Document Title	Specification of Watchdog Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	039
Document Classification	Standard

Document Version	2.5.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
27.09.2011	2.5.0	AUTOSAR Administration	• DET-Error for Wdg_GetVersionInfo added
13.10.2010	2.4.0	AUTOSAR Administration	• Requirement WDG141/WDG143 removed
01.12.2009	2.3.0	AUTOSAR Administration	• Modifications for windowed watchdog concept • Further maintenance for R4.0: see Chapter 11 • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	Legal disclaimer revised
07.12.2007	2.2.0	AUTOSAR Administration	• Section 5.1.2 the file include structure has been changed. • Section 8.6.2 Dem_ReportErrorStatus added as optional interfaces. • Rephrased the requirementsWDG019, WDG031, WDG034. • Modified sequence diagrams in chapter 9. • Document meta information extended • Small layout adaptations made

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
31.01.2007	2.1.0	AUTOSAR Administration	• In chapter 5.1.2 the file include structure has been changed to comply with the SPAL general include structure. • In chapter WdgDefaultMode has been added as PC variant and WDG003 has been changed to allow passing NULL pointer. • For WDG037 the requirement was changed to allow configuration of activation code if the H/W allows for the same. • For WDG078 the requirement was changed to add reference to SPI/DIO for accessing the external watchdog • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
20.03.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template
31.05.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview 6

2 Acronyms and abbreviations 7

3 Related documentation 8

3.1 Input documents 8
3.2 Related standards and norms 8

4 Constraints and assumptions 9

4.1 Limitations 9
4.2 Applicability to car domains 9

5 Dependencies to other modules 10

5.1 File structure 10
5.1.1 Code file structure 10
5.1.2 Header file structure 11
5.1.3 Version check 12
5.2 System clock 13
5.3 Onboard communication handlers 13

6 Requirements traceability 14

7 Functional specification 19

7.1 General design rules 19
7.2 Error classification 20
7.3 Error detection 20
7.4 Error notification 20
7.5 External watchdog driver 21
7.6 Internal watchdog driver 21
7.7 Triggering concept to support windowed watchdogs 22
7.8 Debugging 24

8 API specification 26

8.1 Imported types 26
8.2 Type definitions 26
8.2.1 Wdg_ConfigType 26
8.3 Function definitions 26
8.3.1 Wdg_Init 26
8.3.2 Wdg_SetMode 28
8.3.3 Wdg_SetTriggerCondition 30
8.3.4 Wdg_GetVersionInfo 31
8.4 Call-back Notifications 31
8.4.1 Wdg_Cbk_GptNotification 31
8.5 Scheduled functions 32
8.6 Expected interfaces 32
8.6.1 Mandatory interfaces 33
8.6.2 Optional interfaces 33
8.6.3 Configurable interfaces 33

9 Sequence diagrams 34
9.1 Watchdog initialization, setting trigger condition and mode. 34
9.2 Data exchange between watchdog driver and hardware 35
10 Configuration specification 36
10.1 How to read this chapter 36
10.1.1 Configuration and configuration parameters 36
10.1.2 Containers 36
10.1.3 Specification template for configuration parameters 37
10.2 Containers and configuration parameters 38
10.2.1 Variants 38
10.2.2 Wdg 38
10.2.3 WdgDemEventParameterRefs 38
10.2.4 WdgGeneral 39
10.2.5 WdgSettingsConfig 41
10.2.6 WdgSettingsFast 42
10.2.7 WdgSettingsSlow 42
10.2.8 WdgSettingsOff 43
10.2.9 WdgExternalConfiguration 43
10.3 Published information 43
10.3.1 WdgPublishedInformation 44
11 Changes from Release 3.1 to Release 4.0 45
11.1 Deleted SWS Items 45
11.2 Replaced SWS Items 45
11.3 Changed SWS Items 45
11.4 Added SWS Items 46
12 Not applicable requirements 47

1 Introduction and functional overview

This document specifies the functionality, API and the configuration of the AUTOSAR Basic Software module watchdog driver (Wdg).

This module provides services for initialization, changing the operation mode and setting the trigger condition (timeout).

The functional requirements and the functional scope are the same for both internal and external watchdog drivers. Hence the API is semantically identical.

An internal watchdog driver belongs to the Microcontroller Abstraction Layer (MCAL), whereas an external watchdog driver belongs to the Onboard Device Abstraction Layer. Therefore, an external watchdog driver needs other drivers (in MCAL) in order to access the microcontroller hardware.

2 Acronyms and abbreviations

Acronyms and abbreviations that have a local scope are not contained in the AUTOSAR glossary. These must appear in a local glossary.

Abbreviation / Acronym:	Description:
DIP	Digital Input/Output
DET	Development Error Tracer – module to catch development errors.
DEM	Diagnostic Event Manager – module to handle diagnostic relevant events.
GPT	General Purpose Timer
SPI	Serial Peripheral Interface
WDG	Watchdog (module specific prefix)

Definitions needed for understanding of the concepts

Definition:	Description:
Off-Mode	The watchdog hardware is disabled / shut down. This might be necessary in order to shut down the complete ECU and not get cyclic resets from a still running external watchdog. This mode might not be allowed for safety critical systems. In this case, the Wdg module has to be configured to prevent switching to this mode.
Slow-Mode	Triggering the watchdog hardware can be done with a long timeout period. This mode can e.g. be used during system startup / initialization phase. E.g. the watchdog hardware is configured for toggle mode (no constraints on the point in time at which the triggering is done) and a timeout period of 20 milliseconds.
Fast-Mode	Triggering the watchdog hardware has to be done with a short timeout period. This mode can e.g. be used during normal operations of the ECU. E.g. the watchdog hardware is configured for window mode (triggering the watchdog has to occur within certain minimum / maximum boundaries within the timeout period) and a timeout period of 5 milliseconds.

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[3] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

[4] Requirements on Watchdog Driver
AUTOSAR_SRS_WatchdogDriver.pdf

[5] Specification of Watchdog Interface
AUTOSAR_SWS_WatchdogInterface.pdf

[6] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[7] Specification of Watchdog Driver
AUTOSAR_SWS_GPTDriver.pdf

[8] Specification of Watchdog Driver
AUTOSAR_SWS_RTE.pdf

[9] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

3.2 Related standards and norms

None

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

A Wdg module for an internal (on-chip) watchdog accesses the microcontroller hardware directly and is located in the Microcontroller Abstraction layer.

A Wdg module for an external watchdog uses other modules (e.g. SPI) to access the external watchdog device. Such a Wdg module is located in the Onboard Device Abstraction Layer (see [1]).

[WDG055] ⌈The Wdg module for an external watchdog driver shall have source code that is independent of the microcontroller platform.⌋ ()

5.1 File structure

5.1.1 Code file structure

[WDG079] ⌈The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files (as far as required; for name expansion see WDG169):

- Wdg_Lcfg.c – for link time configurable parameters
- Wdg_PBcfg.c – for post build time configurable parameters
- Wdg_Irq.c – for holding the interrupt frames in case an internal watchdog servicing is implemented as interrupt routine (and not via timer callback)

These files shall contain all link time and post-build time configurable parameters.⌋ (BSW00380, BSW00346, BSW158, BSW00314, BSW12263)

Note: These names are required by BSW00314 and BSW00346

[WDG169] ⌈If more than one watchdog driver instance exists on an ECU (namely an external and an internal one) the implementer shall provide unique code file names by expanding the names according to BSW00347.⌋ (BSW00347)

5.1.2 Header file structure

[WDG061] ⌈The Wdg module shall adhere to the following file structure:

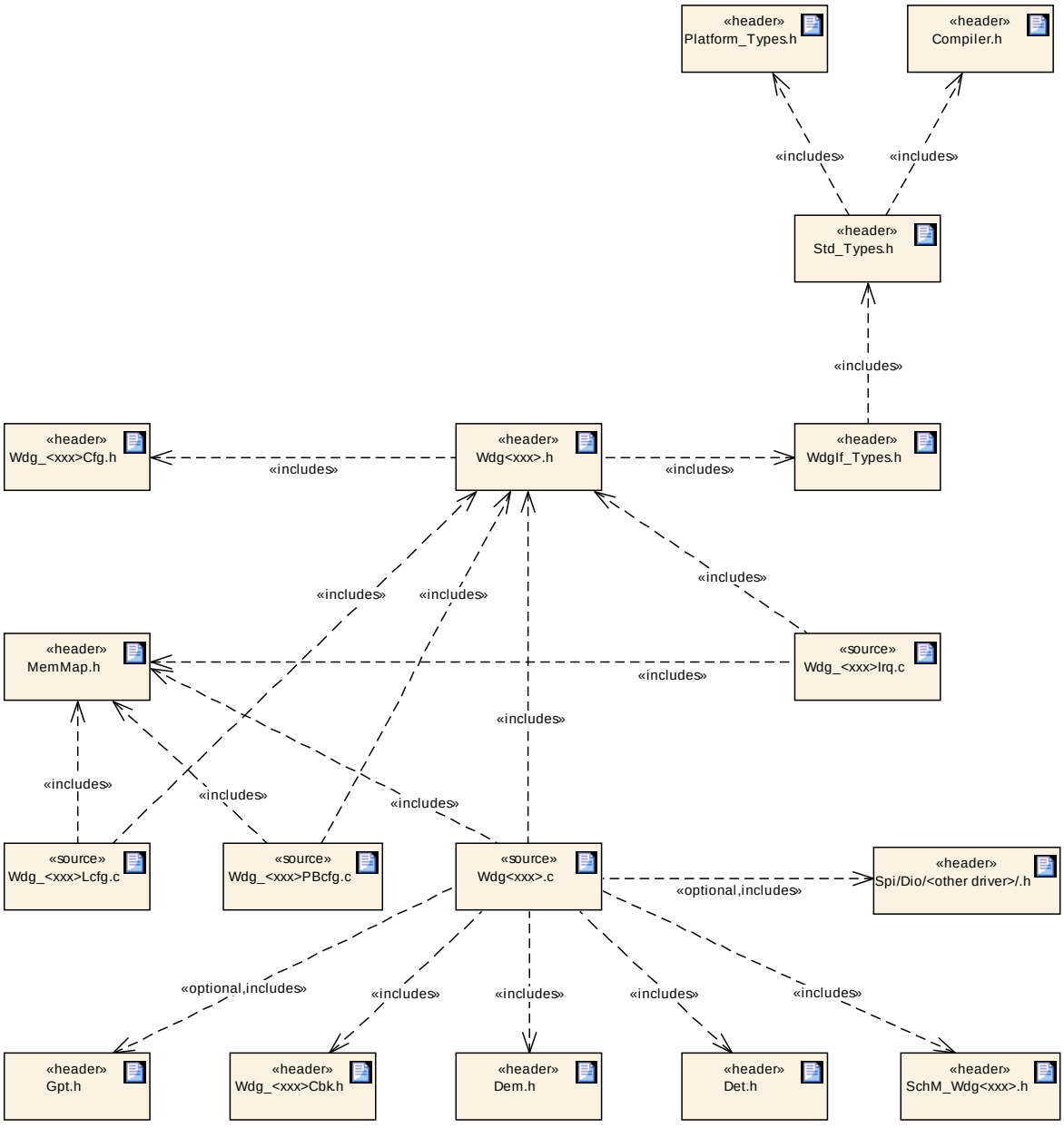

Figure 1: File include structure

⌋ (BSW00345, BSW159, BSW00381, BSW00412, BSW00346, BSW158, BSW00370, BSW00435, BSW00436, BSW00301)

Notes to the figure:

- The possible name expansion for multiple driver modules are indicated as <xxx> in Figure 1.
- Since the API names are also to be expanded, the header Wdg<xxx>.h will become instance specific.

- Wdg<xxx>.h contains the pre-compile configuration macros. It is not expected that, source code is required to implement the pre-compile configuration for the watchdog driver module. This file also contains – if required – references to the c-data for link-time and/or post-build configuration.
- Wdg_<xxx>Cbk.h contains the declaration of callback functions from other modules implemented by the Wdg module (see BSW00370). This file is mandatory, even if there are no callback functions required.
- SchM_Wdg<xxx>.h is a mandatory include file (see BSW00335) provided by the RTE generator (see [8]). Though the Watchdog Driver has no scheduled function, this header is e.g. needed, if the Wdg module defines critical section.
- The need to include headers from GPT-, SPI-, DIO- or other drivers depends on how the watchdog is serviced and the watchdog hardware is accessed, see chapters 7.5 and 7.6.

[WDG170] ⌈If more than one watchdog driver instance exists on an ECU (namely an external and an internal one) the implementer shall provide unique header file names by expanding the names according to BSW00347.⌋ (BSW00347)

[WDG080] ⌈The Wdg module shall include the `Dem.h` file for any production errors reported during implementation.

By this inclusion, the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.⌋ ()

Note:
In case of multiple watchdog driver instances, the Event Id symbols for production errors defined in this specification (see WDG010 and WDG148_Conf) might be expanded in the configuration of the DEM in order to make them unique.

5.1.3 Version check

[WDG087] ⌈The Wdg module shall avoid the integration of incompatible files by the following pre-processor checks:

For included (external) header files:

- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION

shall be verified.

If the values are not identical to the values expected by the Wdg module, an error shall be reported.⌋ (BSW167, BSW004)

5.2 System clock

If the hardware of the internal watchdog depends on the system clock, changes to the system clock (e.g. PLL on → PLL off) may also affect the clock settings of the watchdog hardware.

5.3 Onboard communication handlers

A Wdg module for an external watchdog device depends on the API and capabilities of the used onboard communication handlers or drivers (e.g. SPI handler).

6 Requirements traceability

Requirement	Satisfied by
-	WDG034
-	WDG154
-	WDG109
-	WDG153
-	WDG139
-	WDG151
-	WDG107
-	WDG165
-	WDG164
-	WDG068
-	WDG067
-	WDG173
-	WDG174
-	WDG145
-	WDG140
-	WDG111
-	WDG080
-	WDG168
-	WDG136
-	WDG110
-	WDG152
-	WDG148
-	WDG103
-	WDG105
-	WDG063
-	WDG161
-	WDG099
-	WDG146
-	WDG149
-	WDG138
-	WDG055
-	WDG150
BSW00301	WDG061
BSW00302	WDG175
BSW00304	WDG175
BSW00306	WDG175
BSW00307	WDG175
BSW00308	WDG175

BSW00309	WDG175
BSW00312	WDG175
BSW00314	WDG079
BSW00321	WDG175
BSW00323	WDG090, WDG091, WDG092, WDG089, WDG026, WDG025
BSW00325	WDG166
BSW00326	WDG166
BSW00327	WDG013, WDG010
BSW00328	WDG175
BSW00330	WDG175
BSW00331	WDG013, WDG010
BSW00333	WDG175
BSW00334	WDG175
BSW00335	WDG019, WDG018, WDG017
BSW00336	WDG031
BSW00337	WDG013, WDG010
BSW00338	WDG052, WDG090, WDG018, WDG091, WDG017, WDG092, WDG089, WDG035, WDG026, WDG025
BSW00339	WDG175
BSW00341	WDG175
BSW00343	WDG155
BSW00344	WDG175
BSW00345	WDG061
BSW00346	WDG079, WDG061
BSW00347	WDG169, WDG172, WDG170
BSW00348	WDG175
BSW00350	WDG045
BSW00353	WDG175
BSW00355	WDG175
BSW00358	WDG106
BSW00359	WDG175
BSW00360	WDG175
BSW00361	WDG175
BSW00369	WDG066
BSW00370	WDG061
BSW00371	WDG175
BSW00373	WDG175
BSW00375	WDG175
BSW00376	WDG175
BSW00377	WDG175
BSW00378	WDG175
BSW00380	WDG079

BSW00381	WDG061
BSW00383	WDG175
BSW00385	WDG013, WDG010
BSW00386	WDG045, WDG065, WDG064
BSW00387	WDG163
BSW004	WDG087, WDG086
BSW00400	WDG001
BSW00401	WDG175
BSW00404	WDG175
BSW00405	WDG175
BSW00406	WDG019
BSW00409	WDG062
BSW00410	WDG175
BSW00412	WDG061
BSW00413	WDG175
BSW00414	WDG171, WDG106
BSW00415	WDG175
BSW00416	WDG175
BSW00417	WDG175
BSW00419	WDG175
BSW00422	WDG175
BSW00423	WDG175
BSW00424	WDG175
BSW00425	WDG175
BSW00426	WDG040
BSW00427	WDG166
BSW00428	WDG175
BSW00429	WDG040
BSW00432	WDG175
BSW00433	WDG175
BSW00434	WDG175
BSW00435	WDG061
BSW00436	WDG061
BSW00437	WDG175
BSW00439	WDG166
BSW00440	WDG175
BSW00441	WDG175
BSW00443	WDG175
BSW00444	WDG175
BSW00445	WDG175
BSW00446	WDG175
BSW00447	WDG175

BSW00449	WDG175
BSW00450	WDG175
BSW005	WDG175
BSW006	WDG175
BSW007	WDG175
BSW009	WDG175
BSW010	WDG175
BSW101	WDG001
BSW12015	WDG051, WDG160
BSW12018	WDG160
BSW12019	WDG094, WDG093, WDG095, WDG162, WDG166, WDG134, WDG135, WDG144
BSW12056	WDG175
BSW12057	WDG100, WDG101
BSW12063	WDG175
BSW12064	WDG016, WDG017
BSW12067	WDG175
BSW12068	WDG175
BSW12069	WDG175
BSW12075	WDG175
BSW12077	WDG175
BSW12078	WDG175
BSW12092	WDG076
BSW12105	WDG001, WDG100, WDG101
BSW12106	WDG026, WDG025
BSW12125	WDG100, WDG101
BSW12129	WDG166
BSW12155	WDG175
BSW12163	WDG031, WDG026, WDG025
BSW12165	WDG077
BSW12166	WDG078
BSW12167	WDG175
BSW12168	WDG175
BSW12263	WDG079
BSW12265	WDG175
BSW12267	WDG175
BSW12448	WDG090, WDG091, WDG017, WDG092, WDG089
BSW12461	WDG100, WDG101
BSW12462	WDG175
BSW12463	WDG175
BSW157	WDG175
BSW158	WDG079, WDG061
BSW159	WDG061

BSW161	WDG175
BSW162	WDG175
BSW164	WDG166
BSW167	WDG087, WDG086
BSW168	WDG175
BSW170	WDG175
BSW172	WDG175

7 Functional specification

7.1 General design rules

[WDG086] ⌈The Wdg module shall statically check the configuration parameters (at the latest during compile time) for correctness.⌋ (BSW167, BSW004)

[WDG031] ⌈The Wdg module shall not implement an interface for de-initialization/shutdown. If the watchdog supports a de-initialization/shutdown and the environment allows the usage of this feature, the de-initialization/shutdown shall be achieved by calling the `Wdg_SetMode` routine with OFF mode parameter.⌋ (BSW00336, BSW12163)

Rationale: Some watchdogs do not support the de-initialization/shutdown functionality and in some environments this feature must not be used (e.g. in safety critical systems).

[WDG034] ⌈The start address of the watchdog trigger routine shall be statically configurable to a fixed memory location by the user. The user needs to take care that Configured memory location is valid for the platform on which driver is being implemented on.This configuration parameter shall only be given if supported/needed by the hardware.⌋ ()

Rationale: This allows the watchdog device to identify the correct trigger input if supported by the hardware.

[WDG040] ⌈If interrupts have to be disabled in order to ensure data consistency or correct functionality of this module (e.g. while switching the watchdog mode or during the watchdog trigger routine), this shall be done by using the corresponding BSW Scheduler functionality if possible (this means definition of an exclusive area). The internal watchdog driver (because it belongs to MCAL) may also directly disable interrupts – see BSW00429.⌋ (BSW00426, BSW00429)

[WDG168] ⌈Depending on a static configuration (see WDG147_Conf), the code of the Wdg module is executed either from ROM or from RAM.⌋ ()

Motivation: For certain use cases, e.g. for flash programming in bootloader mode, the watchdog module has to be part of an executable which runs in RAM.

Hint: This is more a requirement for the build environment than for the watchdog module itself. However, since it might also influence the implementation of the code, it is stated here and a corresponding configuration parameter is given.

7.2 Error classification

[WDG062] ⌈The Wdg module shall take the values for production code Event Ids from the file Dem_IntErrId.h which is included via Dem.h.⌋ (BSW00409)

[WDG063] ⌈Development error values are of type uint8.⌋ ()

[WDG010] ⌈The Wdg module shall detect the following errors and exceptions depending on its configuration (development/production mode):

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service used in wrong context (e.g. module not initialized).	Development	`WDG_E_DRIVER_STATE`	0x10
API service called with wrong / inconsistent parameter(s)	Development	`WDG_E_PARAM_MODE` `WDG_E_PARAM_CONFIG`	0x11 0x12
The passed timeout value is higher than the maximum timeout value	Development	`WDG_E_PARAM_TIMEOUT`	0x13
API is called with wrong pointer value (e.g. NULL pointer)	Development	`WDG_E_PARAM_POINTER`	0x14
Setting a watchdog mode failed (during initialization or mode switch).	Production	`WDG_E_MODE_FAILED`	Assigned by DEM
Initialization or watchdog mode switch failed because it would disable the watchdog though this is not allowed in this configuration	Production	`WDG_E_DISABLE_REJECTED`	Assigned by DEM

⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

7.3 Error detection

[WDG045] ⌈The detection of development errors is configurable by the parameter `WdgDevErrorDetect` (see chapter **Fehler! Verweisquelle konnte nicht gefunden werden.**) .⌋ (BSW00386, BSW00350)

[WDG064] ⌈If the `WdgDevErrorDetect` switch is enabled, API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.2 and chapter 8.⌋ (BSW00386)

[WDG065] ⌈The detection of production code errors cannot be switched **off.**⌋ (BSW00386)

7.4 Error notification

[WDG066] ⌈Detected evelopment errors shall be reported to the Development Error Tracer (DET) if the pre-processor switch `WdgDevErrorDetect` is set. The error codes shall not be used as return values of the called function.⌋ (BSW00369)

[WDG013] ⌈Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the module's implementation documentation. The classification and enumeration shall be compatible to the errors listed above [WDG010].⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

7.5 External watchdog driver

[WDG076] ⌈To access the external watchdog hardware, the corresponding Wdg module instance shall use the functionality and API of the corresponding handler or driver, e.g. the SPI handler or DIO driver.⌋ (BSW12092)

[WDG162] ⌈The routine servicing an external watchdog shall be implemented as a GPT driver callback (see [7] for the GPT driver and chapter 8.4.1 for details of the callback).⌋ (BSW12019)

Hint: An external watchdog driver is part of the Onboard Device Abstraction Layer (see [1]), which excludes direct hardware access.

[WDG077] ⌈A Wdg module for an external watchdog shall satisfy the same functional requirements and offer the same functional scope as a Wdg module for an internal watchdog. Hence their respective APIs are semantically identical.⌋ (BSW12165)

[WDG078] ⌈The Wdg module shall add all parameters required for accessing the external watchdog hardware, e.g. the used SPI channel or DIO port, to the module's published parameters and to the module's configuration parameters.⌋ (BSW12166)

7.6 Internal watchdog driver

[WDG161] ⌈To access the internal watchdog hardware, the corresponding Wdg module instance shall access the hardware for watchdog servicing directly.⌋ ()

Hint: An internal watchdog driver is part of the Microcontroller Abstraction Layer (see [1]), which allows direct hardware access.

[WDG166] ⌈The routine servicing an internal watchdog shall be implemented either as an interrupt routine driven by a hardware timer or as a GPT driver callback (see [7] for the GPT driver and chapter 8.4.1 for details of the callback).⌋ (BSW00427, BSW164, BSW00325, BSW00326, BSW00439, BSW12129, BSW12019)

Notes:
In both cases, the watchdog servicing routine runs in interrupt context.

If the watchdog servicing routine is implemented as an interrupt routine (i.e. as a cat1 or cat2 interrupt routine and not via the GPT), it shall be described in the Basic

Software Module Description and the implementation shall follow the requirements for interrupt handling as given by [2] and [2] [3] (BSW00427, BSW00325, BSW00326, BSW00439, BSW00314, BSW00429, BSW12129).

7.7 Triggering concept to support windowed watchdogs

In former versions of this specification, the watchdog servicing routine was called from an upper layer of the software which made it difficult to guarantee timing constraints namely for windowed watchdog conditions. This concept has been changed leading to the requirements explained in this chapter.

The basic idea of this concept is to decouple the timing for servicing the watchdog hardware from the logical control.

As already stated by WDG162 and WDG166, the time base for triggering the watchdog shall be provided by means of a hardware timer or via the GPT driver. This ensures minimum timing jitter.

These two requirements WDG162 and WDG166 also imply that servicing of the watchdog hardware is done directly from a timer ISR. This ensures minimum latencies.

These two conditions – minimum jitter and latencies - ensure that the time window of a windowed watchdog can be met.

The Wdg Driver expects, that the logical control of the watchdog (whether the watchdog shall be triggered or not) shall be the responsibility of the environment, e.g. the Wdg Manager, so that the basic concepts of the Wdg Manager (alive supervision) shall remain unchanged.

[WDG144] ⌈The Wdg Manager (or other entities) shall control the watchdog driver via a so called trigger condition: as long as the trigger condition is valid the Wdg Driver services the watchdog hardware, if the trigger condition becomes invalid the Wdg Driver stops triggering and the watchdog expires.
The semantics of the trigger condition can be interpreted as a “permission to service the watchdog for the next n milliseconds”. Within this time frame the trigger condition has to be updated by the controlling entity else the watchdog will expire.
Handover of the watchdog control logic is simply done by shared usage of the trigger condition (e.g. during startup / shutdown).⌋ (BSW12019)

[WDG134] ⌈If the trigger counter is greater than zero, the watchdog servicing routine shall decrement the trigger counter and trigger the hardware watchdog.⌋ (BSW12019)

[WDG135] ⌈If the trigger counter has reached zero, the watchdog servicing routine shall do nothing (i.e. the watchdog is not triggered and will therefore expire).⌋ (BSW12019)

[WDG093] ⌈If the watchdog hardware requires an activation code which can be configured or changed, the Wdg Driver shall handle the activation code internally. In this case, the Wdg Driver shall pass the correct activation code to the watchdog hardware and the watchdog hardware in turn shall update the Wdg module's internal variable where the next expected access code is stored.⌋ (BSW12019)

[WDG094] ⌈If the watchdog hardware requires an activation code which can be configured or changed, the trigger cycle of the Wdg Driver shall be defined with a value so that updating the activation code by the watchdog hardware can be guaranteed (see

The diagram shows the sequence to trigger the watchdog hardware. Note that this is only an example. Depending on the implementation, the interrupt routine for servicing the watchdog could also come via a GPT Driver callback. For an external watchdog, the watchdog hardware cannot be accessed directly, but only via drivers of the MCAL layer, like SPI or DIO.

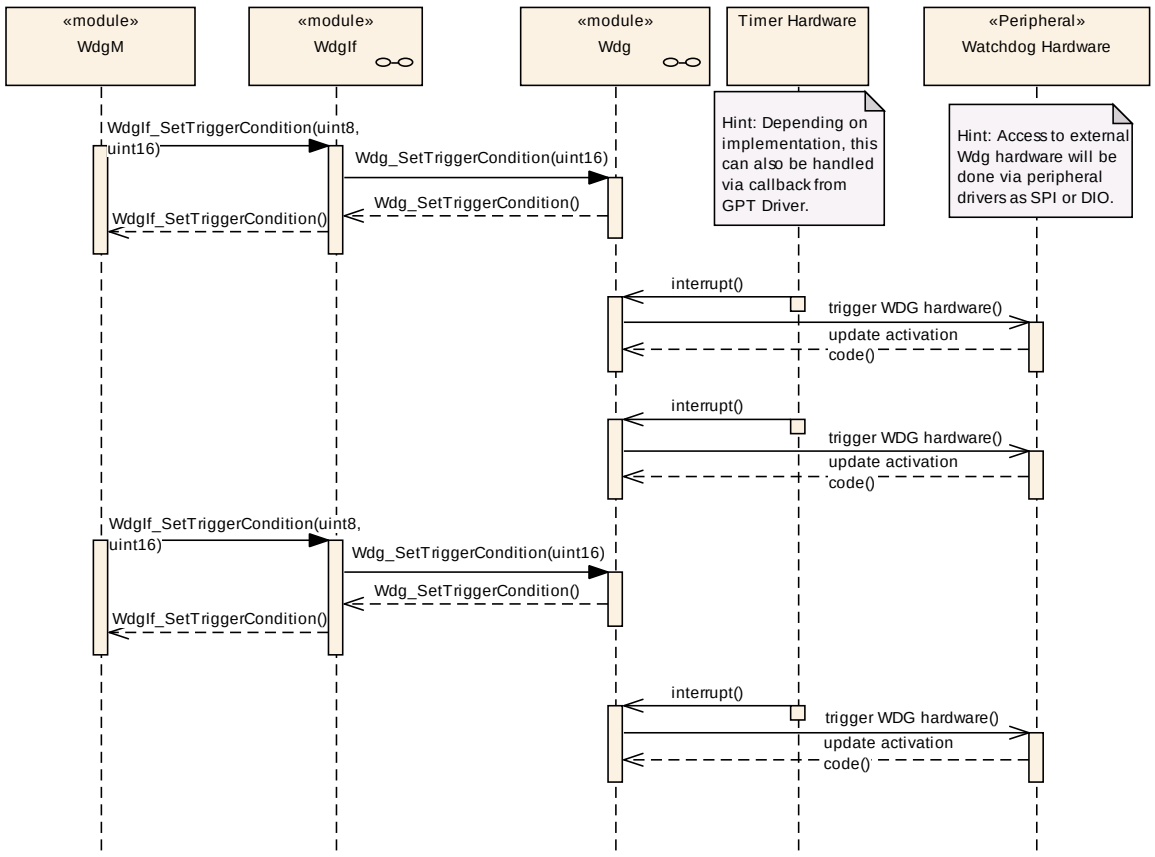

Figure 3).

⌋ (BSW12019)

[WDG095] ⌈If the watchdog hardware requires an activation code which can be configured or changed and the initial activation code can be configured, the activation

code shall be provided in the Wdg Driver's configuration set. If the activation code is fixed for a particular hardware the above requirement can be ignored.」(BSW12019)

[WDG035] 「When development error detection is enabled for the Wdg Driver module: the watchdog servicing routine shall check whether the Wdg module's state is `WDG_IDLE` (meaning the watchdog driver and hardware are initialized and the watchdog is currently not being triggered or switched). If this is not the case, the function shall not trigger the watchdog hardware but raise the development error `WDG_E_DRIVER_STATE`.」(BSW00338)

[WDG052] 「When development error detection is enabled for the Wdg Driver module: the watchdog servicing routine shall set the Wdg module's state to `WDG_BUSY` during its execution (indicating, that the module is busy) and shall reset the module's state to `WDG_IDLE` (indicating, that the module is initialized and not busy) as last operation before it returns.」(BSW00338)

Note: This specification prescribes the symbols `WDG_IDLE` and `WDG_BUSY` only, if they are externally visible, e.g. for debugging (see BSW00335). Choosing the data type for the status variable is up to the implementation.

Hint for the integration: The Wdg module's environment shall make sure that the Wdg Driver module has been initialized before watchdog servicing routine is called.

7.8 Debugging

[WDG148] 「Each variable that shall be accessible by AUTOSAR Debugging shall be defined as global variable.」()

[WDG149] 「All type definitions of variables which shall be debugged shall be accessible by the header file Wdg.h.」()

[WDG150] 「The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"`sizeof`".」()

[WDG151] 「Variables available for debugging shall be described in the respective Basic Software Module Description.」()

[WDG152] 「The internal state of the module (which indicates whether it is not initialized, idle or busy) shall be available for debugging.」()

[WDG153] 「The internal variable for the watchdog timeout counter shall be available for debugging.」()

[WDG154] ⌈The internal variable for the watchdog mode shall be available for debugging.⌋ ()

8 API specification

[WDG172] ⌈If more than one watchdog driver instance exits on an ECU (namely an external and an internal one) the API names and instance specific type names specified in this chapter shall be made unique by expansion according to BSW00347.
⌋(BSW00347)

8.1 Imported types

In this chapter all types included from the following files are listed:

[WDG105] ⌈

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
Gpt	Gpt_ChannelType
	Gpt_ValueType
Std_Types	Std_ReturnType
	Std_VersionInfoType
WdgIf	WdgIf_ModeType

The types from Gpt_Types.h are needed optionally, depending on whether and how a GPT Driver callback is implemented (see also chapter 8.6.2).⌋()

8.2 Type definitions

8.2.1 Wdg_ConfigType

[WDG171] ⌈

Name:	`Wdg_ConfigType`	
Type:	`Structure`	
Range:	`Hardware dependent structure`	Structure to hold the watchdog driver configuration set.
Description:	Used for pointers to structures holding configuration data provided to the Wdg module initialization routine for configuration of the module and watchdog hardware.	

⌋(BSW00414)

8.3 Function definitions

8.3.1 Wdg_Init

[WDG106] ⌈

Service name:	Wdg_Init	
Syntax:	`void Wdg_Init( const Wdg_ConfigType* ConfigPtr )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to configuration set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the module.	

⌋ (BSW00358, BSW00414)

[WDG001] ⌈The `Wdg_Init` function shall initialize the Wdg module and the watchdog hardware, i.e. it shall set the default watchdog mode and timeout period as provided in the configuration set.⌋ (BSW00400, BSW101, BSW12105)

Note:
Via post-build configuration, the user can choose the configuration set to be used with the `Wdg_Init` function from a limited number of statically configured sets (see also BSW00314).

[WDG100] ⌈The `Wdg_Init` function shall initialize all global variables of the Wdg module and set the default watchdog mode and initial timeout period⌋ (BSW12057, BSW12125, BSW12461, BSW12105)

[WDG101] ⌈The `Wdg_Init` function shall initialize those controller registers that are needed for controlling the watchdog hardware and that do not influence/depend on other (hardware) modules.

Registers that can influence or depend on other modules are initialized by a common system module.⌋ (BSW12057, BSW12125, BSW12461, BSW12105)

[WDG025] ⌈If disabling the watchdog is not allowed (because pre-compile configuration parameter `WdgDisableAllowed==OFF`) and if the default mode given in the provided configuration set disables the watchdog, the `Wdg_Init` function shall not execute the initialization but raise the production error `WDG_E_DISABLE_REJECTED`.⌋ (BSW00338, BSW00323, BSW12163, BSW12106)

[WDG173] ⌈If switching the Wdg module and the watchdog hardware into the default mode is not possible, e.g. because of inconsistent mode settings or because some timing constraints have not been met, the `Wdg_Init` function shall raise the production error `WDG_E_MODE_FAILED`.⌋ ()

[WDG089] ⌈When development error detection is enabled for the Wdg module: The function `Wdg_Init` shall check that the parameter `ConfigPtr` is not NULL (except for the Pre-Compiled variant). If this error is detected, the function Wdg_Init shall not

execute the initialization but raise the development error WDG_E_PARAM_POINTER.」(BSW00338, BSW00323, BSW12448)

[WDG090] 「When development error detection is enabled for the Wdg module: The Wdg_Init function shall check that the (hardware specific) contents of the given configuration set is within the allowed boundaries. If this error is detected, the function Wdg_Init shall not execute the initialization but raise the development error WDG_E_PARAM_CONFIG.」(BSW00338, BSW00323, BSW12448)

[WDG019] 「When development error detection is enabled for the Wdg module: The Wdg_Init function shall set the Wdg module's internal state from WDG_UNINIT (the default state indicating a non-initialized module) to WDG_IDLE if the initialization was successful.」(BSW00406, BSW00335)

Note: This specification prescribes the symbols WDG_IDLE and WDG_UNINIT only, if they are externally visible, e.g. for debugging (see BSW00335). Choosing the data type for the status variable is up to the implementation.

8.3.2 Wdg_SetMode

[WDG107] 「

Service name:	Wdg_SetMode	
Syntax:	`Std_ReturnType Wdg_SetMode( WdgIf_ModeType Mode )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Mode	One of the following statically configured modes: 1. WDGIF_OFF_MODE 2. WDGIF_SLOW_MODE 3. WDGIF_FAST_MODE
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	Std_ReturnType.
Description:	Switches the watchdog into the mode Mode.	

」()

[WDG160] 「The function Wdg_SetMode shall switch the watchdog driver from the current watchdog mode into the mode given by the argument Mode. This means: By choosing one of a limited number of statically configured settings (e.g. toggle or window watchdog, different timeout periods) the Wdg module and the watchdog hardware are switched to one of the following three different modes:

- WDGIF_OFF_MODE
- WDGIF_SLOW_MODE
- WDGIF_FAST_MODE 」(BSW12015, BSW12018)

[WDG051] ⌈The configuration set provided to the Wdg module's initialization routine shall contain the hardware / driver specific parameters to be used in the different watchdog modes.⌋ (BSW12015)

Hint: This may, for example, include mode-dependent settings for the GPT timer, if GPT callbacks are used.

[WDG145] ⌈The `Wdg_SetMode` function shall reset the watchdog timeout counter based on the new watchdog mode i.e. the timeout frame remaining shall be recalculated based on a changed trigger period.⌋ ()

[WDG103] ⌈The `Wdg_SetMode` function shall return `E_OK` if the mode switch has been executed completely and successfully, i.e. all parameters of the Wdg module and the watchdog hardware have been set to the new values⌋ ()

[WDG016] ⌈If switching the Wdg module and the watchdog hardware into the requested mode is not possible, e.g. because of inconsistent mode settings or because some timing constraints have not been met, the `Wdg_SetMode` function shall return the value `E_NOT_OK` and raise the production error `WDG_E_MODE_FAILED`.⌋ (BSW12064)

[WDG026] ⌈If disabling the watchdog is not allowed (e.g. in safety relevant systems, see WDG115_Conf) the `Wdg_SetMode` function shall check whether the settings for the requested mode would disable the watchdog. In this case, the function shall not execute the mode switch but raise the production error `WDG_E_DISABLE_REJECTED` and return with the value `E_NOT_OK`.⌋ (BSW00338, BSW00323, BSW12163, BSW12106)

[WDG091] ⌈When development error detection is enabled for the Wdg module: The `Wdg_SetMode` function shall check that the parameter Mode is within the allowed range. If this is not the case, the function shall not execute the mode switch but raise development error `WDG_E_PARAM_MODE` and return with the value `E_NOT_OK`⌋ (BSW00338, BSW00323, BSW12448)

[WDG092] ⌈When development error detection is enabled for the Wdg module: The `Wdg_SetMode` function shall check that the (hardware specific) settings for the requested mode are within the allowed boundaries. If this is not the case, the function shall not execute the mode switch but raise the development error `WDG_E_PARAM_MODE` and return with the value `E_NOT_OK`. ⌋ (BSW00338, BSW00323, BSW12448)

[WDG017] ⌈When development error detection is enabled for the Wdg module: The `Wdg_SetMode` function shall check that the Wdg module's state is `WDG_IDLE` (meaning the Wdg module and the watchdog hardware are initialized and the watchdog is currently not being triggered or switched). If this is not the case, the function shall not execute the mode switch but raise the development error `WDG_E_DRIVER_STATE` and return with the value `E_NOT_OK`. ⌋ (BSW00338, BSW00335, BSW12064, BSW12448)

[WDG018] ⌈When development error detection is enabled for the Wdg module: The function `Wdg_SetMode` shall set the Wdg module's state to `WDG_BUSY` during its execution (indicating, that the module is busy) and shall reset the Wdg module's state to `WDG_IDLE` as last operation before it returns to the caller. ⌋ (BSW00338, BSW00335)

Note: This specification prescribes the symbols `WDG_IDLE` and `WDG_BUSY` only, if they are externally visible, e.g. for debugging (see BSW00335). Choosing the data type for the status variable is up to the implementation.

8.3.3 Wdg_SetTriggerCondition

[WDG155] ⌈

Service name:	Wdg_SetTriggerCondition

Syntax:	`void Wdg_SetTriggerCondition(`
	`  uint16 timeout`
	`)`

Service ID[hex]:	0x03
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	timeout — Timeout value (milliseconds) for setting the trigger counter.
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Sets the timeout value for the trigger counter.

⌋ (BSW00343)

[WDG136] ⌈The function `Wdg_SetTriggerCondition` shall reset the watchdog timeout counter according to the timeout value passed.⌋ ()

[WDG138] ⌈The timeout value passed shall be interpreted as 'milliseconds'. The conversion from milliseconds to the corresponding counter value shall be done internally by the Wdg module.⌋ ()

[WDG139] ⌈The current watchdog mode shall be taken into account when calculating the counter value from the timeout parameter.⌋ ()

[WDG140] ⌈This function shall also allow to set "0" as the time frame for triggering which will result in an (almost) immediate stop of the watchdog triggering and an (almost) instantaneous watchdog reset of the ECU.⌋ ()

[WDG146] ⌈When development error detection is enabled for the module: The function `Wdg_SetTriggerCondition` shall check that the timeout parameter given is less or equal to the maximum timeout value (`WdgMaxTimeout`). If this is not the

case the function shall not reload the timeout counter but raise the development error WDG_E_PARAM_TIMEOUT and return to the caller.」()

8.3.4 Wdg_GetVersionInfo

[WDG109] 「

Service name:	Wdg_GetVersionInfo	
Syntax:	`void Wdg_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information of the module.	

」()

[WDG067] 「The Wdg_GetVersionInfo function shall return the version information of this module. The version information includes:

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

[WDG068] 「The Wdg_GetVersionInfo function shall be pre-compile time configurable On/Off by the configuration parameter WdgVersionInfoApi.」()

[WDG099] 「If source code for caller and callee of the Wdg_GetVersionInfo function is available, the module Wdg should realize this function as a macro, defined in the module's header file.」()

[WDG174] 「If DET is enabled for the Wdg Driver module, the function Wdg_GetVersionInfo shall raise WDG_E_PARAM_POINTER, if the argument is a NULL pointer and return without any action.」()

8.4 Call-back Notifications

This chapter lists all functions provided by the Wdg module to lower layer modules.

8.4.1 Wdg_Cbk_GptNotification

[WDG163] ⌈In case GPT callbacks are used for servicing the hardware watchdog, the Wdg module shall implement one or more callback entries of the following form (see [7]):

Service name:	Wdg_Cbk_GptNotification<xxx>
Syntax:	`void Wdg_Cbk_GptNotification<xxx>( void )`
Service ID[hex]:	0x05
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Watchdog triggering routine in case it is called from a GPT callback.

⌋ (BSW00387)

The number of such callback routines and the associated `GptChannelConfiguration(s)` (which also resolve the <xxx> of the name given above) depend on the hardware properties, especially on the hardware specific implementation of the different watchdog modes. Thus it cannot be standardized and the following holds:

[WDG164] ⌈If GPT callbacks are used for servicing the hardware watchdog, the required `GptChannelConfiguration(s)` shall be referenced by vendor specific configuration parameters. These parameters shall be added to the appropriate sub-container(s) below `WdgSettingsConfig` as part of a vendor specific extension.⌋ ()

[WDG165] ⌈The content of the required `GptChannelConfiguration(s)` shall be restricted, if appropriate, by preconfigured configuration values.⌋ ()

8.5 Scheduled functions

This chapter lists all functions provided by the Wdg module and called directly by the Basic Software Module Scheduler.

The Wdg module has no scheduled functions.

8.6 Expected interfaces

This chapter lists all functions that the Wdg module requires from other modules.

8.6.1 Mandatory interfaces

This chapter lists all interfaces which are required to fulfill a mandatory functionality of the module.

[WDG110] ⌈

API function	***Description***
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.

In addition to the functions listed above, further functions might be used to access the external watchdog over Dio or Spi. ⌋ ()

8.6.2 Optional interfaces

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

[WDG111] ⌈

API function	***Description***
Det_ReportError	Service to report development errors.
Gpt_DisableNotification	Disables the interrupt notification for a channel (relevant in normal mode).
Gpt_EnableNotification	Enables the interrupt notification for a channel (relevant in normal mode).
Gpt_StartTimer	Starts a timer channel.
Gpt_StopTimer	Stops a timer channel.

The interfaces to the GPT Driver are listed here, because they are used optionally, depending on how the watchdog servicing routine is implemented (see chapters 7.5 and 7.6).⌋ ()

8.6.3 Configurable interfaces

This module does not require any configurable interfaces.

9 Sequence diagrams

9.1 Watchdog initialization, setting trigger condition and mode.

The diagram shows the sequence to initialize the Wdg module, to set the trigger condition and to change the watchdog mode. Note that this is only an example. Especially, another “client” module than the Watchdog Manager (WdgM) could set the trigger condition.

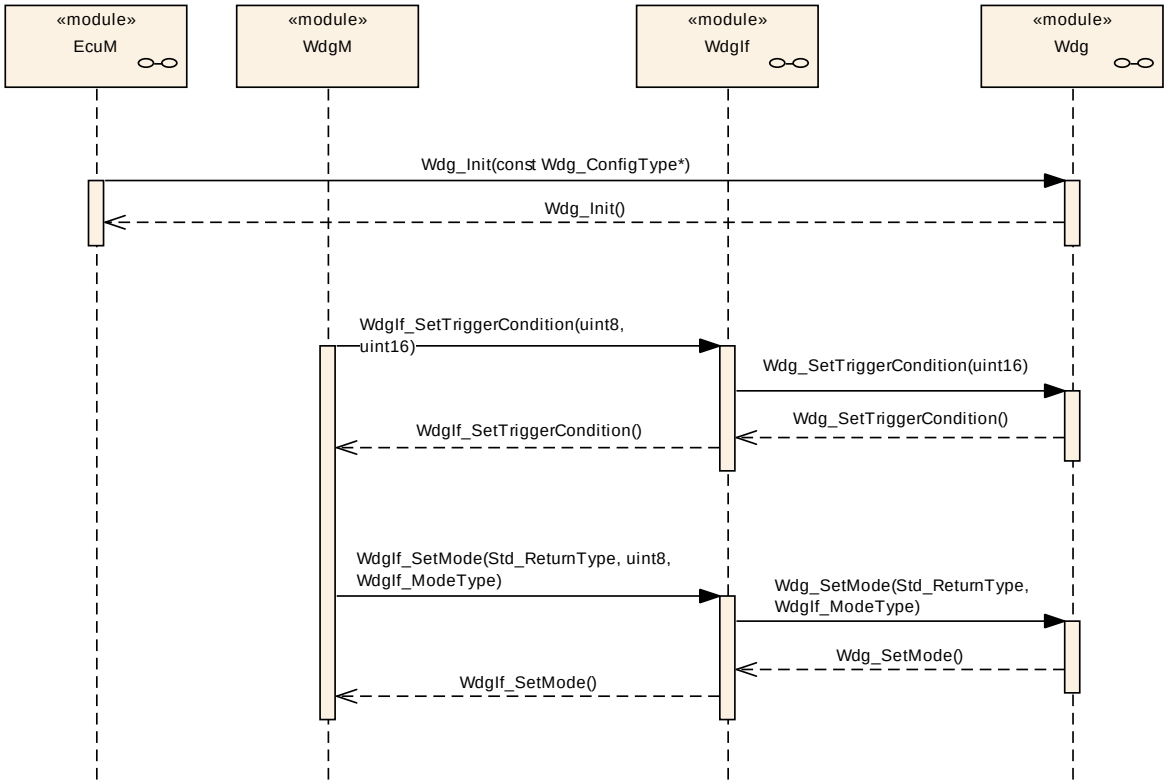

Figure 2: Sequence of watchdog initialization, setting trigger condition and mode switching.

9.2 Data exchange between watchdog driver and hardware

The diagram shows the sequence to trigger the watchdog hardware. Note that this is only an example. Depending on the implementation, the interrupt routine for servicing the watchdog could also come via a GPT Driver callback. For an external watchdog, the watchdog hardware cannot be accessed directly, but only via drivers of the MCAL layer, like SPI or DIO.

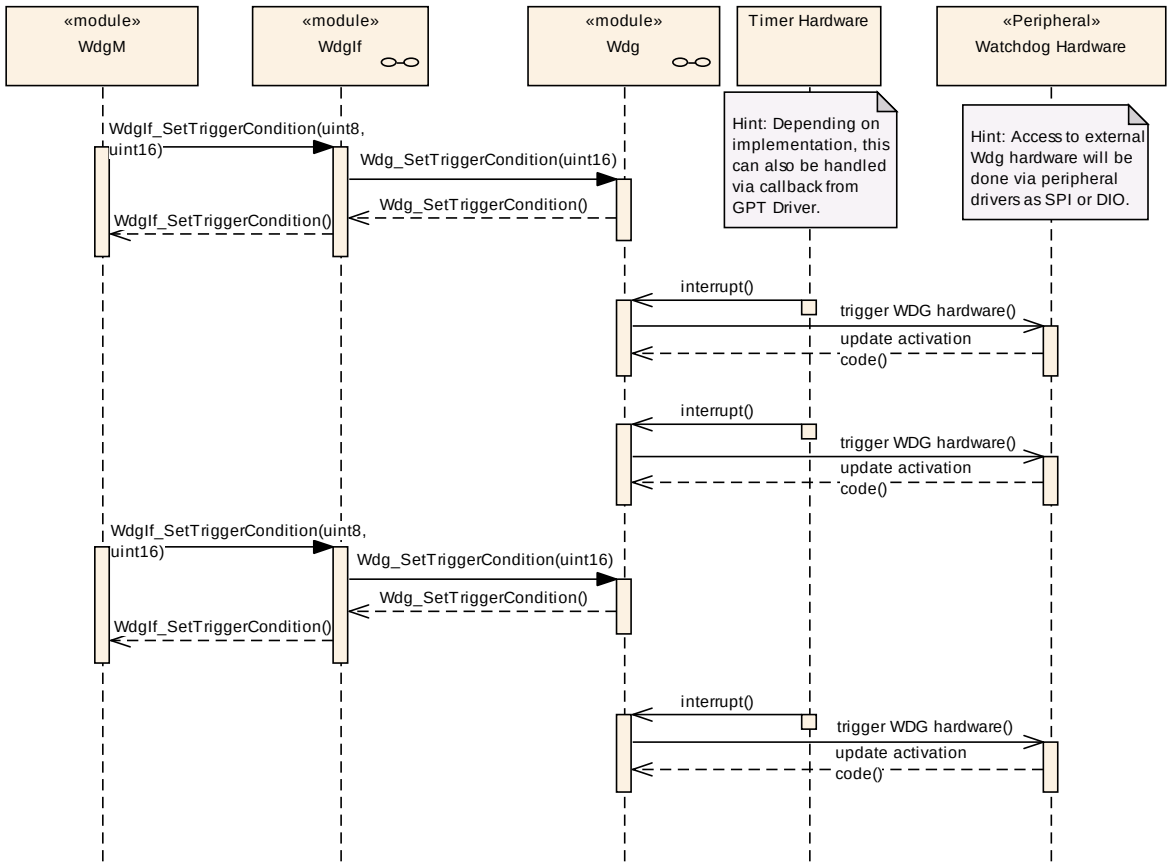

Figure 3: Data exchange between watchdog driver and hardware

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Wdg.

Chapter 10.3 specifies published information of the module Wdg.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture
- AUTOSAR ECU Configuration Specification
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration meta-model in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

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

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

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

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

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

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

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

10.2.1 Variants

[WDG157] ⌈This module shall support the configuration variant VARIANT-PRE-COMPILE. Only parameters with "Pre-compile time" configuration are allowed in this variant.⌋ ()

[WDG158] ⌈This module shall support the configuration variant VARIANT-LINK-TIME.
Parameters with "Pre-compile time" and "Link time" are allowed in this variant.⌋ ()

[WDG159] ⌈This module shall support the configuration variant VARIANT-POST-BUILD. Parameters with "Pre-compile time", "Link time" and "Post-build time" are allowed in this variant.⌋ ()

10.2.2 Wdg

SWS Item	**WDG073_Conf :**
Module Name	*Wdg*
Module Description	Configuration of the Wdg (Watchdog driver) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor-specific error references.
WdgGeneral	1	All general parameters of the watchdog driver are collected here.
WdgPublishedInformation	1	Container holding all Wdg specific published information parameters
WdgSettingsConfig	1	Configuration items for the different watchdog settings, including those for external watchdog hardware. Note: All postbuild parameters are handled via this container.

10.2.3 WdgDemEventParameterRefs

SWS Item	**WDG148_Conf :**
Container Name	WdgDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be

	extended by vendor-specific error references.
Configuration Parameters	

SWS Item	**WDG150_Conf :**		
Name	WDG_E_DISABLE_REJECTED		
Description	Reference to the DemEventParameter which shall be issued when the error "Initialization or mode switch failed because it would disable the watchdog" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**WDG149_Conf :**		
Name	WDG_E_MODE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "Setting a watchdog mode failed (during initialization or mode switch)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.4 WdgGeneral

SWS Item	**WDG114_Conf :**
Container Name	WdgGeneral
Description	All general parameters of the watchdog driver are collected here.
Configuration Parameters	

SWS Item	**WDG115_Conf :**		
Name	WdgDevErrorDetect {WDG_DEV_ERROR_DETECT}		
Description	Compile switch to enable / disable development error detection for this module. True: Development error detection enabled False: Development error detection disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**WDG116_Conf :**
Name	WdgDisableAllowed {WDG_DISABLE_ALLOWED}
Description	Compile switch to allow / forbid disabling the watchdog driver

	during runtime. True: Disabling the watchdog driver at runtime is allowed. False: Disabling the watchdog driver at runtime is not allowed.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Safety relevant compile switch, this has to be in accordance with the corresponding settings for the watchdog manager.		

SWS Item	**WDG117_Conf :**		
Name	WdgIndex		
Description	Represents the watchdog driver's ID so that it can be referenced by the watchdog interface.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**WDG130_Conf :**		
Name	WdgInitialTimeout		
Description	The initial timeout (sec) for the trigger condition to be initialized during Init function. It shall be not larger than WdgMaxTimeout.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**WDG131_Conf :**		
Name	WdgMaxTimeout		
Description	The maximum timeout (sec) to which the watchdog trigger condition can be initialized.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**WDG147_Conf :**

Name	WdgRunArea		
Description	Represents the watchdog driver execution area is either from ROM(Flash) or RAM as required with the particular microcontroller.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RAM	Watchdog driver to be executed out of RAM area	
	ROM	Watchdog driver to be executed out of ROM area	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**WDG118_Conf :**		
Name	WdgTriggerLocation {WDG_TRIGGER_LOCATION}		
Description	Location (memory address) of the watchdog trigger routine.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Only relevant if provided by hardware and needed by the system.		

SWS Item	**WDG119_Conf :**		
Name	WdgVersionInfoApi		
Description	Compile switch to enable / disable the version information API * True: API enabled * False: API disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.5 WdgSettingsConfig

SWS Item	**WDG082_Conf :**
Container Name	WdgSettingsConfig [Multi Config Container]
Description	Configuration items for the different watchdog settings, including those for external watchdog hardware. Note: All postbuild parameters are handled via this container.
Configuration Parameters	

SWS Item	**WDG120_Conf :**		
Name	WdgDefaultMode {WDG_DEFAULT_MODE}		
Description	Default mode for watchdog driver initialization. ImplementationType: WdgIf_ModeType		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	WDGIF_FAST_MODE		Default watchdog mode is "fast"
	WDGIF_OFF_MODE		Default watchdog mode is "off"
	WDGIF_SLOW_MODE		Default watchdog mode is "slow"
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module dependency: "Off" mode only possible if disabling the watchdog driver is allowed.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
WdgExternalConfiguration	0..1	Configuration items for an external watchdog hardware
WdgSettingsFast	1	Hardware dependent settings for the watchdog driver's "fast" mode.
WdgSettingsOff	1	Hardware dependent settings for the watchdog driver's "off" mode.
WdgSettingsSlow	1	Hardware dependent settings for the watchdog driver's "slow" mode.

Note:
The three modes are provided as containers for the reason that they might be referred by other modules and hence no parameters are needed. However those containers might be extended by the vendor (resp. hardware) specific configuration parameters, but these could not be standardized.

10.2.6 WdgSettingsFast

SWS Item	**WDG121_Conf :**
Container Name	WdgSettingsFast{WDG_SETTINGS_FAST}
Description	Hardware dependent settings for the watchdog driver's "fast" mode.
Configuration Parameters	

No Included Containers

10.2.7 WdgSettingsSlow

SWS Item	**WDG123_Conf :**
Container Name	WdgSettingsSlow{WDG_SETTINGS_SLOW}

Description	Hardware dependent settings for the watchdog driver's "slow" mode.
Configuration Parameters	

No Included Containers

10.2.8 WdgSettingsOff

SWS Item	**WDG122_Conf :**
Container Name	WdgSettingsOff{WDG_SETTINGS_OFF}
Description	Hardware dependent settings for the watchdog driver's "off" mode.
Configuration Parameters	

No Included Containers

10.2.9 WdgExternalConfiguration

SWS Item	**WDG112_Conf :**
Container Name	WdgExternalConfiguration{Wdg_ExternalConfiguration}
Description	Configuration items for an external watchdog hardware
Configuration Parameters	

SWS Item	**WDG113_Conf :**		
Name	WdgExternalContainerRef {WDG_EXTERNAL_CONTAINER_REF}		
Description	Reference to either - a DioChannelGroup container in case the hardware watchdog is connected via DIO pins - an SpiSequenceConfiguration container in case the watchdog hardware is accessed via SPI		
Multiplicity	0..1		
Type	Choice reference to [DioChannelGroup , SpiSequence]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module dependency: See DIO resp. SPI SWS		

No Included Containers

10.3 Published information

[WDG156] ⌈The standardized common published parameters as required by BSW00402 in the SRS General on Basic Software Modules [2] shall be published within the header file of this module and need to be provided in the BSW Module Description [9]. ⌋(BSW00374, BSW00379, BSW003, BSW00318)

Additional module-specific published parameters are listed below if applicable.

10.3.1 WdgPublishedInformation

SWS Item	**WDG074_Conf :**
Container Name	WdgPublishedInformation
Description	Container holding all Wdg specific published information parameters
Configuration Parameters	

SWS Item	**WDG127_Conf :**		
Name	WdgTriggerMode {WDG_TRIGGER_MODE}		
Description	Watchdog trigger mode (toggle/window/both)		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	WDG_BOTH	--	
	WDG_TOGGLE	--	
	WDG_WINDOW	--	
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency			

No Included Containers

Note:
`WdgTriggerMode` is only published for information purposes; this parameter is not used to configure the Watchdog Driver or the modules using the Watchdog Driver.

11 Changes from Release 3.1 to Release 4.0

11.1 Deleted SWS Items

SWS Item	*Rationale*
WDG133	Was a duplicate of WDG036
WDG108	Wdg_Trigger is obsolete due to windowed Wdg concept
WDG041	Wdg_Trigger is obsolete due to windowed Wdg concept
WDG036	Removed due to windowed Wdg concept
WDG104	Changed into to an integration hint (now in chapter 7.7)
WDG124	Not needed anymore due to windowed Wdg concept
WDG125	Not needed anymore due to windowed Wdg concept
WDG126	Not needed anymore due to windowed Wdg concept
WDG075	Not needed anymore due to windowed Wdg concept
WDG141	No testable requirement, only description
WDG143	No testable requirement, only description

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
WDG130	WDG160	WDG130 was redundant; WDG160 is more precise and was added
WDG142	WDG162, WDG166	Needed to split between external/internal Wdg,
All IDs of chapter 10	Suffix _conf added	Due to Standard rule in M1_Model

11.3 Changed SWS Items

SWS Item	*Rationale*
WDG067	Added instance ID removed again
WDG045	Reformulated
WDG080	“optional” removed
WDG076	Clarified wording in connection
WDG110	Added remark on GPT driver
WDG017, WDG018, WDG019, WDG052	Added explanations to the internal module’s states, as they are no more defined via WdgIf_StatusType
WDG001	Added note on the semantics of the init-configuration
WDG087	Improved (adopted from WdgIf); removed intra-module checks
WDG010	Added WDG_E_DISABLE_REJECTED to the table (was defined, but not shown in table)
WDG040	Added note on direct disabling of interrupts (has been allowed recently)
WDG010, WDG016	Renamed error code
WDG117_Conf	Refined min/max values
WDG082_Conf	Renamed and slightly restructured
WDG061	Replaced Gpt_Cbk.h by Wdg_<xxx>Cbk.h
WDG105	Dem_EventStatusType
WDG010, WDG089	Added WDG_E_PARAM_POINTER to the table due to bug #45367
WDG061	Wdg_<xxx>Cbk.h is now mandatory not optional

11.4 Added SWS Items

SWS Item	*Rationale*
WDG148	Debugging concept
WDG149	Debugging concept
WDG150	Debugging concept
WDG151	Debugging concept
WDG152	Debugging concept
WDG153	Debugging concept
WDG154	Debugging concept
WDG155	The API definition for Wdg_SetTriggerCondition() was lacking an ID; Added “milliseconds” in the description field of the API
WDG156	Added
WDG157	Added
WDG158	Added
WDG159	Added
WDG160	Added
WDG161	Added
WDG162	Added
WDG163	Added ID for existing requirement; Modified the explaining text
WDG164	Added
WDG165	Added
WDG166	Added
WDG168	Added
WDG147_Conf	WdgRunArea added
WDG130_Conf	WdgInitialTimeout added
WDG131_Conf	WdgMaxTimeout added
WDG169, WDG170, WDG172	Need to distinguish between internal and external watchdog API
WDG171	ID for Wdg_ConfigType was missing
WDG173	Added
WDG148_Conf, WDG149_Conf, WDG150_Conf	Added
WDG174	Added

12 Not applicable requirements

[WDG175]「 These requirements are not applicable to this specification.」(BSW00344, BSW00404, BSW00405, BSW170, BSW00419, BSW00383, BSW00375, BSW00416, BSW00437, BSW168, BSW00423, BSW00424, BSW00425, BSW00428, BSW00432, BSW00433, BSW00450, BSW00434, BSW00339, BSW00422, BSW00417, BSW161, BSW162, BSW005, BSW00415, BSW007, BSW00413, BSW00441, BSW00307, BSW00373, BSW00410, BSW00447, BSW00348, BSW00353, BSW00361, BSW00302, BSW00328, BSW00312, BSW006, BSW00449, BSW00377, BSW00304, BSW00355, BSW00378, BSW00306, BSW00308, BSW00309, BSW00371, BSW00376, BSW00359, BSW00360, BSW00440, BSW00330, BSW00443, BSW00444, BSW00445, BSW00446, BSW009, BSW00401, BSW172, BSW010, BSW00333, BSW00321, BSW00341, BSW00334, BSW12056, BSW12267, BSW12462, BSW12463, BSW12068, BSW12069, BSW157, BSW12155, BSW12063, BSW12075, BSW12067, BSW12077, BSW12078, BSW12265, BSW12167, BSW12168)