must match the number of global transitions having that CP as a starting point according to the EDF.
Test 19	If an alive supervision is defined for a CP, then the `WdgMExpectedAliveIndications` 65 member of that CP must match the number of expected alive indications of the alive supervision, as specified in the EDF.
Test 20	If an alive supervision is defined for a CP, then the `WdgMMinMargin` 66 member of that CP must match the corresponding attribute (`.../WdgMMinMargin`) in the alive supervision, as specified in the EDF.
Test 21	If an alive supervision is defined for a CP, then the `WdgMMaxMargin` 66 member of that CP must match the corresponding attribute (`.../WdgMMaxMargin`) in the alive supervision, as specified in the EDF.
Test 22	If an alive supervision is defined for a CP, then the `WdgMSupervisionReferenceCycle` 66 member of that CP must match the corresponding attribute (`.../WdgMSupervisionReferenceCycle`) in the alive supervision, as specified in the EDF.
Test 27	The `NrOfLocalTransitions` value of a CP must be set to the number of local transitions having that CP as a destination point according to the EDF.
Test 28	The `NrOfGlobalTransitions` value of a CP must be set to the number of global transitions having that CP as a destination point according to the EDF.
Test 32	If no alive supervision is defined for a CP, then the `WdgMExpectedAliveIndications` 65 member of that CP must be zero (see Test 19).
Test 33	If no alive supervision is defined for a CP, then the `WdgMMinMargin` 66 member of that CP must be zero (see Test 20).
Test 34	If no alive supervision is defined for a CP, then the `WdgMMaxMargin` 66 member of that CP must be zero (see Test 21).
Test 35	If no alive supervision is defined for a CP, then the `WdgMSupervisionReferenceCycle` 66 member of that CP must be zero (see Test 22).
Test 37	`WdgM_TransitionType->WdgMDeadlineMin` 69 must match the corresponding value in the EDF.
Test 38	`WdgM_TransitionType->WdgMDeadlineMax` 69 must match the corresponding value in the EDF.
Test 39	`WdgM_GlobalTransitionType->WdgMDeadlineMin` 69 must match the corresponding value in the EDF.

Test 40	`WdgM_GlobalTransitionType->WdgMDeadlineMax` 69 must match the corresponding value in the EDF.
Test 41	The `WdgMitialStatus` value of each SE must match the value entered as `WdgMSupervisedEntityInitialMode` 58 for the `WdgMLocalStatusParams` element assigned to the SE.
Test 42	For every entity: `X` must match `Y`, where `X` is the `WdgMFailedSupervisionRefCycleTol` 57 member of an SE in the generated configuration and `Y` is the element `WdgMFailedSupervisionRefCycleTol` 57 in the `WdgMLocalStatusParams` defined for the same entity in the EDF.
Test 43	For every entity: `X` must match `Y`, where `X` is the `WdgMFailedDeadlineRefCycleTol` 58 member of an SE in the generated configuration and `Y` is the element `WdgMFailedDeadlineRefCycleTol` 58 in the `WdgMLocalStatusParams` defined for the same entity in the EDF.
Test 44	For every entity: `X` must match `Y`, where `X` is the `WdgMDeadlineReferenceCycle` 59 member of a that supervised entity in the generated configuration and `Y` is the element `WdgMDeadlineReferenceCycle` 59 in the `WdgMLocalStatusParams` defined for the same entity in the EDF.
Test 45	For every entity: `X` must match `Y`, where `X` is the `WdgMFailedProgramFlowRefCycleTol` 59 member of an SE in the generated configuration and `Y` is the element `WdgMFailedProgramFlowRefCycleTol` 59 in the `WdgMLocalStatusParams` defined for the same entity in the EDF.
Test 46	For every entity: `X` must match `Y`, where `X` is the `WdgMProgramFlowReferenceCycle` 60 member of a that supervised entity in the generated configuration and `Y` is the element `WdgMProgramFlowReferenceCycle` 60 in the `WdgMLocalStatusParams` defined for the same entity in the EDF.
Test 47	Each SE in the generated configuration must have its `OSApplication` set to `WDGM_INVALID_OSAPPLICATION`.
Test 85	The set of relations between alive supervisions and CPs in the EDF is the same as in the generated configuration file, i.e. each CP has on both sides either the same or no alive supervision associated. **Note:** Related to Error 1092 111.
Test 86	In the generated configuration file, for each SE: All CPs that are referenced in the SE are defined (in array `WdgMCheckPoint`). **Note:** This includes the check for references by `CP-ID` and references by address to CP-list item (related to Error 1093) 111.
Test 89	The `WdgMGeneral` parameter `WdgMVersionInfoApi` 40 and the constant `WDGM_VERSION_INFO_API` 40 defined in `WdgM_Cfg_Features.h` must match.
Test 90	The `WdgMGeneral` parameter `WdgMDevErrorDetect` 38 and the constant `WDGM_DEV_ERROR_DETECT` 38 defined in `WdgM_Cfg_Features.h` must match.
Test 91	The `WdgMGeneral` parameter `WdgMDemReport` 38 and the constant `WDGM_DEM_REPORT` 38 defined in WdgM_Cfg_Features.h must match.

Test 92	The WdgMGeneral parameter WdgMDefensiveBehavior 41 and the constant WDGM_DEFENSIVE_BEHAVIOR 41 defined in WdgM_Cfg_Features.h must match.
Test 93	The WdgMGeneral parameter WdgMImmediateReset 39 and the constant WDGM_IMMEDIATE_RESET 39 defined in WdgM_Cfg_Features.h must match.
Test 94	The WdgMGeneral parameter WdgMOffModeEnabled 40 and the constant WDGM_OFF_MODE_ENABLED 40 defined in WdgM_Cfg_Features.h must match.
Test 95	The WdgMGeneral parameter WdgMUseOsSuspendInterrupt 43 and the constant WDGM_USE_OS_SUSPEND_INTERRUPT 43 defined in WdgM_Cfg_Features.h must match.
Test 96	The WdgMGeneral parameter WdgMTimebaseSource 44 and the constant WDGM_TIMEBASE_SOURCE 44 defined in WdgM_Cfg_Features.h must match.
Test 97	The WdgMGeneral parameter WdgMSecondResetPath 45 and the constant WDGM_SECOND_RESET_PATH 45 defined in WdgM_Cfg_Features.h must match.
Test 98	The WdgMGeneral parameter WdgMTickOverrunCorrection 46 and the constant WDGM_TICK_OVERRUN_CORRECTION 46 defined in WdgM_Cfg_Features.h must match.
Test 99	The WdgMGeneral parameter WdgMEntityDeactivationEnabled 47 and the constant WDGM_ENTITY_DEACTIVATION_ENABLED 47 defined in WdgM_Cfg_Features.h must match.
Test 100	The WdgMGeneral parameter WdgMStateChangeNotification 47 and the constant WDGM_STATE_CHANGE_NOTIFICATION 47 defined in WdgM_Cfg_Features.h must match.
Test 101	The WdgMGeneral parameter WdgMUseRte 42 and the constant WDGM_USE_RTE 42 defined in WdgM_Cfg_Features.h must match.
Test 102	The WdgMGeneral parameter WdgMDemSupervisionReport 42 and the constant WDGM_DEM_SUPERVISION_REPORT 42 defined in WdgM_Cfg_Features.h must match.
Test 103	The WdgMGeneral parameter WdgMFirstCycleAliveCounterReset 48 and the constant WDGM_FIRSTCYCLE_ALIVECOUNTER_RESET 48 defined in WdgM_Cfg_Features.h must match.
Test 104	The value WDGM_GLOBAL_TRANSITIONS in WdgM_Cfg_Features.h must be STD_ON if the configuration includes global transitions and STD_OFF otherwise.
Test 105	The value WDGM_AUTOSAR_3_1_X_COMPATIBILITY in WdgM_Cfg_Features.h must be STD_ON if there is at least one SE with its attribute WdgMSupportedAutosarAPI 62 set to the enumeration value API_3_1. Otherwise this value must be STD_OFF.

Test 106	The value WDGM_MULTIPLE_TRIGGER_MODES must be STD_ON if WdgMTrigger elements have more than one WdgMTriggerMode subelement. Otherwise this value must be STD_OFF. **Note:** It is required elsewhere that all triggers have the same amount of trigger modes. Therefore you can take any trigger for performing this test.

5.1.4 Integrity Checks

Test No.	Requirement
Test 18	If the WdgMIsEndCheckpointGlobal value of a CP is TRUE, then that CP must not be the source of any global transition.
Test 23	The WdgMAliveLRef value of a CP must only be NULL_PTR if and only if there is no alive supervision defined for that CP.
Test 24	The WdgMAliveGRef value of a CP must only be NULL_PTR if and only if there is no alive supervision defined for that CP.
Test 25	The WdgMDeadlineMonitoring value of a CP must be set to TRUE if that CP is the source or destination of at least one transition with associated deadline monitoring. Otherwise this value will be set to FALSE.
Test 26	The WdgMOutgoingDeadlineMax value of a CP must be set to the maximum deadline associated to any of the transitions having that CP as a starting point.
Test 29	The WdgMLocalTransitionRef member of a CP must be set to NULL_PTR if and only if there are no local transitions having that CP as a destination point.
Test 30	The WdgMGlobalTransitionsRef member of a CP must be set to NULL_PTR if and only if there are no global transitions having that CP as a destination point.
Test 31	The WdgMStartsAGlobalTransition value of a CP must be set to TRUE if that CP is the starting point of a global transition. Otherwise this value must be set to FALSE.
Test 48	The following condition must be fulfilled for each SE: Either WdgMFailedProgramFlowRefCycleTol [59] is greater than zero, or WdgMProgramFlowReferenceCycle [60] is zero (see Error 1053 [109])
Test 49	The following condition must be fulfilled for each SE: Either WdgMFailedDeadlineRefCycleTol [58] is zero, or WdgMDeadlineReferenceCycle [59] is greater than zero (see Error 1054 [109]).
Test 50	The following condition must be fulfilled for each SE: Either WdgMFailedDeadlineRefCycleTol [58] is greater than zero, or WdgMDeadlineReferenceCycle [59] is zero (see Error 1055 [110]).

Test 51	The following condition must be fulfilled for systems with internal software timebase source: The shortest `WdgMDeadlineMax`[69] greater zero value among all `WdgMDeadlineSupervision` elements must be greater or equal to `WdgMSupervisionCycle`[52] (see Error 1059[110]).
Test 52	The following condition must be fulfilled: `1 / WdgMTicksPerSecond`[51]`) <= WdgMSupervisionCycle`[52] (see Error 1060[110]).
Test 53	The `WdgMSupervisionCycle`[52] stored in the EDF must be greater than zero (see Error 1123[113]).
Test 54	The following condition must be fulfilled: `0 < ticks_per_second <= rti_hz / 2`.
Test 55	The targeted precision must fulfill the following condition: `int (round (ticks_per_second * window_start * 0.001)) <= 65535`. **Note:** `65535` is the maximum 16-bit integer (see Error 1075[111]).
Test 56	The targeted precision must fulfill the following condition: `int (round (ticks_per_second * condition_value * 0.001)) <= 65535`. **Note:** `65535` is the maximum 16-bit integer (see Error 1075[111]).
Test 57	Each `WdgMWatchdog` element must have a `WdgMTrigger` value associated to it (see Error 1076[111]).
Test 58	In each SE, there must be a maximum of one CP having an alive counter (see Error 1086[111]).
Test 59	Make sure that transitions reference existing CPs (see Error 1091[111]).
Test 60	Make sure that global transitions reference only existing SEs as source (see Error 1094[111]).
Test 61	Make sure that global transitions reference only existing SEs as destination (see Error 1095[111]).
Test 62	The minimum deadline of each `WdgMDeadlineSupervision` element must be less or equal to the maximum deadline (see Error 1096[111]).
Test 63	No deadline value must be greater than `(1 / tps) * MAX_16_BIT_VALUE` (see Error 1097[111]).
Test 64	The following condition must be fulfilled for configurations with an internal software tick counter source: `(1 / WdgMTicksPerSecond[Hz]) = WdgMSupervisionCycle [s]` (see Error 1098[111]).
Test 65	The trigger modes belonging to each trigger must build a zero-based list of increasing integers without a gap (see Error 1109[112]).
Test 66	Every transition must have no more than one `WdgMDeadlineSupervision` element assigned to it (see Error 1114[113]).

Test 67	The WdgMProgramFlowMonitoring boolean value of an SE must be true if and only if there are local or global transitions starting or ending in any of the CPs of that SE.
Test 87	All defined Watchdog devices in the EDF must have the same number of WdgMTrigger elements. **Note:** Not necessarily the same modes with respect to mode settings.
Test 88	The following condition must be fulfilled: (WdgMFailedProgramFlowRefCycleTol = 0) OR (WdgMProgramFlowRefCycle > 0). **Note:** Related to Error 1052⊳109.
Test 107	The WdgMTriggerTimeout field in each element in the WdgMTriggerMode array (of type WdgM_TriggerModeType) must have a value greater than zero (Error 1124⊳113).

5.1.5 Errors To Be Detected by the Verifier to Protect the Embedded Code

Test No.	Requirement
Test 6	The WdgMSupervisedEntityRef value of the main structure shall be a NULL pointer if and only if the number of SEs according to the EDF is zero.
Test 7	The EntityStatusLRef member of each SE must not be a NULL pointer.
Test 8	The EntityStatusGRef member of each SE must not be a NULL pointer.
Test 9	The WdgMAliveLRef member of each checkpoint shall be NULL_PTR if and only if the member WdgMAliveGRef in the same SE is NULL_PTR.
Test 10	The main WdgM_ConfigType structure shall have its DataGSRef member set to a non-NULL pointer.
Test 11	The main WdgM_ConfigType structure shall have its DataGRef member set to a non-NULL pointer.
Test 12	The main WdgM_ConfigType structure shall have its EntityGSRef member set to a non-NULL pointer.
Test 13	The main WdgM_ConfigType structure shall have its GlobalTransitionFlagsGS member set to NULL if and only if there are no global transitions.
Test 14	The value of WdgM_GlobalTransitionType->GlobalTransitionFlagId must match the position of the current element in the WdgM_GlobalTransitionType array.
Test 15	The EntityStatusLRef member of each SE must point to a unique variable.
Test 16	The EntityStatusGRef member of each SE must point to a unique variable.

Test 68	The CPs belonging to each SE must have IDs that build a zero-based list of increasing integers without a gap (see Error 1001 108).
Test 69	Each SE must have at least one CP (see Error 1016 108).
Test 70	There must be either a global transition or a local transition for every `WdgMDeadlineSupervision` element (see Error 1031 109).
Test 71	The ID of each SE must be unique (see Error 1062 110). **Note:** Actually superseded by handling of Error 1071 111. See below.
Test 72	Each SE must have an initial checkpoint (see Error 1064 110).
Test 73	There must be at least one callback function for the SEs or for the main structure if the flag `WDGM_STATE_CHANGE_NOTIFICATION` 47 is set to `STD_ON` (see Error 1066 110).
Test 74	The number of SEs must not be zero (see Error 1090 111).
Test 75	The `WdgM_LocalStateChangeCbk` member of each SE must point to the callback function configured for that SE according to the EDF. Otherwise this member must be `NULL_PTR` (see Error 1066 110).
Test 76	The `WdgM_GlobalStateChangeCbk` member of the main structure must be `NULL_PTR` if no callback function was configured for signaling a global state change (see Error 1066 110).
Test 77	The callback functions assigned to SEs must have a unique name (see Error 1068 110).
Test 78	CPs defined as local end CPs must not have outgoing local transitions (see Error 1069 110).
Test 79	CPs defined as local initial CPs must not have incoming local transitions (see Error 1070 110).
Test 80	The SE IDs must build a zero-based list of increasing integers without a gap (see Error 1071 111).
Test 81	If the `WdgMFailedSupervisionRefCycleTol` 57 `of an SE` is set to greater than zero, then there shall be an alive supervision counter associated to one of the CPs of that SE (see Error 1057 110).
Test 82	Each CP configured to be an SE initial CP must have CP `ID = 0`.
Test 83	The `STD_OFF` and `STD_ON` constants must be defined as zero (**0**) and one (**1**).
Test 84	The value for `WdgMTicksPerSecond` 51 must be greater than zero.

6 Abbreviations

Abbreviation	Description
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
BswM	Basic Software Module
CP	Checkpoint
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DVC	DaVinci Configurator Pro (by Vector Informatik GmbH)
ECU	Electronic Control Unit
EDF	ECU Description File
ISO	International Organization for Standardization
MCU	Microcontroller Unit
N/A	Not available
OS	Operating System
QM	Quality Managed Software (software development process)
RTE	Run-Time Environment
SCHM	Schedule Manager module (according AUTOSAR 4.0 r1)
SE	Supervised Entity
SEID	Supervised Entity Identifier
SW-C, SWC	Software Component

Abbreviation	Description
S-Wdg	Safe Watchdog Driver (implementation by TTTech)
S-WdgIf	Safe Watchdog Interface (implementation by TTTech)
S-WdgM	Safe Watchdog Manager (implementation by TTTech)
WD	Watchdog
WdgM	AUTOSAR 4.0 r1 Watchdog Manager

7 Glossary

Term	Description
Alive Indications	An indication provided by a Supervised Entity Alive counter to signal its aliveness to the S-WdgM.
Alive Monitoring	A kind of S-WdgM monitoring (supervision) that checks if a Supervised Entity is executed sufficiently often and not too often.
Checkpoint	A point in the control flow of a supervised entity where the activity is reported to the S-WdgM.
Closed Graph	A closed graph is a directed graph where every Checkpoint is reachable, starting from the local initial Checkpoint.
Configuration Tool	A tool used for creating a S-WdgM configuration, e.g, DaVinci Configurator Pro.
Container	Refers to the AUTOSAR term "container". Represents a structure with different parameters.
Deadline Monitoring	Kind of S-WdgM monitoring (supervision) that checks if the execution time between two Checkpoints is lower or higher as the configured limits.
Destination Checkpoint	End point of a transition.
End Checkpoint	The last Checkpoint that is monitored for a Supervised Entity. After passing the End Checkpoint, the S-WdgM expects that the entity is not monitored. To start the monitoring again the Initial Checkpoint must be passed first. A Supervised Entity can have zero or more End Checkpoints.
Error	Discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.
Failure	Termination of the ability of an element, to perform a function as required.
Fault	Abnormal condition that can cause an element or an item to

	fail.
Fault Detection Time	See. *S-WdgM Fault Detection Time*.
Fault Reaction Time	The Fault Reaction Time is the S-WdgM Fault Reaction Time plus the S-Wdg Fault Reaction Time.
Global Monitoring Status	Status that summarizes the Local Monitoring Status of all supervised entities.
Global Transition	A global transition is a transition between two checkpoints in the logical program flow (i.e., *source* and *destination* checkpoint), where the checkpoints belong to different supervised entities.
Initial Checkpoint	The first Checkpoint that is monitored in the Supervised Entity. The monitoring of a Supervised Entity must start at this Checkpoint. A Supervised Entity has exactly one Initial Checkpoint.
Local Monitoring Status	Status that represents the current result of supervision of a single Supervised Entity.
Local Transition	A Local Transition is the transition between two checkpoints (i.e., *source* and *destination* checkpoint) in the logical program flow in the same Supervised Entity.
Program Flow Monitoring	Kind of S-WdgM monitoring (supervision) that checks if the inspected software is executed in a predefined sequence. This sequence is defined by the user and collected in the S-WdgM configuration.
S-WdgM Fault Detection Time	The time-span from the occurrence of a fault to the detection of the fault by the S-WdgM. The detection of a fault is indicated by a change of the state `WDGM_LOCAL_STATE_OK` or `WDGM_GLOBAL_STATE_OK` to a different state. It is called *diagnostic test interval* in [6]128, part1.
S-WdgM Tick (Counter)	Tick Counter is used for deadline monitoring time measurement. Depending on the parameter `WdgMTimebaseSource` the Tick Counter is incremented by 1 for each supervision cycle or, for higher precision, with the API function `WdgM_UpdateTickCounter()` or with a hardware counter.

Safe State	The Safe State is the operating mode of an item without an unreasonable level of risk ([6]128, part1).
Safe Watchdog Manager Stack	The software module consisting of Safe Watchdog Manager, Safe Watchdog Interface and Safe Watchdog Driver.
Safe Watchdog Manager **(S-WdgM)**	The hardware-independent upper software layer of the Safe Watchdog Manager Stack.
Safe Watchdog Interface **(S-WdgIf)**	The hardware-independent middle software layer of the Safe Watchdog Manager Stack.
Safe Watchdog Driver (S-Wdg)	The hardware-dependent lowest layer of the Safe Watchdog Manager Stack. Controls the Watchdog device.
Source Checkpoint	Start point of a transition.
Supervised Entity	A software entity that is monitored by the S-WdgM. Each supervised entity has exactly one identifier. A supervised entity denotes a collection of checkpoints within a software component or basic software module. There may be zero, one or more supervised entities in a software component or basic software module. Each entity has a state that is based on the states reported from all its checkpoints. All checkpoints of one entity belong to the same memory context.
Supervision Cycle	The time period of the S-WdgM in which the cyclic supervision algorithm is performed.
Supervision Reference Cycle	The number of Supervision Cycles used as a reference by *Alive*, *Deadline* and *Program Flow Supervision* for periodic supervision. Every kind of supervision has its own reference cycle.
Timebase Tick	The S-WdgM measures the deadline of a Transition in Timebase Ticks. (In the context of this document also referred to as *S-WdgM Tick*.) **Note:** The Timebase Tick is provided either by the S-WdgM itself, or it can be provided by an external source.

Trigger Mode	The S-WdgM Trigger Mode is a set of Watchdog trigger times and Watchdog mode. One Trigger Mode is a group of the following three parameters: • `WdgMTriggerWindowStart` • `WdgMTriggerConditionValue` • `WdgMWatchdogMode` Each Watchdog device can have one or more Trigger Modes.
Watchdog Device	The Watchdog Device is the hardware part which represents the watchdog functionality. It can be an internal watchdog integrated on the MCU chip, or it can be an external watchdog device outside the MCU.

8 References

[1] AUTOSAR, *Specification of Watchdog Manager*. 080. V. 2.0.0. Rel. 4.0. Rev. 1.

[2] AUTOSAR, *Specification of Watchdog Interface*. 041. V. 2.3.0. Rel. 4.0. Rev. 1.

[3] AUTOSAR, *Specification of Watchdog Driver*. 039. V. 2.3.0. Rel. 4.0. Rev. 1.

[4] TTTech Automotive GmbH, *Safe Watchdog Interface*, User Manual. D-MSP-M-70-006.

[5] TTTech Automotive GmbH, *Safe Watchdog Manager*, Safety Manual. D-SAFEX-S-70-001.

[6] ISO 26262-2011, *Road vehicles – Functional safety*. International Standard. International Organization for Standardization (ISO), 2011.

[7] AUTOSAR, *Specification of Watchdog Manager*. 080. V. 1.2.2. Rel. 3.1. Rev. 1.

9 License Information

The S-WdgM Configuration Generator is copyright TTTech Automotive GmbH © 2011 – 2012. All rights reserved. The use of the software is subject to TTTech's Standard Software License Terms for Embedded Software and Software Tools provided together with the software. In case you don't have access to TTTech's Standard Software License Terms please contact office@tttech-automotive.com

The S-WdgM Configuration Generator was developed with the Python programming language (Copyright © 2001-2012 Python Software Foundation; All Rights Reserved) - For Python parts of the software see PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2 in the LICENSE file provided with this software.

The S-WdgM Configuration Generator includes the lxml library (Copyright © 2004 Infrae. All rights reserved) - for the lxml library see the full license text in the LICENSE file provided with this software.

Index

- A -

Abbreviations 122
Alive counter violation 27
Alive indications 26
Alive supervision 33
Aliveness 18
Aliveness parameters 19
Aliveness supervision 26
Aliveness violation 18
API description 73
  type definitions 73
Appl_Dem.h 12
Appl_Det.h 12
Appl_Mcu.h 12
Appl_Mcu_PerformReset 89
AUTOSAR 3.1 and 4.0 Compatibility 33

- C -

Checkpoint 9, 18
  destination checkpoint 21
  end checkpoint 21
  initial checkpoint 21
  local initial checkpoint 22
CheckpointID 78
Compiler.h 11
Compiler_Cfg 12
Configuration generation 102
  output files 107
  workflow 105

- D -

Deadline 15
Deadline monitoring 33
Deadline reference cycle 17
Deadline violation 17, 27
Default reset path 30
Dem_ReportErrorStatus() 12
Destination checkpoint 15
Det_ReportError() 12

- E -

End checkpoint 21
Entities, checkpoint, transitions 34
Error messages 108
  basic errors 108
  semantic errors 108

- F -

Fault detection time 27
Fault reaction time 27

- G -

Global data 97
Global memory 97
Global shared data 97
Global shared memory 97
Global state 33, 39, 40, 41, 42, 43, 44, 45
  WdgMExpiredSupervisionCycleTol 33
Global transitions 15, 22
  WdgMGlobalTransitionDestRef 22
  WdgMGlobalTransitionSourceRef 22
Glossary 124

- I -

Initial checkpoint 21
Integration 94
  deadline measurement 100
  initialization of the S-WdgM 94
  memory sections 95
  tick counter 100
  timing setup 97
Introduction 4
  architecture overview 5
  use cases 7

- L -

local end checkpoint 22
Local entity data 96
Local entity memory 96
Local reflexive transition 22
Local state 31
Local transition 15, 21

- M -

Maximum deadline
  WdgMDeadlineMax 15, 69
Maximum reaction time 29

MCU reset 30
MCU reset state 30
Mcu_PerformReset() 12
MemMap.h 12
Minimum deadline
    WdgMDeadlineMin 15, 69
Minimum reaction time 28

- N -

Node 9

- P -

PlatformTypes.h 12
Primary reset path 30
Program flow monitoring 33
Program flow reference cycle 14
Program flow violation 27
Program flow violations 14

- R -

Reference cycle 33
Reference cycle tolerance 33
Rte_Type.h 12
Rte_WdgM_Type.h 12

- S -

Safe state 27, 30
Safe Watchdog Manager 9
    alive counter options 65
    alive supervision 18
    API description 73
    basic functionality 13
    checkpoint options 65
    configuration parameters 38
    deadline monitoring 15
    deviations from AUTOSAR 4.0 r1 34
    ECU description configuration 72
    fault reaction time 27
    file structure 10
    general settings 49
    global deadline options 69
    global preprocessor settings 38
    global state 33
    global transition options 68
    global transitions 22
    local deadline options 69
    local entity state 31
    local transition options 67
    program flow supervision 13
    reset path 30
    safe state 30
    supervised entity 13
    supervised entity options 57
    supervision cycle 25
SchM_WdgM.h 12
Secondary reset path 30
SEID 78
Std_Types.h 11
Std_VersionInfoType 76
Supervised entity 6, 9, 13, 18
Supervision cycle 25
    main function 25
    WdgM_MainFunction() 25
    WdgMTriggerConditionValue 26, 77
    WdgMTriggerWindowStart 26, 77
Supervision reference cycle 31
S-WdgM 9, 122
    application level API functions 76
    callback functions 84
    expected interfaces 88
    system level API functions 84
    type definitions 73
S-WdgM supervision cycle 19, 25
S-WdgM Verifier 103

- T -

Tolerance value 31
Tolerances 36
Transition 9, 15
    global 15
    local 15
    local reflexive 22
    WdgMDeadlineStartRef 15, 70
    WdgMDeadlineStopRef 15, 70

- W -

Watchdog and Reset 36
WDGIF_MODE_OFF 40
WdgIf_Types.h 11
WdgIfDeviceRef 55
WdgInitialTimeout 99
WdgM.c 11
WdgM.h 11
WdgM_ActivateAliveSupervision(SEID) 91
WdgM_ActivateSupervisionEntity() 37, 47, 82

WdgM_Cbk_GptNotification() 93
WdgM_Cfg.h 11
WdgM_Cfg_Features.h 11, 33
WdgM_Checkpoint.c 11
WdgM_CheckpointIdType 74
WdgM_CheckpointReached() 13, 18, 78, 96
WdgM_ConfigType 73
WdgM_DeactivateAliveSupervision(SEID) 92
WdgM_DeactivateSupervisionEntity() 37, 47, 81
WdgM_DeInit() 37
WDGM_DEM_REPORT 38
WDGM_DEV_ERROR_DETECT 38
WDGM_E_CPID 78
WDGM_E_NO_INIT 78
WDGM_E_PARAM_SEID 78
WDGM_E_PARAM_STATE 78
WDGM_ENTITY_DEACTIVATION_ENABLED 47
WDGM_EXTERNAL_SOFTWARE_TICK 44
WDGM_FIRSTCYCLE_ALIVECOUNTER_RESET 48, 95
WdgM_GetAliveSupervisionStatus(SEID, *status) 91
WdgM_GetGlobalStatus() 79
WdgM_GetGlobalStatus(*status) 91
WdgM_GetLocalStatus() 79
WdgM_GetMode() 77
WdgM_GetMode(*Mode) 91
WdgM_GetVersionInfo() 92
WDGM_GLOBAL_STATE_STOPPED 43
WdgM_GlobalStatusType 75
WdgM_GssChangeCbk(status) 92
WDGM_IMMEDIATE_RESET 31, 39
WdgM_Init(&Config) 92
WdgM_Init() 95
WDGM_INTERNAL_HARDWARE_TICK 45
WDGM_INTERNAL_SOFTWARE_TICK 45
WdgM_IssChangeCbk(status) 92
WDGM_LOCAL_STATUS_DEACTIVATED 21, 82
WDGM_LOCAL_STATUS_FAILED 81
WDGM_LOCAL_STATUS_OK 81
WdgM_LocalStatusType 74
WdgM_MainFunction() 25, 27, 95, 97
WdgM_MainFunction_AliveSupervision() 93
WdgM_MemMap.h 11, 50, 96, 106
WdgM_ModeType 74
WdgM_OSMemMap.h 11, 50, 96, 106
WdgM_PBcfg.c 11
WdgM_PBcfg.h 11
WdgM_PerformReset() 80
WDGM_SECOND_RESET_PATH 30, 33, 45
WdgM_SetMode() 76
WdgM_SetMode(Mode) 90
WDGM_STATE_CHANGE_NOTIFICATION 47
WdgM_SupervisedEntityIdType 73
WDGM_TICK_OVERRUN_CORRECTION 46
WdgM_TimeBaseTickType 75
WdgM_UpdateAliveCounter(SEID) 91
WdgM_UpdateTickCounter() 99
WdgMAliveSupervisionCheckpointRef 67
WdgMAppTaskRef 64
WdgMCallerId 48
WdgMCheckpointId 65
WdgMConfig_Mode0 107
WdgMDeadlineMax 15, 69
WdgMDeadlineMin 15, 69
WdgMDeadlineReferenceCycle 18, 33, 59
WdgMDeadlineStartRef 15, 70
WdgMDeadlineStopRef 15, 70
WdgMDefensiveBehavior 41
WdgMDemReport 38
WdgMDemSupervisionReport 42
WdgMDevErrorDetect 37, 38
WdgMEnableEntityDeactivation 47, 61
WdgMEntityDeactivationEnabled 47
WdgMExpectedAliveIndications 19, 65
WdgMExpiredSupervisionCycleTol 33, 53
WdgMFailedDeadlineRefCycleTol 18, 33, 58
WdgMFailedProgramFlowRefCycleTol 14, 33, 59
WdgMFailedSupervisionRefCycleTol 33, 57
WdgMFirstCycleAliveCounterReset 18, 48
WdgMGlobalCheckpointFinalRef 53
WdgMGlobalCheckpointInitialRef 54
WdgMGlobalMemoryAppTaskRef 49
WdgMGlobalStateChangeCbk 49, 84
WdgMGlobalTransitionDestRef 22, 68
WdgMGlobalTransitionSourceRef 22, 69
WdgMImmediateReset 39
WdgMInitialTriggerModeId 50
WdgMLocalCheckpointFinalRef 63
WdgMLocalCheckpointInitialRef 64
WdgMLocalStateChangeCbk 63, 84
WdgMLocalStatusSupervisedEntityRef 61
WdgMLocalTransitionDestRef 21, 67
WdgMLocalTransitionSourceRef 21, 68
WdgMMaxMargin 19, 66
WdgMMinMargin 19, 66
WdgMModeID 50, 56
WdgMOffModeEnabled 40
WdgMProgramFlowReferenceCycle 14, 26, 33, 60
WdgMSecondResetPath 45
WdgMStateChangeNotification 47
WdgMSupervisedEntityId 61
WdgMSupervisedEntityInitialMode 58
WdgMSupervisionCycle 19, 52, 53, 99

WdgMSupervisionReferenceCycle 19, 33, 66
WdgMSupportedAutosarAPI 62, 90
WdgMTickOverrunCorrection 46
WdgMTicksPerSecond 51, 99
WdgMTimebaseSource 44
WdgMTriggerConditionValue 26, 56, 77, 99
WdgMTriggerModeId 51
WdgMTriggerWatchdogRef 57
WdgMTriggerWindowStart 26, 56, 77, 99
WdgMUseOsSuspendInterrupt 43
WdgMUseRte 42
WdgMVersionInfoApi 40
WdgMWatchdogMode 55, 77
WdgMWatchdogName 54
WdgWindowStart 99

- Z -

Zero alive indication 20