

<b>Document Title</b>	Specification of BSW Scheduler
<b>Document Owner</b>	AUTOSAR GbR
<b>Document Responsibility</b>	AUTOSAR GbR
<b>Document Version</b>	1.0.1
<b>Document Status</b>	Final
<b>Part of Release</b>	2.1
<b>Revision</b>	0014

<b>Document Change History</b>			
<b>Date</b>	<b>Version</b>	<b>Changed by</b>	<b>Change Description</b>
24.01.2007	1.0.1	AUTOSAR Administration	<ul style="list-style-type: none"><li>• “Advice for users” revised</li><li>• “Revision Information” added</li></ul>
01.12.2006	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

## Disclaimer

**Any use** of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2006 AUTOSAR Development Partnership. All rights reserved.

## Advice to users of AUTOSAR Specification Documents:

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

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

## Table of Contents

1	Introduction and functional overview.....	6
2	Acronyms and abbreviations.....	8
3	Related documentation.....	9
3.1	Input documents.....	9
3.2	Related standards and norms.....	9
4	Constraints and assumptions.....	10
4.1	Limitations.....	10
4.2	Applicability to car domains.....	10
5	Dependencies to other modules.....	11
5.1	File structure.....	11
5.1.1	Code file structure.....	11
5.1.2	Header file structure.....	12
5.1.3	Process description (informative).....	14
5.1.4	Basic Software Module Description Template (informative).....	16
6	Requirements traceability.....	18
7	Functional specification.....	23
7.1	Embedding main processing functions into the AUTOSAR OS context.....	23
7.1.1	General considerations.....	23
7.1.2	Triggering main processing functions.....	24
7.1.2.1	General concepts.....	24
7.1.2.2	RecurringEvents.....	25
7.1.2.3	SporadicEvents.....	27
7.1.2.4	Main processing functions triggered by RecurringEvents and SporadicEvents.....	32
7.2	Data consistency.....	35
7.2.1	Background.....	35
7.2.2	Built-in data consistency.....	35
7.2.3	ExclusiveAreas.....	36
7.2.3.1	Concept.....	36
7.2.3.2	Implementation.....	37
7.3	Error handling concept.....	39
7.3.1	Background.....	39
7.3.2	Error handling in the context of AUTOSAR OS/OSEK.....	39
7.3.2.1	Fatal errors.....	39
7.3.2.2	Protection errors.....	40
7.3.2.3	Application errors.....	40
7.3.3	Error handling of the BSW Scheduler.....	40
7.3.3.1	Production errors.....	40
7.3.3.2	Development errors.....	40
8	API specification.....	42
8.1	Imported types.....	42
8.1.1	Standard types.....	42

8.2	Type definitions .....	42
8.2.1	SchM_ReturnType .....	42
8.2.2	SchM_ConfigType.....	42
8.3	Function definitions .....	42
8.3.1	SchM_Init .....	42
8.3.2	SchM_Deinit.....	43
8.3.3	SchM_GetVersionInfo .....	43
8.3.4	SchM_Enter .....	44
8.3.5	SchM_Exit.....	45
8.3.6	SchM_ActMainFunction .....	45
8.3.7	SchM_CancelMainFunction .....	46
8.4	Call-back notifications .....	47
8.5	Scheduled functions .....	47
8.6	Expected Interfaces.....	47
8.6.1	Mandatory Interfaces .....	48
8.6.2	Optional Interfaces .....	48
8.6.3	Configurable interfaces .....	48
9	Sequence diagrams.....	49
10	Configuration specification.....	51
10.1	How to read this chapter .....	51
10.1.1	Configuration and configuration parameters .....	51
10.1.2	Variants.....	51
10.1.3	Containers.....	52
10.2	Containers and configuration parameters .....	53
10.2.1	Variants.....	53
10.2.2	BSWSchedulerConfiguration.....	53
10.2.3	MainFunctionMapping.....	54
10.2.4	ScheduleTableMapping .....	55
10.2.5	ModuleConfiguration .....	56
10.2.6	InstanceConfiguration .....	57
10.2.7	ExclusiveAreaConfiguration .....	58
10.2.8	ActivationPointConfiguration .....	59
10.2.9	CancelationPointConfiguration.....	60
10.2.10	SporadicEventConfiguration.....	61
10.3	Published Information.....	62
11	Changes to Release 1 .....	63
11.1	Deleted SWS Items .....	63
11.2	Replaced SWS Items .....	63
11.3	Changed SWS Items.....	63
11.4	Added SWS Items.....	63

# 1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module BSW Scheduler. It does neither specify nor recommend a particular implementation of the BSW Scheduler.

Figure 1 shows the position of the BSW Scheduler within the AUTOSAR Layered Software Architecture (see [2]). The BSW Scheduler is part of the System Services and provides therefore services for all modules of all layers.

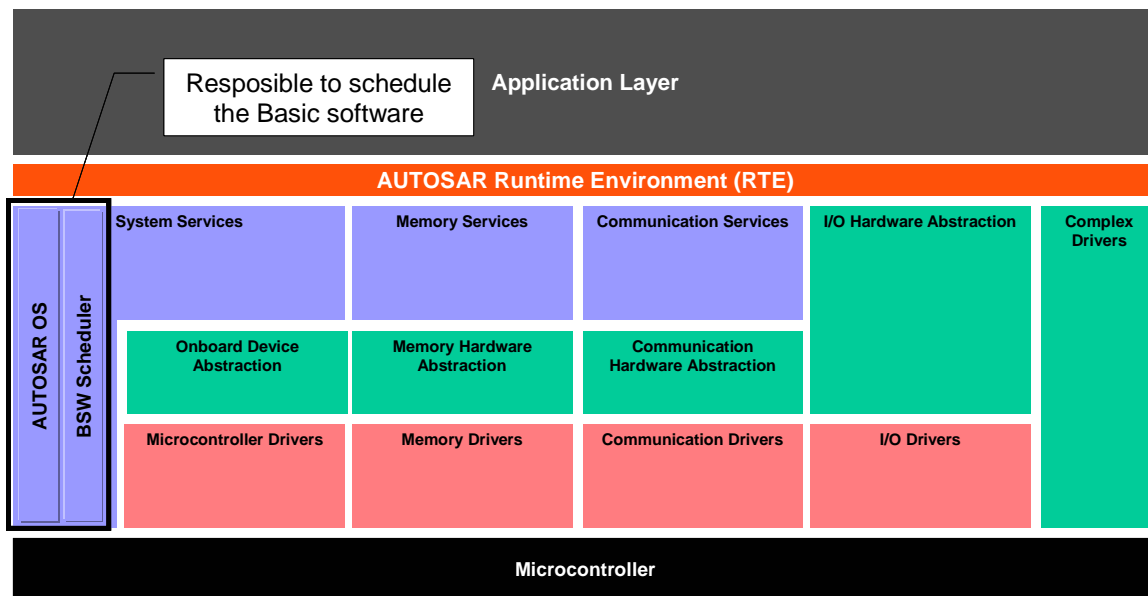


Figure 1: Position of the BSW Scheduler in the AUTOSAR Layered Software Architecture

However, unlike all other BSW modules, the BSW Scheduler offers concepts and services for the *integration*. Hence, this central module provides means to

- embed BSW module implementations into the AUTOSAR OS context
- trigger main processing functions of the BSW modules
- apply data consistency mechanisms for the BSW modules

The integrators task is to *apply* given means (of the AUTOSAR OS) in order to assemble BSW modules in a well-defined and efficient manner in a project specific context.

This also means that the BSW Scheduler only *uses* the AUTOSAR OS. It is *not* in the least a competing entity for the AUTOSAR OS scheduler.

Example 1.1: The BSW Scheduler uses the concept of the AUTOSAR OS ‘Task’ to implement the concept ‘Triggering of a BSW main processing function’.

Example 1.2: The BSW Scheduler uses the AUTOSAR OS services 'DisableAllInterrupts' and 'EnableAllInterrupts' to implement the BSW Scheduler services for entering and/or exiting a sequence of non-preemptable instructions.

The implementation of the BSW Scheduler bases on

- the BSW Module Description of the BSW modules
- the configuration of the BSW Scheduler

Example 1.3: The BSW Module Description depicts the BSW module `Module` by two main processing functions `<Module>_MainFunction_Transmit` and `<Module>_MainFunction_Receive`, beside others.

Example 1.4: The BSW Scheduler implements the triggering of two main processing functions `<Module1>_MainFunction` and `<Module2>_MainFunction` by calling them in a Task `Task_8ms`. The Task `Task_8ms` calls `<Module1>_MainFunction` before `<Module2>_MainFunction`. The implementation of the BSW Scheduler module activates the Task `Task_8ms` cyclically every eight millisecond. This means, the configuration of the BSW Scheduler mainly determines its implementation.

## 2 Acronyms and abbreviations

<b><i>Abbreviation / Acronym:</i></b>	<b><i>Description:</i></b>
BSW	Basic Software
RTE	AUTOSAR Runtime Environment
SW-C	AUTOSAR Software Component
OS	Operation System

### 3 Related documentation

#### 3.1 Input documents

- [1] List of Basic Software Modules  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_BasicSoftwareModules.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_BasicSoftwareModules.pdf)
- [2] Layered Software Architecture  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_LayeredSoftwareArchitecture.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_LayeredSoftwareArchitecture.pdf)
- [3] General Requirements on Basic Software Modules  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_SRS\\_General.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_SRS_General.pdf)
- [4] Specification of Standard Types  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_SWS\\_StandardTypes.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_SWS_StandardTypes.pdf)
- [5] Specification of Communication Stack Types  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_SWS\\_ComStackTypes.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_SWS_ComStackTypes.pdf)
- [6] Specification of ECU Configuration  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_ECU\\_Configuration.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_ECU_Configuration.pdf)
- [7] Specification of Operating System  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_SWS\\_OS.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_SWS_OS.pdf)
- [8] Software Component Template  
[https://svn.autosar.org/repos/10Releases/AUTOSAR\\_SoftwareComponentTemplate.pdf](https://svn.autosar.org/repos/10Releases/AUTOSAR_SoftwareComponentTemplate.pdf)

#### 3.2 Related standards and norms

- [9] OSEK/VDX Operating System, Version 2.2.2  
[www.osek-vdx.org](http://www.osek-vdx.org)

## **4 Constraints and assumptions**

### **4.1 Limitations**

This specification assumes that all modules of the AUTOSAR Basic Software run in the same OS-Application (see [7] for details).

### **4.2 Applicability to car domains**

There are no explicit restrictions for the applicability to car domains.

## 5 Dependencies to other modules

As explained in the introduction chapter the BSW Scheduler is the central integration entity. Therefore, the BSW Scheduler has dependencies to all BSW modules; whereas two different dependency types - use dependencies and realize dependencies (seen from the BSW Scheduler) - are distinguished.

**INTEGR004:** Use dependencies: The BSW Scheduler *uses* the main processing functions of the BSW modules in the sense that the AUTOSAR OS objects 'Task' call them. The AUTOSAR OS is a particular module, because the BSW Scheduler uses its entire functionality.

**INTEGR005:** Realize dependencies: The BSW Scheduler provides services (APIs) for all other BSW modules, except for the AUTOSAR OS. Hence, the BSW Scheduler *realizes* functionality for those BSW modules (see Chapter 8.3).

From an architectural point of view, there exists currently no relationship between the RTE and the BSW Scheduler. This may be a weakness in the architecture, because of the following reasons. The RTE provides the infrastructure for Software Components, i.e. the RTE embeds runnable entities in a task context. The BSW Scheduler provides the infrastructure for the BSW modules, i.e. the BSW Scheduler embeds main processing function in a task context. However, with the current architecture, runnable entities and main processing functions cannot be embedded in the same task.

### 5.1 File structure

#### 5.1.1 Code file structure

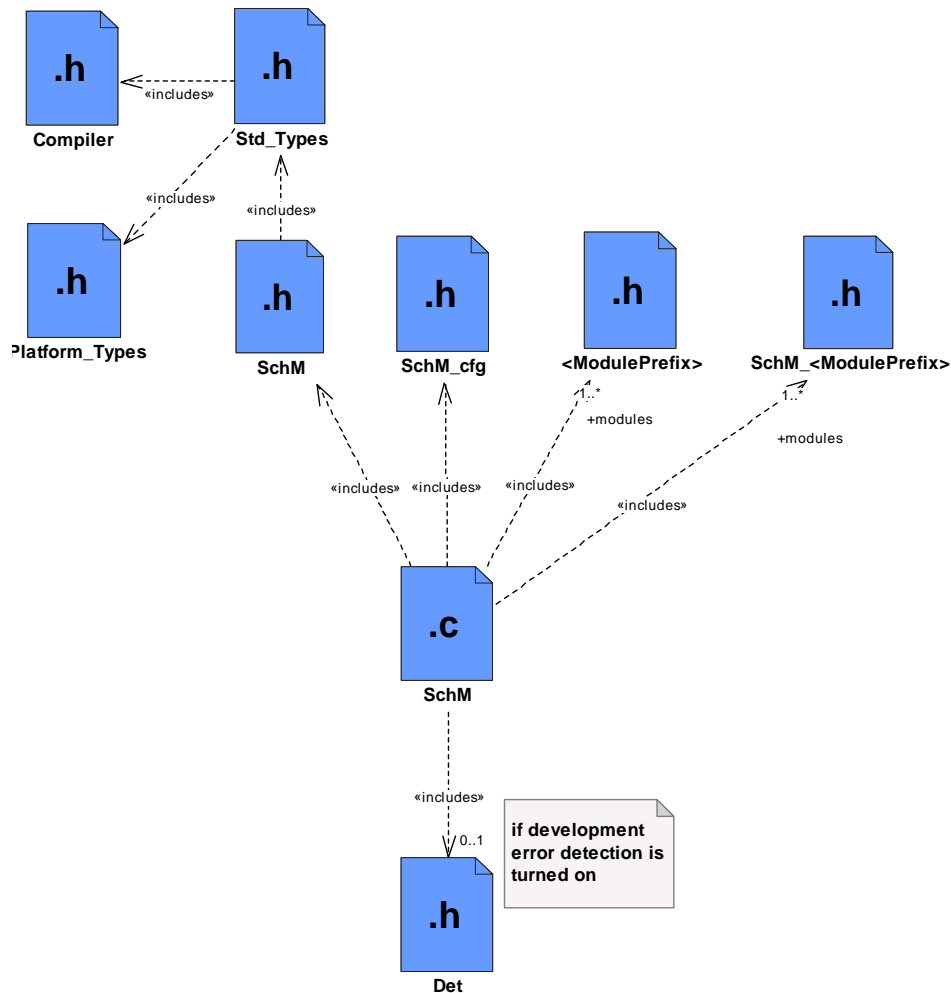
**INTEGR006:** In general, this specification shall not restrict implementations in a particular context (of a project). Therefore, this specification shall not define the complete code file structure.

**INTEGR017:** The BSW Scheduler is the sole entity that is responsible to schedule BSW modules (except for AUTOSAR OS and ECU State Manager) in order to ease and simplify the task of their integration. It is assumed impossible to guarantee a proper integration of the BSW (valid schedule, data consistency, efficiency), if the BSW Scheduler module and therefore the task of integration were split.

**INTEGR007:** The code-file structure shall include the file `SchM_Cfg.c` in case the implementation of the BSW Scheduler uses generated configuration parameters of the class 'pre-compile time'. This file shall contain all generated pre-compile time configuration parameter.

**5.1.2 Header file structure**

**INTEGR071:** The include file structure for the BSW Scheduler module `SchM.c` shall be as follows:



**Figure 2: Header file structure of the BSW Scheduler**

**INTEGR091:** The implementation of the BSW Scheduler module `SchM.c` shall include the header file `SchM.h`. `SchM.h` includes `Std_Types.h`, which in turn includes `Compiler.h` and `Platform_Types.h`.

`SchM.h` exports general services of the BSW Scheduler, i.e. `SchM_Init`, `SchM_Deinit` and `SchM_GetVersionInfo`.

**INTEGR072:** The implementation of the BSW Scheduler module `SchM.c` shall include all header files `SchM_<ModulePrefix>.h`.

`SchM_<ModulePrefix>.h` exports specific services of the BSW Scheduler for a particular module `<ModulePrefix>`, i.e. `SchM_Enter_<ModulePrefix>`,

SchM\_Exit\_<ModulePrefix>, SchM\_ActMainFunction\_<ModulePrefix> and SchM\_CancelMainFunction\_<ModulePrefix>.

If the module <ModulePrefix> is available as source code, the implementer of the BSW Scheduler module (i.e. the integrator) is free to substitute this header file by an optimized version that contains (in the best case) a macro-based implementation of the functionality.

**INTEGR010:** The implementation of the BSW Scheduler module SchM.c shall include all header files <ModulePrefix>.h of the used BSW modules.

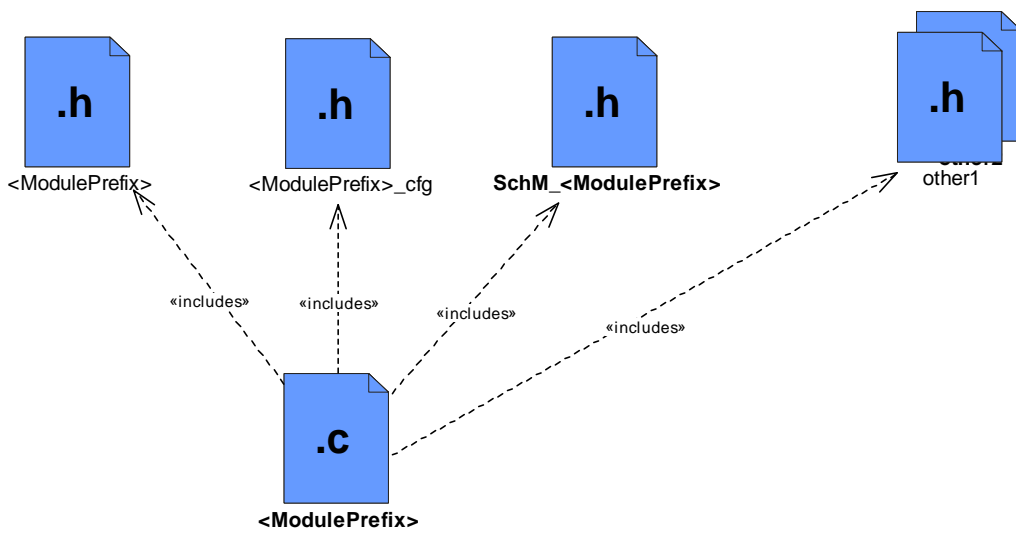
The BSW Scheduler needs to include <ModulePrefix>.h in order to be able to access the main processing functions of the BSW module <ModulePrefix>.

**INTEGR073:** The implementation of the BSW Scheduler module SchM.c shall include SchM\_Cfg.h for pre-compile-time configuration parameter of the BSW Scheduler.

**INTEGR074:** The implementation of the BSW Scheduler module SchM.c shall include Det.h if development error detection is turned on (SCHM\_DEV\_ERROR\_DETECT = ON).

The include file structures of the BSW modules need to be extended by the respective header files SchM\_<ModulePrefix>.h in order to access the module specific functionality provided by the BSW Scheduler.

**INTEGR092:** Each BSW module implementation <ModulePrefix>.c shall include its respective header file SchM\_<ModulePrefix>.h.

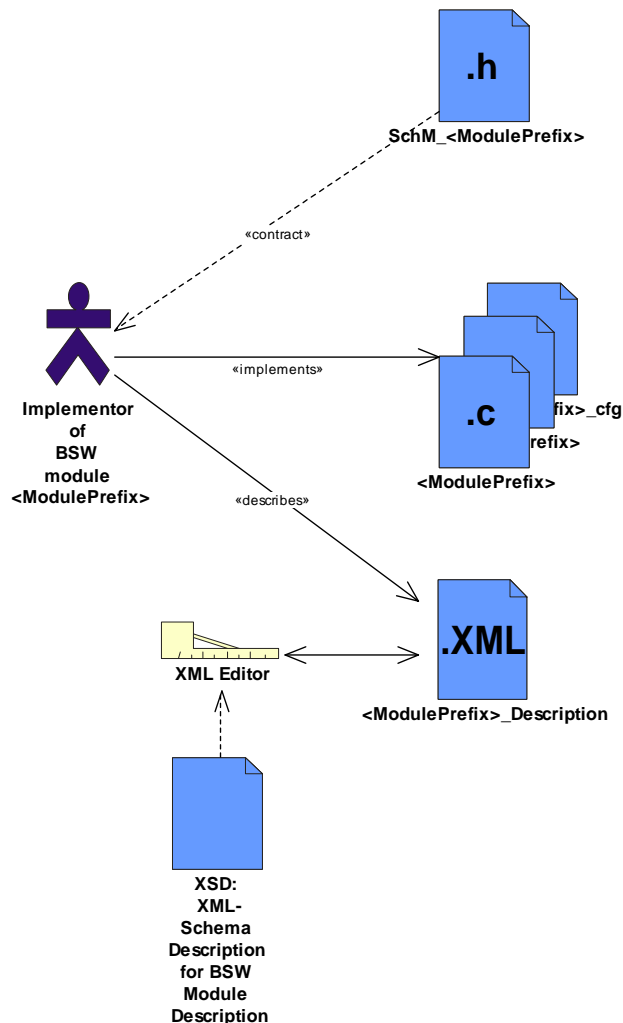


**Figure 3: Header file structure of a BSW module <ModulePrefix>.c**

**5.1.3 Process description (informative)**

This section shall shortly describe the collaboration between the implementers and the integrator, which is required to implement the BSW Scheduler. Note that this section is only of informative and informal character and not part of the specification. Furthermore, this description is not exhaustive!

Figure 4 shows the process using the services of AUTOSAR BSW Scheduler from an implementer’s point-of-view.



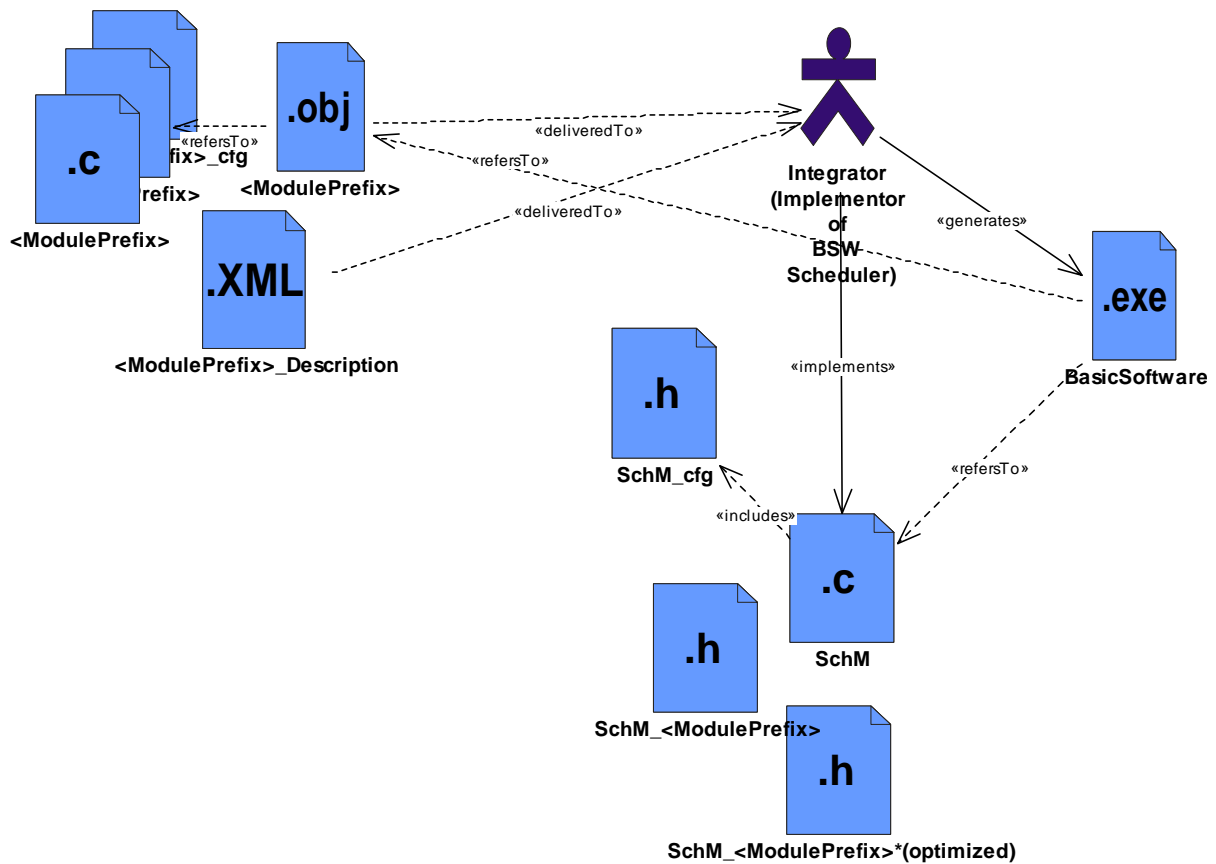
**Figure 4: Development process from an implementer’s point-of-view**

The Header file `SchM_<ModulePrefix>.h` depicts a contract for the implementer, because it exports all necessary services for this particular module `<ModulePrefix>`. For the BSW module internal testing, a 'stub' implementation of the services in file `SchM_<ModulePrefix>.h` has to be developed. This 'stub software' is then not delivered to the Integrator (Implementer of the BSW Scheduler). Beyond the 'normal' implementation artifacts (source files or object code), the implementer needs to describe its implementation further in the Basic Software Module Description. This description contains specific information about entities, behavior and implementation of the BSW module and serves as an important input

for the integrator. E.g., the integrator has now information about which module entities protect critical sections (concept of ExclusiveAreas) or which entities call others modules services and call therefore potentially others critical sections.

Since the BSW Module Description Template is an integral part of the AUTOSAR meta model, deduced module descriptions can be represented and exchanged via XML. This also enables to edit the XML document of a specific module description directly by using a XML editor under the assumption that an XML Schema Description represents the corresponding BSW Module Description Template.

Figure 5 shows the process from an integrators point-of-view.



**Figure 5: Integration process from an integrators point-of-view**

The integrator receives all implementation relevant artifacts - source code, object code and module description - from the module implementers in order to bind them into executable software. As described before, the implementation of the BSW Scheduler is an integral task of this integration work. If a module is available as source code, it may be possible to implement the corresponding BSW Scheduler functionality as a macro layer. In this case, the integrator is free to substitute the related header file `SchM_<ModulePrefix>.h` by an optimized variant.

The module descriptions delivered by the implementers are important inputs for integration decisions and therefore for the configuration and finally the implementation of the BSW Scheduler.

#### 5.1.4 Basic Software Module Description Template (informative)

As indicated above, the implementation of the BSW Scheduler heavily relies on proper module descriptions that base on a BSW Module Description Template. Therefore, this section will shortly subsume the main concepts. Unfortunately, this language is currently under development and the specification yet to fix. Therefore, the shown meta model diagrams serve only as assumption for this specification<sup>1</sup>. Like in Section 5.1.3 this description is neither exhaustive nor is it part of this specification, by no means.

In general, it is assumed that the modules and their implementations need to describe extensively in order to allow for a correct and easy integration. E.g., the integrator of the Basic Software needs deep knowledge in order to apply a proper protection scheme ensuring data consistency:

- name and list potentially shared resources
- name and list all (internal) entities that access that particular resources
- establish a connection between entities of other modules and the entry points that finally access shared resources
- assumptions for the implementations of critical sections (ExclusiveArea)
- take 'possible' task contexts of the resource access into account
- and other issues

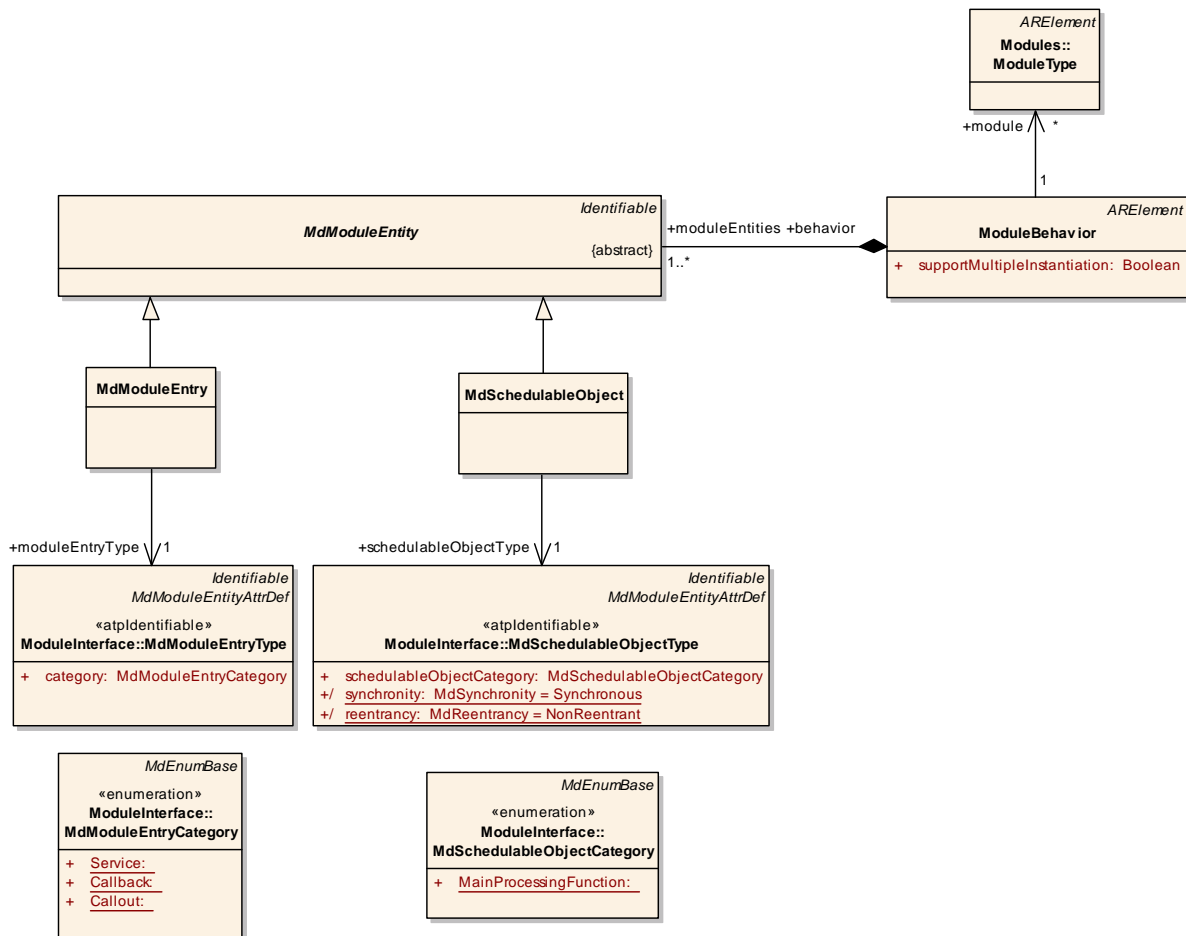
The main assumption for this document is that the BSW Module Description will follow the concepts and principles of the AUTOSAR Software Component Template (see [8]).

This means that there also three different layers exist:

- module/interface level, which describes the entities of the module
- behavior level, which describes the triggers of schedulable objects and means for ensuring data consistency (ExclusiveAreas)
- implementation level, which describes the memory consumption, the timing of a specific implementation and the like

---

<sup>1</sup> The corresponding meta model can currently be found in the Scratchpad/BswModuleDescription view of the AUTOSAR meta model AR\_MetaModel\_Master.eap



**Figure 6: Classification of BSW module entities**

Figure 6 shows an excerpt of the BSW Module Description Template, whereas it covers elements of the first two levels. It shows that there are means to describe modules with their interfaces and behaviors (a module can have several behaviors).

Beside others, the module behavior is characterized by its entities (see Figure 6). Thereby, the module might have two different categories: module entries and schedulable objects. Module entries are typed by a specific ModuleEntryType, which specifies the module entry category (service, callback, callout) as well as the signature of the API. Schedulable objects are also typed; here by a specific SchedulableObjectType, which only has one category: MainProcessingFunction. Furthermore, schedulable objects are specified to be non-reentrant, have no parameters and no return value.

Note that the meta model elements ModuleEntryType and SchedulableObjectType belong to the first level: the module/interface level.

All other relevant meta model elements, necessary for the comprehension of this specification (like critical sections (ExclusiveAreas)), will be introduced at the point where the corresponding concepts are discussed; that means mainly in Chapter 7.

## 6 Requirements traceability

Document: AUTOSAR General Requirements on Basic Software Modules

<b>Requirement</b>	<b>Satisfied by</b>
[BSW00344] Reference to link-time configuration	<a href="#">INTEGR083</a>
[BSW00404] Reference to post build time configuration [approved]	Not applicable (only #define's as pre-compile time parameters)
[BSW00405] Reference to multiple configuration sets	Not applicable (only #define's as pre-compile time parameters)
[BSW00345] Pre-compile-time configuration	<a href="#">INTEGR007</a> , <a href="#">INTEGR073</a>
[BSW159] Tool-based configuration	Not applicable (requirement on implementation, not on specification)
[BSW167] Static configuration checking	Not applicable (requirement on implementation, not on specification)
[BSW171] Configurability of optional functionality	Chapter 8 and Chapter 10
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00380] Separate C-Files for configuration parameters	<a href="#">INTEGR007</a>
[BSW00419] Separate C-Files for pre-compile time configuration parameters	<a href="#">INTEGR007</a>
[BSW00381] Separate configuration header file for pre-compile time parameters	<a href="#">INTEGR073</a>
[BSW00412] Separate H-File for configuration parameters	Not applicable (implementation related requirement)
[BSW00383] List dependencies of configuration files	Chapter 10
[BSW00384] List dependencies to other modules	Chapter 5, <a href="#">INTEGR004</a> , <a href="#">INTEGR005</a> , <a href="#">INTEGR006</a> , <a href="#">INTEGR017</a> , <a href="#">INTEGR007</a> , <a href="#">INTEGR071</a> , <a href="#">INTEGR091</a> , <a href="#">INTEGR072</a> , <a href="#">INTEGR010</a> , <a href="#">INTEGR073</a> , <a href="#">INTEGR074</a> , <a href="#">INTEGR092</a>
[BSW00387] Specify the configuration class of callback function	Not applicable (this module does not provide any callback functions)
[BSW00388] Introduce containers	Chapter 10
[BSW00389] Containers shall have names	Chapter 10
[BSW00390] Parameter content shall be unique within the module	Chapter 10
[BSW00391] Parameter shall have unique names	Chapter 10
[BSW00392] Parameters shall have a type	Chapter 10
[BSW00393] Parameters shall have a range	Chapter 10
[BSW00394] Specify the scope of the parameters	Chapter 10
[BSW00395] List the required parameters (per parameter)	Chapter 10
[BSW00396] Configuration classes	Chapter 10
[BSW00397] Pre-compile-time parameters	Chapter 10; However, this specification does not specify a particular implementation or a set of implementations of the BSW Scheduler. Therefore, Chapter 10 does not list implementation specific parameters: There might

	be additional implementation relevant pre-compile-time parameters, then listed in Chapter 10.
[BSW00398] Link-time parameters	Not applicable (only #define's as pre-compile time parameters)
[BSW00399] Loadable Post-build time parameters	Not applicable (only #define's as pre-compile time parameters)
[BSW00400] Selectable Post-build time parameters	Not applicable (only #define's as pre-compile time parameters)
[BSW00402] Published information	<a href="#">INTEGR045</a>
[BSW00375] Notification of wake-up reason	Not applicable (this module does not provide any wake-up reason)
[BSW101] Initialization interface	<a href="#">INTEGR023</a> ,
[BSW00416] Sequence of Initialization	Not applicable (this module does not handle initialization)
[BSW00406] Check module initialization	Not applicable (requirement on implementation, not on specification)
[BSW168] Diagnostic Interface of SW components	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00407] Function to read out published parameters	<a href="#">INTEGR030</a>
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00429] Restricted BSW OS functionality access	Chapter 1 and Chapter 7
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (this module does not provide any schedulable objects)
[BSW00336] Shutdown interface	<a href="#">INTEGR027</a>
[BSW00337] Classification of errors	Chapter 7.3
[BSW00338] Detection and Reporting of development errors	<a href="#">INTEGR080</a> , <a href="#">INTEGR077</a> , <a href="#">INTEGR078</a> , <a href="#">INTEGR085</a> , <a href="#">INTEGR076</a>
[BSW00369] Do not return development error codes via API	API specified in Chapter 8
[BSW00339] Reporting of production relevant error status	Not applicable (this module does not raise production relevant errors)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (this module is not part of the Non-Basic Software)
[BSW00323] API parameter checking	<a href="#">INTEGR076</a> , Chapter 7.3.3.2
[BSW004] Version check	<a href="#">INTEGR045</a>
[BSW00409] Header files for production code error IDs	Not applicable (this module does not raise production relevant errors)
[BSW00385] List possible error notifications	Chapter 7.3.3.2
[BSW00386] Configuration for detecting an error	Chapter 7.3
[BSW161] Microcontroller abstraction	Not applicable (requirement on system design, not on a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on system design, not on a single module)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on system design, not on a single

	module)
[BSW00415] User dependent include files [approved]	<a href="#">INTEGR072</a>
[BSW164] Implementation of interrupt service routines	Not applicable (requirement on implementation, not on specification)
[BSW00325] Runtime of interrupt service routines	Not applicable (requirement on implementation, not on specification)
[BSW00326] Transition from ISRs to OS tasks	Chapter 7.1.2.3.1, Chapter 7.2.3.1
[BSW00342] Usage of source code and object code	Chapter 7
[BSW00343] Specification and configuration of time	<a href="#">INTEGR069</a>
[BSW160] Human-readable configuration data	Chapter 10
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW00300] Module naming convention	API specifications in Chapter 8
[BSW00413] Accessing instances of BSW modules	API specifications in Chapter 8
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (this module does not specify a BSW driver)
[BSW00305] Self-defined data types naming convention	Type specification in Chapter 8
[BSW00307] Global variables naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00310] API naming convention	API specification of Chapter 8
[BSW00373] Main processing function naming convention	Not applicable (this module does not provide any schedulable objects)
[BSW00327] Error values naming convention	<a href="#">INTEGR070</a>
[BSW00335] Status values naming convention	<a href="#">INTEGR070</a>
[BSW00350] Development error detection keyword	<a href="#">INTEGR067</a>
[BSW00408] Configuration parameter naming convention	Chapter 10
[BSW00410] Compiler switches shall have defined values	Chapter 10, <a href="#">INTEGR067</a>
[BSW00411] Get version info keyword	<a href="#">INTEGR031</a>
[BSW00346] Basic set of module files	Chapter 5.1
[BSW158] Separation of configuration from implementation	Chapter5, <a href="#">INTEGR071</a>
[BSW00314] Separation of interrupt frames and service routines	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00370] Separation of callback interface from API	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00348] Standard type header	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00353] Platform specific type header	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00361] Compiler specific language extension header	Not applicable (requirement on AUTOSAR architecture, not a single module)

[BSW00301] Limit imported information	<a href="#">INTEGR071</a>
[BSW00302] Limit exported information	<a href="#">INTEGR071</a>
[BSW00328] Avoid duplication of code	Not applicable (requirement on implementation, not on specification)
[BSW00312] Shared code shall be reentrant	Not applicable (requirement on implementation, not on specification)
[BSW006] Platform independency	Not applicable (requirement on implementation, not on specification)
[BSW00357] Standard API return type	API specification of Chapter 8
[BSW00377] Module specific API return types	API specification of Chapter 8.2, <a href="#">INTEGR070</a>
[BSW00304] AUTOSAR integer data types	API specification of Chapter 8
[BSW00355] Do not redefine AUTOSAR integer data types	API specification of Chapter 8
[BSW00378] AUTOSAR boolean type	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not on specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not on specification)
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not on specification)
[BSW00371] Do not pass function pointers via API	API specification of Chapter 8
[BSW00358] Return type of <code>init()</code> functions	<a href="#">INTEGR023</a>
[BSW00414] Parameter of <code>init</code> function	<a href="#">INTEGR023</a>
[BSW00376] Return type and parameters of main processing functions	Not applicable (this module does not provide any schedulable objects)
[BSW00359] Return type of callback functions	Not applicable (this module does not provide any callback functions)
[BSW00360] Parameters of callback functions	Not applicable (this module does not provide any callback functions)
[BSW00329] Avoidance of generic interfaces	No generic interface specified.
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not on specification, but examples of Chapter 7 point in this direction)
[BSW00331] Separation of error and status values	Chapter 8.1.1 vs Chapter 8.2.1
[BSW009] Module User Documentation	Not applicable (requirement on documentation, not on specification)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (requirement on documentation, not on specification)
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (requirement on documentation, not on specification)
[BSW010] Memory resource documentation	Not applicable (requirement on documentation, not on specification)
[BSW00333] Documentation of callback function	Not applicable

context	(requirement on documentation, not on specification)
[BSW00374] Module vendor identification	<a href="#">INTEGR045</a>
[BSW00379] Module identification	<a href="#">INTEGR045</a>
[BSW003] Version identification	<a href="#">INTEGR045</a>
[BSW00318] Format of module version numbers	<a href="#">INTEGR045</a>
[BSW00321] Enumeration of module version numbers	Not applicable (requirement on documentation, not on specification)
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation, not on specification)
[BSW00334] Provision of XML file	XML is only one representation of input or configuration information for the BSW Scheduler. Configuration and module description (templates) are handled in Chapter 10.

Document: AUTOSAR General requirements on Basic Software Modules, BSW Scheduler

<b>Requirement</b>	<b>Satisfied by</b>
[BSW00424] BSW main processing function task allocation	<a href="#">INTEGR068</a>
[BSW00425] Trigger conditions for schedulable objects	Chapter 7.1.2, <a href="#">INTEGR064</a> , <a href="#">INTEGR044</a> , <a href="#">INTEGR063</a>
[BSW00426] Exclusive areas in BSW modules	Chapter 7.2, <a href="#">INTEGR047</a> , <a href="#">INTEGR093</a> , <a href="#">INTEGR094</a>
[BSW00427] ISR description for BSW modules	Not applicable (requirement on documentation, not on specification)
[BSW00428] Execution order dependencies of main processing functions	<a href="#">INTEGR068</a>
[BSW00431] The BSW Scheduler module implements task bodies	<a href="#">INTEGR015</a>
[BSW00433] Calling of main processing functions	<a href="#">INTEGR012</a> , <a href="#">INTEGR013</a>
[BSW00434] The Schedule Module shall provide an API for exclusive areas	<a href="#">INTEGR032</a> (Enter), <a href="#">INTEGR035</a> (Exit)

## 7 Functional specification

The software specifications of the BSW modules specify services and main processing functions. However, module specification as well as the implementation is decoupled from a certain infrastructure (mainly determined by the OS). Integration in this context signifies mapping: the BSW Scheduler maps the implementation onto the given infrastructure, regardless of whether the implementation is available as source code or as object code.

In this sense, this chapter continues the discussion started in Chapter 1. There, we already formulated main requirements for the BSW Scheduler. In detail, it shall provide means to

- embed the implementation of BSW modules into the AUTOSAR OS context
- trigger main processing functions of the BSW modules
- apply data consistency mechanisms for the BSW modules

Altogether, they shall ensure that particular implementations of modules can be integrated into the Basic Software properly.

### 7.1 Embedding main processing functions into the AUTOSAR OS context

#### 7.1.1 General considerations

**INTEGR012:** The BSW Scheduler shall be the only instance that embeds BSW main processing functions into the AUTOSAR OS infrastructure.

Practically, [INTEGR012](#) means that the BSW Scheduler shall

- set up AUTOSAR OS Task bodies
- arrange the calls of the main processing functions within the OS Task bodies

In this sense, the BSW Scheduler implementation bases deeply on its configuration (see Chapter 10.2 for details). Because in order to satisfy [INTEGR012](#), the BSW Scheduler configuration needs to refer to OS Tasks, needs to specify the assignments of the main processing functions to given OS Tasks and finally, it needs to specify the execution order of the main processing functions within a particular OS Task.

**INTEGR015:** The BSW Scheduler shall implement the AUTOSAR OS Task bodies of the BSW Layer based on its configuration.

**INTEGR013:** The BSW Scheduler shall implement the calls to the main processing functions within the AUTOSAR OS Task bodies based on its configuration. Thereby, the BSW Scheduler shall respect the configured (unambiguous) execution order of main processing functions within a single AUTOSAR OS Task context.

AUTOSAR OS classifies two types of tasks: basic tasks and extended tasks. The configuration of the BSW Scheduler allows mapping of main processing functions to both types; basic tasks and extended tasks. However, note that the main processing functions themselves must not enter a wait state.

**INTEGR014:** The BSW Scheduler (generator/implementer) shall reject configurations where main processing functions can enter wait states.

Example 7.1:

Implementation of the BSW Scheduler module:

Schm.c:

```
TASK (Task_7ms) {
    ...
    Com_MainFunction_Receive ();
    Com_MainFunction_Transmit ();
    ...
}
```

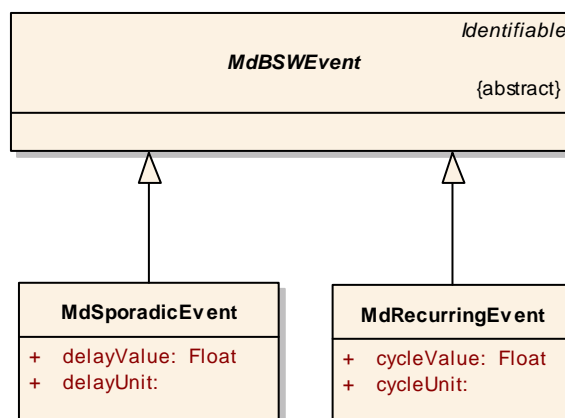
**7.1.2 Triggering main processing functions**

**7.1.2.1 General concepts**

Although Chapter 7.1.1 discusses main approaches, it did not consider aspects of triggering main processing functions for the integration into an OS context.

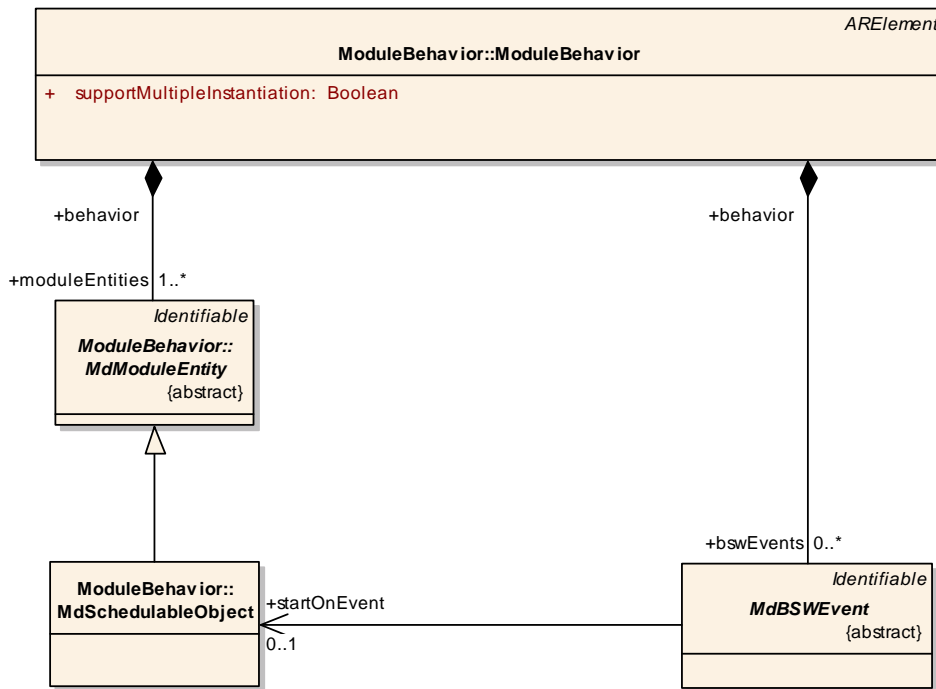
In general, main processing functions, as schedulable objects, shall react on specific trigger conditions. The BSW Module Description (Template) models trigger conditions as so-called *BSWEvents*. Currently, the concept of *BSWEvents* distinguishes two different types (see Figure 7):

- *SporadicEvents*, which comprise trigger that appear occasionally
- *RecurringEvents*, which comprise trigger that recur cyclically



**Figure 7: Types of BSWEvents**

Furthermore, Figure 8 shows the general connection between *BSWEvents* and the reaction of schedulable objects (main processing function). The occurrence of a specific BSWEvent can activate (start) a schedulable function (main processing function). However, unlike the meta model of the Software Component Template, schedulable objects (main processing functions) must not enter a wait state.



**Figure 8: Interactions of BSWEvents and SchedulableObjects**

Note again, that the aforementioned concept *only* denotes the context of main processing functions. It does *not* describe the implementation with respect to an AUTOSAR OS context. The BSW Module Description serves only as an input to the BSW Scheduler and therefore sets the conditions (requirements) for a ‘proper’ implementation of the BSW Scheduler.

Example 7.2: The cyclical occurrence of the *RecurringEvent TE7MS* activates a main processing function every seven millisecond.

### 7.1.2.2 RecurringEvents

#### 7.1.2.2.1 Background

The concept of *RecurringEvents* depicts the ability to process recurring events. The following classes of *RecurringEvents* are possible:

- fixed or variable cyclic recurring events, where the type of trigger condition is Time<sup>2</sup> - in the document referred to as timed *RecurringEvents*

<sup>2</sup> This type of event is called TimingEvent in the Software Component Template

- recurring events, where the type of trigger condition only indirectly relates to Time (e.g., angle of a crankshaft)

#### 7.1.2.2.2 Fixed cyclic RecurringEvents based on Time

The term 'fixed cyclic' means that a specific configured cycle time shall not change during runtime.

Thus, a most likely implementation uses AUTOSAR OS 'Tasks' activated by AUTOSAR OS 'Alarms' for these time-based *RecurringEvents* with fixed cycle times. The same task can call (implement) different main processing functions if they have the same attribute 'cycle time'.

However, note that different objects 'Alarm' can base on different objects 'Counter'. The system counter is not the only counter in the system. Be aware of the fact that this might lead to different implementation with different behaviors.

E.g., two main processing functions triggered by different *RecurringEvents* but identical attributes might be implemented by two different tasks triggered by two different alarms, on one hand. Alternatively, only one task will implement (call) them. The implementation of the BSW Scheduler will behave differently.

However, the same task will most likely trigger different main processing in case the same *RecurringEvent* triggers them.

The BSW Scheduler is not responsible for the lifecycle management. The AUTOSAR OS scheduler manages and schedules the AUTOSAR OS objects 'Task'.

Example 7.3: Two main processing functions `Com_MainFunction_Receive ()` and `Com_MainFunction_Transmit ()` shall process every seven millisecond.

The BSW Scheduler module implements this by using the AUTOSAR OS objects 'Task' and 'Alarm'.

Implementation of the BSW Scheduler module:

#### AUTOSAR OS configuration:

```
COUNTER SYSTEM_COUNTER {
    /* 1MS system counter */
    MAXALLOWEDVALUE = 65535;
    TICKSPERBASE = 1000000;
    MINCYCLE = 1;
};

ALARM TASK_7MS {
    COUNTER = SYSTEM_COUNTER;
    ACTION = ACTIVATETASK {
        TASK = Task_7ms;
    };
};
```

SchM\_cfg.h:

```
#define START_TIME_7MS          0
#define CYCLE_TIME_7MS         7
```

SchM.c:

```
void
SchM_Init () {
    SetRelAlarm (TASK_7MS, START_TIME_7MS, CYCLE_TIME_7MS);
}

TASK (Task_7ms) {
    ...
    Com_MainFunction_Receive ();
    Com_MainFunction_Transmit ();
    ...
}
```

#### 7.1.2.2.3 Variable cyclic RecurringEvents based on Time

Variable cyclic here means that the cycle time might be mode dependent and vary during runtime.

Thus, the AUTOSAR OS object 'ScheduleTable' likely implements *RecurringEvents* with variable cycles.

#### 7.1.2.2.4 RecurringEvents not directly based on Time

Since there are various use cases and therefore implementations possible, the implementation of this type of RecurringEvents will neither be specified nor documented.

### 7.1.2.3 SporadicEvents

#### 7.1.2.3.1 Concept

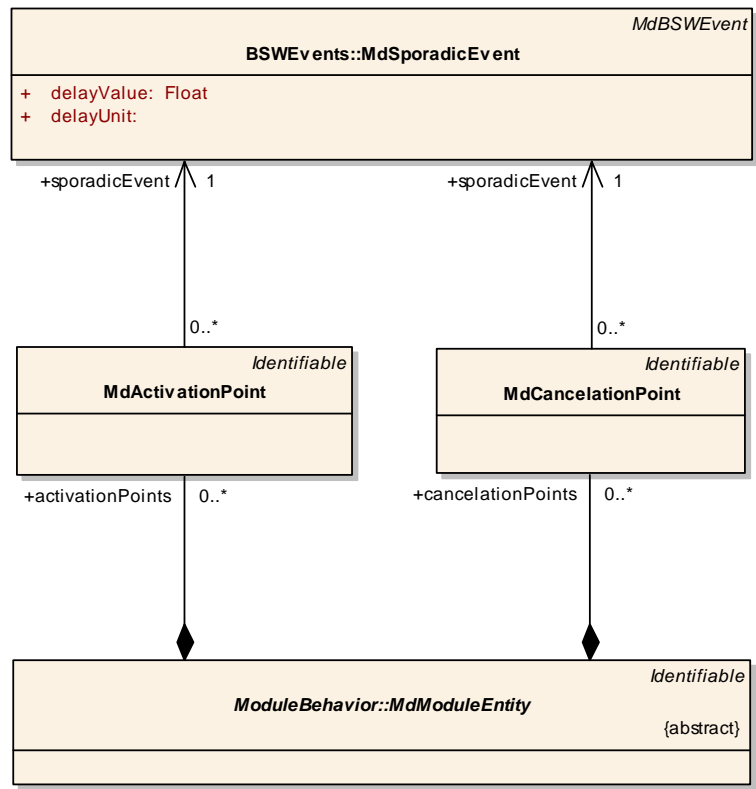
The concept of *SporadicEvents* expresses the ability to synchronize module entities. This comprises the decoupling of interrupt and task context as well as the starting of timeouts and connected actions.

Figure 9 shows the relation of *ModuleEntities* (code), *ActivationPoints*, *CancelationPoints* and *SporadicEvents*, Figure 8 the correlation of BSWEvents and schedulable functions (main processing function).

The class *SporadicEvents* shall contain an attribute, which denotes the minimum delay time (given in time units, tick, etc.) until a main processing function is started. Since the least possible delay is zero, *SporadicEvents* can express an immediate trigger, which in turn could start a corresponding main processing function immediately. The case where the value of the 'delay attribute' is greater than zero denotes a delayed trigger. Likewise, the trigger will activate the corresponding main processing function with a delay.

*ActivationPoints* are the means to refer to *SporadicEvents* within the code of a BSW module entity, whereas each *ActivationPoint* points exactly to one *SporadicEvent*. It is also possible that more than one *ActivationPoint* references a particular *SporadicEvent*.

Thus, the shown concept allows for hoisting different timeout timers at different points during execution, but activating the very same main processing function after their expiration.



**Figure 9: ActivationPoints and CancelationPoints in the context of SporadicEvents**

*CancelationPoints* are the counterparts of the *ActivationPoints*. Like *ActivationPoints* each *CancelationPoints* point exactly to one *SporadicEvent*. In case the corresponding *SporadicEvent* is not yet fired, it will be canceled.

### 7.1.2.3.2 Implementation

The implementation of *SporadicEvents*, *ActivationPoints* and *CancelationPoints* reflects their description: The service `SchM_ActMainFunction` corresponds to *ActivationPoints* (see [INTEGR039](#)). The service `SchM_CancelMainFunction` corresponds to *CancelationPoints* (see [INTEGR044](#)).

The implementation of the real *SporadicEvents* can vary depending on the desired semantic. This document will not specify particular solutions. Proper means to realize the activation of main processing functions might be the AUTOSAR OS services `ActivateTask` for immediate and `SetRelAlarm` or `SetAbsAlarm` for delayed activation. However, there might be other solutions, which use the AUTOSAR OS services differently.

Example 7.4: A main processing function of a BSW module shall explicitly be activated. The implementation of the calling BSW module exists as source code.

#### Implementation of the BSW module <ModulePrefix>:

<ModulePrefix>\_cfg.h:

```
#define TIMEOUT_EVENT_100MS      (1)

#define ACTIVATION_POINT_1      TIMEOUT_EVENT_100MS
```

<ModulePrefix>.c:

```
#include "<ModulePrefix>.h"
#include "<ModulePrefix>_cfg.h"
#include "SchM_<ModulePrefix>.h"

void
<ModulePrefix>_foo () {
    ...
    SchM_ActMainFunction_<ModulePrefix> (ACTIVATION_POINT_1);
    ...
}
```

#### Implementation alternative 1 of the BSW Scheduler module:

SchM\_<ModulePrefix>.h:

```
#define SchM_ActMainFunction_<ModulePrefix> (ActivationPoint) \
    ActMainFunction_<ModulePrefix>_ ## ActivationPoint

#define ActMainFunction_<ModulePrefix>_1 \
    ActivateTask (<ModulePrefix>_MainFunction_<Name>Task)
```

SchM.c:

```
#include "<ModulePrefix>.h"
#include "SchM.h"
#include "SchM_<ModulePrefix>.h"

TASK (<ModulePrefix>_MainFunction_<Name>Task) {
    <ModulePrefix>_MainFunction_<Name> ();
    TerminateTask ();
}
```

Implementation alternative 2 of the BSW Scheduler module:

SchM <ModulePrefix>.h:

```
extern boolean Flag_1;

#define SchM_ActMainFunction_<ModulePrefix> (ActivationPoint) \
    ActMainFunction_<ModulePrefix>_ ## ActivationPoint

#define ActMainFunction_<ModulePrefix>_1 \
    (Flag_1 = (TRUE), SCHM_E_OK)
```

SchM.c:

```
#include "<ModulePrefix>.h"
#include "SchM.h"
#include "SchM_<ModulePrefix>.h"

    boolean Flag_1;

SchM_Init () {
    Flag_1 = FALSE;
}

TASK (Task_Cycle10ms) {
    ...
    if (Flag_1) {
        <ModulePrefix>_MainFunction_<Name> ();
        Flag_1 = FALSE;
    }
    ...
    TerminateTask ();
}
```

Example 7.5: A main processing function shall be activated after a timeout has been expired. The implementation of the calling BSW module exists as object code.

Implementation of the BSW module <ModulePrefix>:

<ModulePrefix>\_cfg.h:

```
#define TIMEOUT_EVENT_10MS      ((uint8)0)
#define TIMEOUT_EVENT_20MS      ((uint8)1)

#define ACTIVATION_POINT_0      TIMEOUT_EVENT_10MS
#define ACTIVATION_POINT_1      TIMEOUT_EVENT_20MS

#define CANCELATION_POINT_0     TIMEOUT_EVENT_10MS
#define CANCELATION_POINT_1     TIMEOUT_EVENT_20MS
```

<ModulePrefix>.c

```
#include "<ModulePrefix>.h"
#include "<ModulePrefix>_cfg.h"
#include "SchM_<ModulePrefix>.h"

void
<ModulePrefix>_foo () {
    SchM_ReturnType status;

    ...
    status = SchM_ActMainFunction_<ModulePrefix> (ACTIVATION_POINT_0);
    ...
    status =
    SchM_CancelMainFunction_<ModulePrefix> (CANCELATION_POINT_0);
    ...
    /* error case, because not activated before */
    status =
    SchM_CancelMainFunction_<ModulePrefix> (CANCELATION_POINT_1);
    ...
}
```

Implementation of the BSW Scheduler module:

AUTOSAR OS configuration:

```
COUNTER SYSTEM_COUNTER {
    /* 1ms system counter */
    MAXALLOWEDVALUE = 65535;
    TICKSPERBASE = 1000000;
    MINCYCLE = 1;
};

ALARM ALARM_0 {
    COUNTER = SYSTEM_COUNTER;
    ACTION = ACTIVATETASK {
        TASK = <ModulePrefix>_MainFunction_<Name>Task;
    };
};
```

```
ALARM ALARM_1 {
    COUNTER = SYSTEM_COUNTER;
    ACTION = ACTIVATETASK {
        TASK = <ModulePrefix>_MainFunction_<Name>Task;
    };
};
```

#### SchM\_cfg.h:

```
#define TIMEOUT_10MS          10
#define TIMEOUT_20MS          20
```

#### SchM\_<ModulePrefix>.h:

```
SchM_ReturnType
SchM_ActMainFunction_<ModulePrefix> (uint8 ActivationPoint);

SchM_ReturnType
SchM_CancelMainFunction_<ModulePrefix> (uint8 CancelationPoint);

struct SchM_<ModulePrefix>_Activation {
    ALARM alarm;
    TickType tick;
};
```

#### SchM.c:

```
#include "<ModulePrefix>.h"
#include "SchM_cfg.h"
#include "SchM_<ModulePrefix>.h"

const struct SchM_<ModulePrefix>_Activation
    <ModulePrefix>_Activation[] =
{
    {ALARM_0, TIMEOUT_10MS},
    {ALARM_1, TIMEOUT_20MS}
};

SchM_ReturnType
SchM_ActMainFunction_<ModulePrefix> (uint8 ActivationPoint) {
    return SetRelAlarm (
        <ModulePrefix>_Activation [ActivationPoint].alarm,
        <ModulePrefix>_Activation [ActivationPoint].tick
    );
}

SchM_ReturnType
SchM_CancelMainFunction_<ModulePrefix> (uint8 CancelationPoint) {
    return CancelAlarm (
        <ModulePrefix>_Activation [CancelationPoint].alarm
    );
}
```

### 7.1.2.4 Main processing functions triggered by RecurringEvents and SporadicEvents

*RecurringEvents* and *SporadicEvents* might trigger the same main processing function.

In general, it has to be made sure that no side effects can occur. Therefore, the most likely implementation is that a task only contains the call to this main processing function. The task may then be activated cyclically (based on Time) or by a sporadic call of 'ActivateTask'.

*RecurringEvents*, which are not directly time based, are supported as long as the event is captured via an OS counter. For all other cases, the implementation of this type of *RecurringEvents* will neither be specified nor documented.

Example 7.6: A main processing function shall be activated by a time-based fixed cyclic *RecurringEvent* and a *SporadicEvent*. The implementation of the BSW module exists as source code.

Implementation of the BSW module <ModulePrefix>:

<ModulePrefix>\_cfg.h:

```
#define IMMEDIATE_TRIGGER_EVENT    ((uint8)0)

#define ACTIVATION_POINT_0         IMMEDIATE_TRIGGER_EVENT
```

<ModulePrefix>.c:

```
#include "<ModulePrefix>_cfg.h"
#include "SchM_<ModulePrefix>.h"

void
<ModulePrefix>_foo () {
    ...
    SchM_ActMainFunction_<ModulePrefix> (ACTIVATION_POINT_1);
    ...
}
```

Implementation of the BSW Scheduler module:

AUTOSAR OS configuration:

```
COUNTER SYSTEM_COUNTER {
    /* 1MS system counter */
    MAXALLOWEDVALUE = 65535;
    TICKSPERBASE = 1000000;
    MINCYCLE = 1;
};

ALARM <MODULEPREFIX>_TRANSMIT_ALARM {
    COUNTER = SYSTEM_COUNTER;
    ACTION = ACTIVATETASK {
        TASK = <ModulePrefix>_MainFunction_TransmitTask;
    };
};
```

```
};
```

SchM\_cfg.h:

```
#define START_TIME_10MS          0  
#define CYCLE_TIME_10MS        10
```

SchM\_<ModulePrefix>.h:

```
#define SchM_ActMainFunction_<ModulePrefix> (ActivationPoint) \  
    ActMainFunction_<ModulePrefix>_ ## ActivationPoint  
  
#define SchM_ActMainFunction_<ModulePrefix>_0 \  
    ActivateTask (<ModulePrefix>_MainFunction_TransmitTask)
```

SchM.c:

```
#include "<ModulePrefix>.h"  
#include "SchM.h"  
#include "SchM_cfg.h"  
#include "SchM_<ModulePrefix>.h"  
  
void  
SchM_Init () {  
    SetRelAlarm (<MODULEPREFIX>_TRANSMIT_ALARM,  
                START_TIME_10MS,  
                CYCLE_TIME_10MS);  
}  
  
TASK (<ModulePrefix>_MainFunction_TransmitTask) {  
    <ModulePrefix>_MainFunction_Transmit ();  
    TerminateTask ();  
}
```

## 7.2 Data consistency

### 7.2.1 Background

The access to shared data from concurrent threads of code execution can potentially cause data inconsistencies. One of these phenomena is known as 'lost update problem'.

As explained above, the specifications of the BSW modules do not define the ultimate execution context, but potentially concurrent threads of executions: main processing functions. Functions called from a main processing function run in its context. That means further that a service of a module will likely run in the context of another module main processing function, too. Since BSW modules are potentially implemented by different vendors, only the integrator of the entire BSW has the information to map the threads of execution (main processing functions) onto the execution context (OS object 'Tasks'). Thus, only the integrator of the BSW has the proper information to handle data consistency problems, correctly.

Although the implementer of a BSW module has a limited view on the problem, an implementation shall consider potential data consistency problems: by the used module interaction scheme, by the BSW Module Description and by the concept of ExclusiveAreas described below (see also Section 5.1.4).

In general, there are two types of shared resources possible: resources accessed *globally* and resources that accessed *locally*. The AUTOSAR BSW uses both types.

In case of *locally accessible* resources, external BSW module can only access the shared resources via an entry point of the module, i.e. APIs or callback functions. Thus, the BSW module encapsulates the resources. Although, the entry points of the module are accessible from the outside (from unknown execution contexts), the module implementer knows all entities (access points) that access the module internal shared resources. However, even in this case the module implementer cannot reason about a 'proper' locking scheme, because of the problems explained above.

In case of *globally accessible* resources, any code entity (of any BSW module) can access a resource directly via a pointer rather than via an entry point of a module. Thus, the BSW module that 'owns' the resource cannot even apply *any* locking scheme to provide data consistency.

AUTOSAR handles both types in different ways.

### 7.2.2 Built-in data consistency

For *globally accessible* resources, only the design of the possible interaction between BSW modules can ensure data consistency. Modules that access a shared resource shall run in the execution context of the 'owner'.

**INTEGR047:** For *globally accessible* resources, only the design of the possible interaction between BSW modules can ensure data consistency. Thus, modules that access a shared resource shall run in the execution context of the 'owner'.

## 7.2.3 ExclusiveAreas

### 7.2.3.1 Concept

For *locally accessible* resources, this document introduces the concept of *ExclusiveAreas*<sup>3</sup>. It expresses the (abstract) means to protect the concurrent access to module internal, but shared resources by different module entities.

**INTEGR093:** Implementation of BSW modules shall use the concept of *ExclusiveAreas* to protect locally accessible (shared) resources.

To address the problem correctly, this document distinguishes between the description of an *ExclusiveArea* as part of the BSW Module Description and its corresponding implementation.

See first the UML diagram of Figure 10: it shows the meta model of the *ExclusiveArea* concept.

An *ExclusiveArea* is part of the internal behavior of a BSW module and has a name for identification. There might be several shared resources per BSW module and so there are a number of differently named *ExclusiveAreas* (per module).

In addition, there are module entries (APIs, callbacks). They can be accessed from other modules and might need to access locally shared resources as well. The latter link is achieved via the meta model element *ModuleEntityCanEnterExclusiveArea*, if there is an association to a particular and named *ExclusiveArea*. In this case, the BSW module implementation is required to call a particular service *SchM\_Enter* (see [INTEGR032](#)) for entering the protection scheme and *SchM\_Exit* (see [INTEGR035](#)) for exiting the protection scheme. Note that both services distinguish the access to differently named *ExclusiveAreas* via a parameter.

---

<sup>3</sup> The term 'ExclusiveArea' has been adopted from the Software Component Template. In general, this concept is known by the term 'critical region'. The meta model of the *ExclusiveArea* has been adopted from the Software Component Template, too.

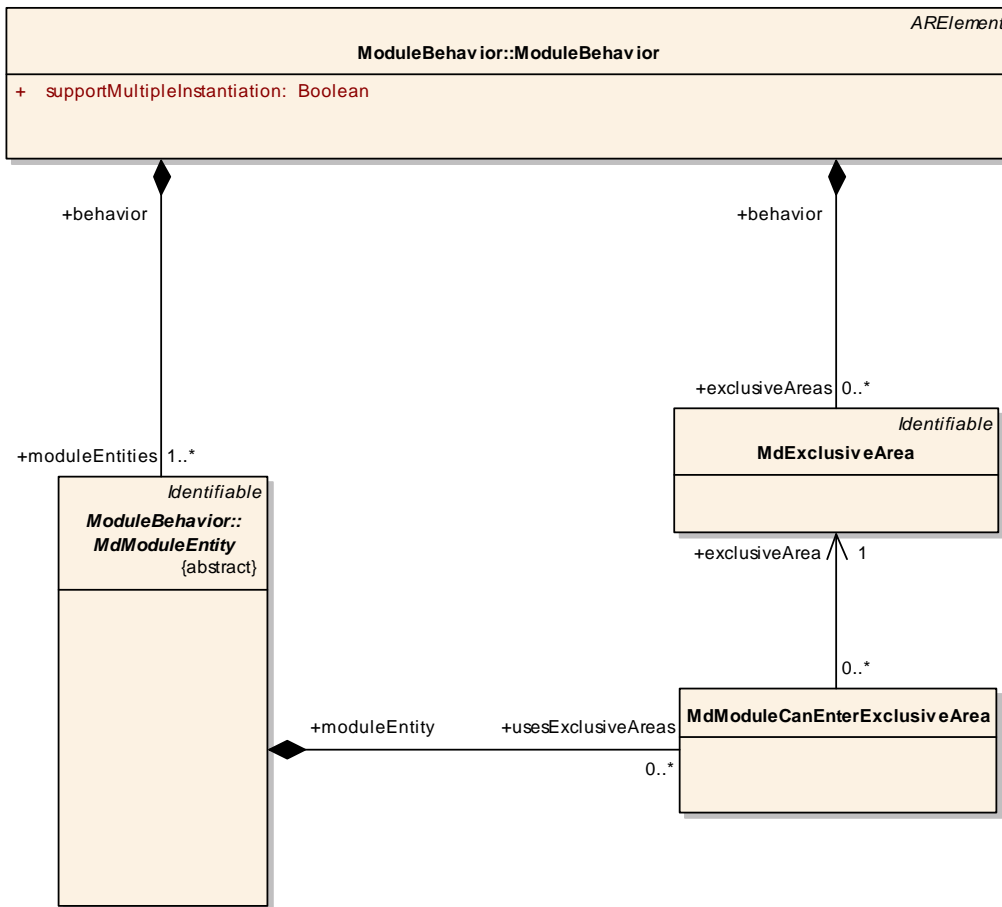


Figure 10: BSW module description template extract concerning *ExclusiveAreas*

### 7.2.3.2 Implementation

**INTEGR094:** The integrator shall implement the particular services `SchM_Enter` and `SchM_Exit`.

The implementation of the services `SchM_Enter` or `SchM_Exit` may differ depending on the context of the project, the implementation of the module or a particular shared resource. Therefore, this document will not specify or favor any solution. Possible implementations comprise

- the use of interrupt locking schemes provided by the AUTOSAR OS services like `DisableAllInterrupts`, `EnableAllInterrupts`
- the use of specific interrupt locking schemes provided by the BSW modules like `Can_DisableControllerInterrupts`, `Can_EnableControllerInterrupts`
- the use of AUTOSAR OS ‘Resources’ preventing priority inversion and deadlocks
- the dedicated sequential invocation of code entities that access the shared resources

There may be other implementations.

Example 7.7: The BSW module <ModulePrefix> shall assure data consistency for three different internal resources (buffer). The implementation of the BSW module <ModulePrefix> exists as source code.

Implementation of the BSW module <ModulePrefix>:

<ModulePrefix>\_cfg.h

```
#define EXCLUSIVE_AREA_0    ((uint8)0)
#define EXCLUSIVE_AREA_1    ((uint8)1)
#define EXCLUSIVE_AREA_2    ((uint8)2)

#define CAN_CONTROLLER_0    ((uint8)0)
```

<ModulePrefix>.c:

```
#include "<ModulePrefix>_cfg.h"
#include "SchM_<ModulePrefix>.h"

SchM_Enter_<ModulePrefix> (EXCLUSIVE_AREA_0);
<Access to buffer 0>
SchM_Exit_<ModulePrefix> (EXCLUSIVE_AREA_0);
...
...
SchM_Enter_<ModulePrefix> (EXCLUSIVE_AREA_1);
<Access to buffer 1>
SchM_Exit_<ModulePrefix> (EXCLUSIVE_AREA_1);
...
...
SchM_Enter_<ModulePrefix> (EXCLUSIVE_AREA_2);
<Access to buffer 2>
SchM_Exit_<ModulePrefix> (EXCLUSIVE_AREA_2);
...
...
```

Implementation of the BSW Scheduler module:

AUTOSAR OS configuration:

```
RESOURCE RESOURCE_2 {
}
```

SchM\_<ModulePrefix>.h:

```
#define SchM_Enter_<ModulePrefix> (ExclusiveArea) \
    SchM_Enter_<ModulePrefix>_ ## ExclusiveArea

#define SchM_Exit_<ModulePrefix> (ExclusiveArea) \
    SchM_Exit_<ModulePrefix>_ ## ExclusiveArea

#define SchM_Enter_<ModulePrefix>_0    DisableAllInterrupts
#define SchM_Exit_<ModulePrefix>_0    EnableAllInterrupts
```

```
#define SchM_Enter_<ModulePrefix>_1 \
    Can_DisableControllerInterrupts (CAN_CONTROLLER_0)
#define SchM_Exit_<ModulePrefix>_1 \
    Can_EnableControllerInterrupts (CAN_CONTROLLER_0)

#define SchM_Enter_<ModulePrefix>_2    GetResource (RESOURCE_2)
#define SchM_Exit_<ModulePrefix>_2    ReleaseResource (RESOURCE_2)
```

## 7.3 Error handling concept

### 7.3.1 Background

In general, the AUTOSAR Basic Software distinguishes between production errors and development errors (see [2]).

As explained above, the BSW Scheduler depicts more the integration of the AUTOSAR Basic Software than it is a 'real' BSW module. With this respect, the BSW Scheduler acts like a facade to hide the embedding of BSW modules into a particular infrastructure. Thereby, the particular infrastructure (layer) may be more or less distinctive, but may cause (production) errors. The BSW Scheduler itself as a facade will not, but will handle development errors.

### 7.3.2 Error handling in the context of AUTOSAR OS/OSEK

Since the AUTOSAR OS (based on OSEK) likely provides the main infrastructure for the AUTOSAR Basic Software, its error-handling concept shall also serve for the following considerations. Unfortunately, the standard does not cleanly differentiate between production and development errors, but defines the subsequent types:

- fatal errors (of OSEK)
- protection errors (of AUTOSAR OS)
- application errors

#### 7.3.2.1 Fatal errors

In case of fatal errors, the AUTOSAR/OSEK OS can no longer assume the correctness of its internal data and calls the centralized system shutdown via ShutdownOS (StatusType error). ShutdownOS then calls the hook routine ShutdownHook, if configured and the integrator is responsible to implement a 'proper' ShutdownHook (see [9] for further information)<sup>4</sup> handling the errors.

Fatal errors are assumed production errors of the AUTOSAR OS.

---

<sup>4</sup> Please note that this is not a specification item for this specification, because it refers to a likely but particular implementation of the BSW Scheduler.

### 7.3.2.2 Protection errors

In case of protection errors (timing or memory), the AUTOSAR OS provides the ability to react by the means of a `ProtectionHook`. The integrator needs to implement a 'proper' `ProtectionHook`.

Protection errors are assumed production errors by the OS.

### 7.3.2.3 Application errors

In case of application errors, the AUTOSAR/OSEK OS cannot execute the requested service correctly, but assumes the correctness of its internal data.

In general, OSEK OS signals erroneous states by return values, whereas the standard defines standard and extended mode values. OSEK refers to the standard mode values as 'warnings', because they not really signal errors. However, they signal the (potentially transient) status of an API request. AUTOSAR refers to this type of errors as *infrastructure errors*. In general, the 'extended mode values' corresponds to development errors.

#### 7.3.2.3.1 Development phase

OSEK support the development process by means of hooks: `ErrorHook`, `PreTaskHook` and `PostTaskHook`. The integrator is responsible for the implementation of the hooks, if configured. `PreTaskHook` and `PostTaskHook` are not integral part of production code.

#### 7.3.2.3.2 Production phase

During the production phase, the BSW Scheduler only forwards *infrastructure errors* to the respective BSW modules via return values of type `SchM_ReturnType` (see [INTEGR070](#)). The BSW Scheduler does not cause production errors by itself. If configured, `ErrorHook` is an integral part of production code.

## 7.3.3 Error handling of the BSW Scheduler

### 7.3.3.1 Production errors

There are no production errors for the BSW Scheduler specified.

### 7.3.3.2 Development errors

**INTEGR076:** Development error values are of type `uint8`.

<i>Type or error</i>	<i>Relevance</i>	<i>Related error code</i>	<i>Value [hex]</i>
API service called with wrong parameter	Development	SCHM_E_PARAM_CONFIG	0x01

**INTEGR085:** A particular implementation of BSW Scheduler module may specify additional (implementation specific) development errors.

**INTEGR077:** The detection of development errors is configurable (*ON/OFF*) at pre-compile time.

The switch *SCHM\_DEV\_ERROR\_DETECT* (see Chapter 10) shall activate or deactivate the detection of all development errors.

**INTEGR080:** The BSW Scheduler shall report detected development errors to the *Det\_ReportError* service of the Development Error Tracer (DET).

**INTEGR078:** The switch *SCHM\_DEV\_ERROR\_DETECT* shall also activate (ON) or deactivate (OFF) API parameter checking. Specification item [INTEGR076](#) and Chapter 8 specify details.

## 8 API specification

### 8.1 Imported types

#### 8.1.1 Standard types

This chapter lists all types included from the file Std\_Types.h:

- Std\_VersionInfoType

### 8.2 Type definitions

#### 8.2.1 SchM\_ReturnType

**INTEGR070:**

<b>Type:</b>	uint8	
	SCHM_E_OK = 0x00	No error occurred.
	SCHM_E_LIMIT = 0x04	An internal limit has been exceeded.
	SCHM_E_NOFUNC = 0x05	Requested service has not the desired effect.
	SCHM_E_STATE = 0x07	Requested service is already in use.
<b>Description:</b>	The values of this BSW Scheduler specific status type are aligned to the values of the AUTOSAR OS status types.	

#### 8.2.2 SchM\_ConfigType

**INTEGR083:**

<b>Type:</b>	Struct	
<b>Range:</b>	implementation specific structure	Structure to hold the module's configuration set. The contents of this data structure are implementation specific.
<b>Description:</b>	A pointer to such a structure is provided to the module's initialization routine for the purpose of configuration.	

### 8.3 Function definitions

#### 8.3.1 SchM\_Init

<b>Service name:</b>	SchM_Init
<b>Syntax:</b>	void SchM_Init (void)
<b>Service ID [hex]:</b>	0x00
<b>Sync/Async:</b>	synchronous

<b>Reentrancy:</b>	non reentrant
<b>Parameters (in):</b>	none
<b>Parameters (out):</b>	none
<b>Return value:</b>	none
<b>Description:</b>	<b>INTEGR023:</b> SchM_Init allocates and initializes resources used by the Scheduler Module. SchM_Init can also call OS services to trigger AUTOSAR OS 'Tasks'.
<b>Caveats:</b>	<b>INTEGR025:</b> The ECUStateManager EcuM shall call SchM_Init during the EcuM Init sequence II.
<b>Configuration:</b>	none

### 8.3.2 SchM\_Deinit

<b>Service name:</b>	SchM_Deinit
<b>Syntax:</b>	void SchM_Deinit (void)
<b>Service ID [hex]:</b>	0x01
<b>Sync/Async:</b>	synchronous
<b>Reentrancy:</b>	non reentrant
<b>Parameters (in):</b>	none none
<b>Parameters (out):</b>	none
<b>Return value:</b>	none
<b>Description:</b>	<b>INTEGR027:</b> SchM_Deinit finalizes and frees resources used by the Scheduler Module.
<b>Caveats:</b>	The ECUStateManager EcuM can call SchM_Deinit at EcuM GO OFF II.
<b>Configuration:</b>	none

### 8.3.3 SchM\_GetVersionInfo

<b>Service name:</b>	SchM_GetVersionInfo
<b>Syntax:</b>	void SchM_GetVersionInfo ( Std_VersionInfoType *versioninfo )
<b>Service ID [hex]:</b>	0x02
<b>Sync/Async:</b>	synchronous
<b>Reentrancy:</b>	non reentrant
<b>Parameters (in):</b>	none
<b>Parameters (out):</b>	versioninfo <b>INTEGR037:</b> The pointer versioninfo points to the memory location, which holds the version information of the module.
<b>Return value:</b>	none

<b>Description:</b>	<p><b>INTEGR030:</b> The service <code>SchM_GetVersionInfo</code> returns the version information of this module. The version information includes:</p> <ul style="list-style-type: none"> <li>- Module Id</li> <li>- Vendor Id</li> <li>- Vendor specific version numbers (BSW00407).</li> </ul> <p><b>INTEGR031:</b> This function shall be pre compile time configurable <code>On/Off</code> by the configuration parameter: <code>SCHM_VERSION_INFO_API</code></p> <p>Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.</p>
<b>Caveats:</b>	none
<b>Configuration:</b>	<code>SCHM_VERSION_INFO_API</code>

### 8.3.4 SchM\_Enter

<b>Service name:</b>	<code>SchM_Enter</code>	
<b>Syntax:</b>	<pre>void SchM_Enter_&lt;ModulePrefix&gt; (     uint8      Instance,     uint8      ExclusiveArea )</pre>	
<b>Service ID [hex]:</b>	0x03	
<b>Sync/Async:</b>	synchronous	
<b>Reentrancy:</b>	non reentrant	
<b>Parameters (in):</b>	Instance	<b>INTEGR051:</b> The parameter <code>Instance</code> of service <code>SchM_Enter_&lt;ModulePrefix&gt;</code> identifies a unique instance of the calling BSW module <code>&lt;ModulePrefix&gt;</code> .
	ExclusiveArea	<b>INTEGR052:</b> The service <code>SchM_Enter_&lt;ModulePrefix&gt;</code> can omit the parameter <code>Instance</code> , if there is only one instance of the BSW module <code>&lt;ModulePrefix&gt;</code> possible. <b>INTEGR053:</b> The parameter <code>ExclusiveArea</code> of service <code>SchM_Enter_&lt;ModulePrefix&gt;</code> identifies a unique internal resource of the BSW Module <code>&lt;ModulePrefix&gt;</code> .
<b>Parameters (out):</b>	none	
<b>Return value:</b>	none	
<b>Description:</b>	<b>INTEGR032:</b> A BSW module <code>&lt;ModulePrefix&gt;</code> invokes the service <code>SchM_Enter</code> to enter a module local exclusive area (also known as critical section).	
<b>Caveats:</b>	none	
<b>Configuration:</b>	<p>The configuration of the service <code>SchM_Enter</code> depends on the implementation of the BSW Scheduler Module. Because the implementation of the BSW Scheduler module belongs to the ECU integrators tasks, the implementation is mainly project specific.</p> <p><b>INTEGR033:</b> The service <code>SchM_Enter</code> expects a valid configuration of system resources in terms of OS objects. The BSW Scheduler shall use the AUTOSAR OS by default.</p>	

### 8.3.5 SchM\_Exit

<b>Service name:</b>	SchM_Exit
<b>Syntax:</b>	void SchM_Exit_<ModulePrefix> ( uint8        Instance, uint8        ExclusiveArea )
<b>Service ID [hex]:</b>	0x04
<b>Sync/Async:</b>	synchronous
<b>Reentrancy:</b>	non reentrant
<b>Parameters (in):</b>	<p>Instance      <b>INTEGR054:</b> The parameter Instance of SchM_Exit_&lt;ModulePrefix&gt; identifies a unique instance of the calling BSW module &lt;ModulePrefix&gt;.</p> <p>                 <b>INTEGR056:</b> The service SchM_Exit_&lt;ModulePrefix&gt; can omit the parameter Instance, if there is only one instance of the BSW module &lt;ModulePrefix&gt; possible.</p> <p>ExclusiveArea      <b>INTEGR055:</b> The parameter ExclusiveArea of SchM_Exit_&lt;ModulePrefix&gt; identifies a unique internal resource of the BSW Module &lt;ModulePrefix&gt; to be protected.</p>
<b>Parameters (out):</b>	none
<b>Return value:</b>	none
<b>Description:</b>	<b>INTEGR035:</b> The BSW module invokes the service SchM_Exit to exit an exclusive area (also known as critical section).
<b>Caveats:</b>	none
<b>Configuration:</b>	<p>The configuration of the service SchM_Exit depends on the implementation of the BSW Scheduler Module. Because the implementation of the BSW Scheduler module belongs to the ECU integrators tasks, the implementation is mainly project specific.</p> <p><b>INTEGR036:</b> The service SchM_Exit expects a valid configuration of system resources in terms of OS objects. The BSW Scheduler shall use the AUTOSAR OS by default.</p>

### 8.3.6 SchM\_ActMainFunction

<b>Service name:</b>	SchM_ActMainFunction
<b>Syntax:</b>	SchM_ReturnType SchM_ActMainFunction_<ModulePrefix> ( uint8        Instance, uint8        ActivationPoint )
<b>Service ID [hex]:</b>	0x05
<b>Sync/Async:</b>	synchronous
<b>Reentrancy:</b>	non reentrant

<b>Parameters (in):</b>	Instance	<p><b>INTEGR057:</b> The parameter Instance of SchM_ActMainFunction_&lt;ModulePrefix&gt; identifies a unique instance of the calling BSW module &lt;ModulePrefix&gt;.</p> <p><b>INTEGR058:</b> The service SchM_ActMainFunction_&lt;ModulePrefix&gt; can omit the parameter Instance, if there is only one instance of the BSW module &lt;ModulePrefix&gt; possible.</p>
	ActivationPoint	<p><b>INTEGR061:</b> The parameter ActivationPoint of SchM_ActMainFunction_&lt;ModulePrefix&gt; identifies the unique execution point, which invokes SchM_ActMainFunction_&lt;ModulePrefix&gt;.</p> <p>The BSW Module Description shall bind the ActivationPoint to a <i>SporadicEvent</i> containing the attribute:</p> <ul style="list-style-type: none"> <li>the minimum delay time (given in time units, ticks ...) until activation of a main processing function</li> </ul> <p>A <i>SporadicEvent</i> is then able to activate a main processing function.</p>
<b>Parameters (out):</b>	none	
<b>Return value:</b>	SCHM_E_OK	No error occurred.
	SCHM_E_LIMIT	To many task activation (of the task context of the main processing function)
	SCHM_E_STATE	ActivationPoint is already in use.
<b>Description:</b>	<p><b>INTEGR039:</b> The BSW module implementation invokes the SchM_ActMainFunction API to trigger the activation of a corresponding main processing function.</p>	
<b>Caveats:</b>	none	
<b>Configuration:</b>	<p>The configuration of the service SchM_ActMainFunction depends on the implementation of the BSW Scheduler Module. Because the implementation of the BSW Scheduler module belongs to the ECU integrators tasks, the implementation is mainly project specific.</p> <p><b>INTEGR064:</b> The service SchM_ActMainFunction expects a valid configuration of system resources in terms of OS objects. The BSW Scheduler shall use the AUTOSAR OS by default.</p>	

### 8.3.7 SchM\_CancelMainFunction

<b>Service name:</b>	SchM_CancelMainFunction
<b>Syntax:</b>	<pre>SchM_ReturnType SchM_CancelMainFunction_&lt;ModulePrefix&gt; (     uint8      Instance,     uint8      ActivationPoint ) </pre>
<b>Service ID [hex]:</b>	0x06
<b>Sync/Async:</b>	synchronous
<b>Reentrancy:</b>	non reentrant

<b>Parameters (in):</b>	Instance	<p><b>INTEGR059:</b> The parameter Instance of SchM_CancelMainFunction_&lt;ModulePrefix&gt; identifies a unique instance of the calling BSW module &lt;ModulePrefix&gt;.</p> <p><b>INTEGR060:</b> The service SchM_CancelMainFunction_&lt;ModulePrefix&gt; can omit the parameter Instance, if there is only one instance of the BSW module &lt;ModulePrefix&gt; possible.</p>
	ActivationPoint	<p><b>INTEGR062:</b> The parameter ActivationPoint of SchM_CancelMainFunction_&lt;ModulePrefix&gt; refers to the corresponding ActivationPoint of SchM_ActMainFunction_&lt;ModulePrefix&gt; to be canceled.</p>
<b>Parameters (out):</b>	none	
<b>Return value:</b>	SCHM_E_OK	No error occurred.
	SCHM_E_NOFUNC	Currently, there is no activation of a main processing function for the given ActivationPoint pending.
<b>Description:</b>	<p><b>INTEGR044:</b> The BSW module invokes the SchM_CancelMainFunction API to trigger the cancellation of the requested activation of a corresponding main processing function.</p>	
<b>Caveats:</b>	<p><b>INTEGR063:</b> The service SchM_CancelMainFunction has no effect for an ActivationPoint where the content of the 'delay attribute' is zero (0 time units, ticks ...), at all. In this case the service will return with value SCHM_E_NOFUNC.</p>	
<b>Configuration:</b>	<p>The configuration of the service SchM_CancelMainFunction depends on the implementation of the BSW Scheduler Module. Because the implementation of the BSW Scheduler module belongs to the ECU integrators tasks, the implementation is mainly project specific.</p> <p><b>INTEGR066:</b> The service SchM_CancelMainFunction expects a valid configuration of system resources in terms of OS objects. The BSW Scheduler shall use the AUTOSAR OS by default.</p>	

## 8.4 Call-back notifications

Not applicable

## 8.5 Scheduled functions

Not applicable

## 8.6 Expected Interfaces

This Chapter lists all interfaces required from other modules.

### 8.6.1 Mandatory Interfaces

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

<i>API function</i>	<i>Module</i>	<i>Description</i>
<ModulePrefix>_MainFunction_<...>	all	The BSW Scheduler Module needs to access all main processing functions of all modules. The BSW Scheduler Module implements or triggers their invocation.

### 8.6.2 Optional Interfaces

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

<i>API function</i>	<i>Module</i>	<i>Description</i>	<i>Configuration parameter (description see chapter 10)</i>
Det_ReportError	Det	Development error notification	SCHM_DEV_ERROR_DETECT

### 8.6.3 Configurable interfaces

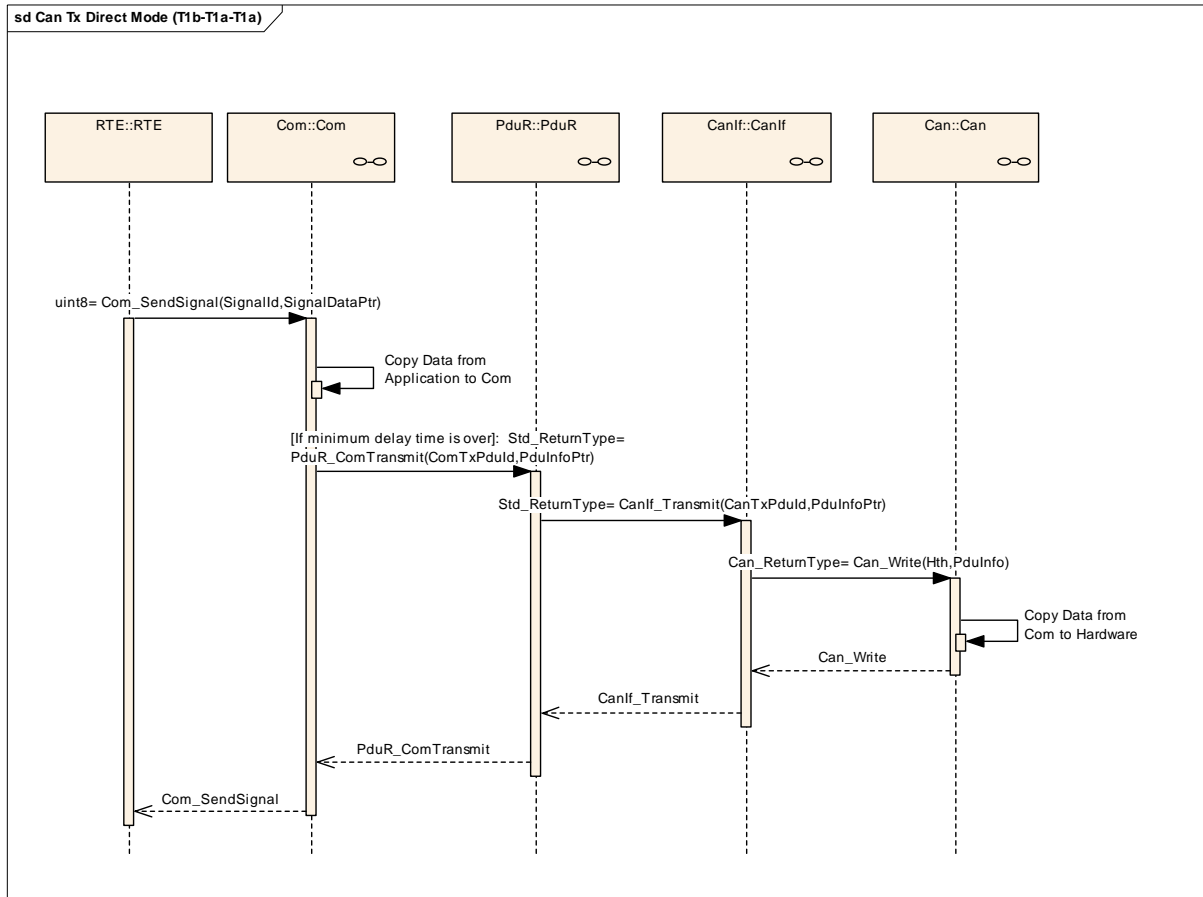
This Chapter lists all interfaces, which are potentially configurable. This is usually a callback function whereas the name is configurable.

Not applicable

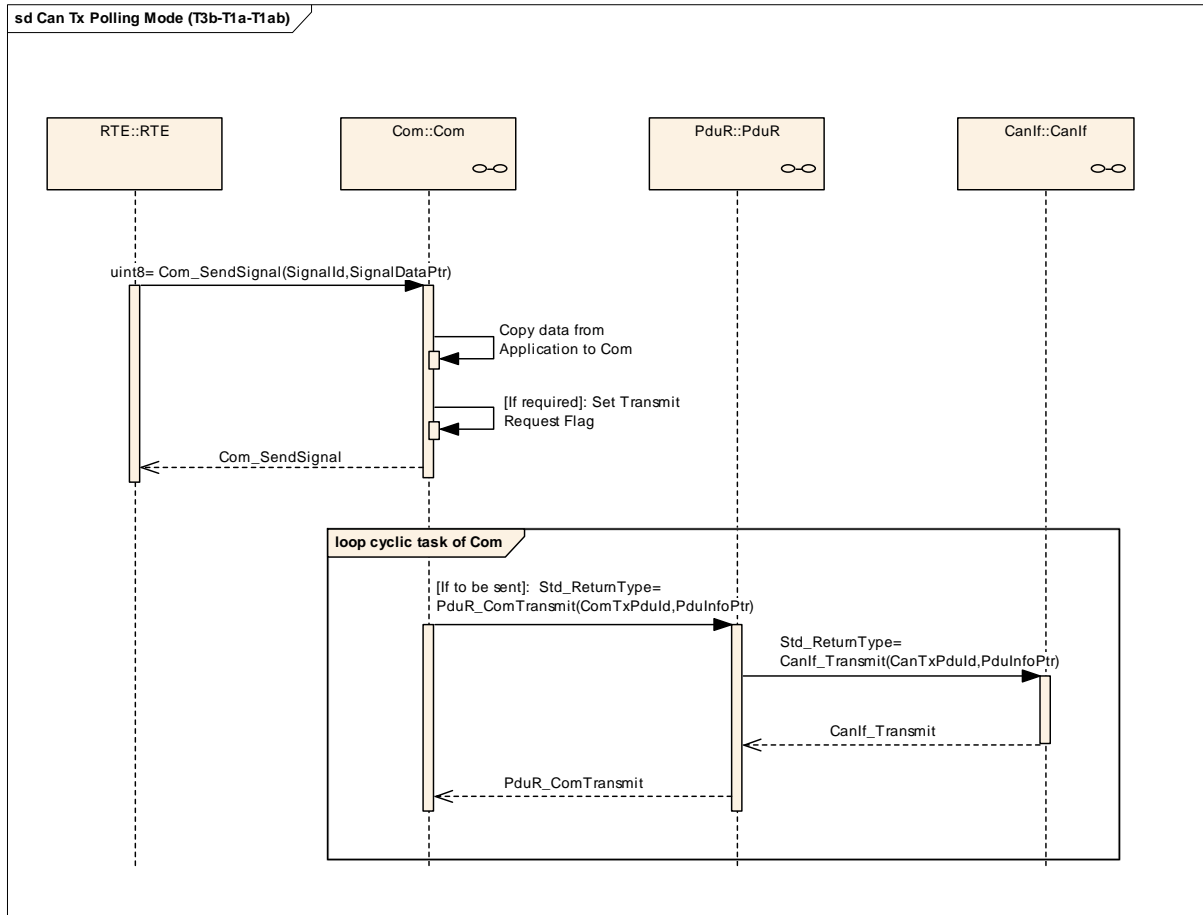
## 9 Sequence diagrams

With respect to data consistency, this section contains two sequence diagrams: One sequence diagram as an example for a globally accessible resource and one sequence diagram as an example for a locally accessible resource.

### Example: Globally accessible buffer:



**Example: Locally accessible buffer:**



## 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 Scheduler Module.

Chapter 10.3 specifies published information of the module Scheduler Module.

### 10.1 How to read this chapter

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

- AUTOSAR Layered Software [2]
- AUTOSAR ECU Configuration Specification [6]  
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 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

### 10.1.3 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.

## 10.2 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

Variant 1: This variant only allows pre-compile time parameters.

Variant 2: none

Variant 3: none

### 10.2.2 BSWSchedulerConfiguration

<b>SWS Item</b>	<b>INTEGR067:</b>
<b>Container Name</b>	BSWSchedulerConfiguration
<b>Description</b>	Top container - this container contains the entire configuration set of the BSW Scheduler.
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_DEV_ERROR_DETECT		
<b>Description</b>	Switches the Development Error Detection and Notification ON or OFF.		
<b>Type or Unit</b>	#define		
<b>Range</b>	STD_ON	Enabled	
	STD_OFF	Disabled	
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Scheduler Module		
<b>Dependency</b>	none		

<b>Name</b>	SCHM_VERSION_INFO_API		
<b>Description</b>	Switches the service SchM_GetVersionInfo ON or OFF.		
<b>Type or Unit</b>	#define		
<b>Range</b>	STD_ON	Enabled	
	STD_OFF	Disabled	
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Scheduler Module		
<b>Dependency</b>	none		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
MainFunctionMapping	1..*	BSW Scheduler Module (per ECU), BSW Module Description <sup>5</sup>
ModuleConfiguration	0..*	for each BSW module, BSW Module Description

### 10.2.3 MainFunctionMapping

<b>SWS Item</b>	<b>INTEGR068:</b>		
<b>Container Name</b>	MainFunctionMapping		
<b>Description</b>	Maps a MainFunction onto one OS Task		
<b>Configuration Parameters</b>			

<b>Name</b>	SCHM_MAINFUNCTION_REF		
<b>Description</b>	The parameter depicts the reference to the MainFunction.		
<b>Type or Unit</b>	Reference to MainFunction		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Scheduler Module		
<b>Dependency</b>	None		

<b>Name</b>	SCHM_MAPPED_TO_TASK		
<b>Description</b>	The parameter depicts the reference to the OS Task the MainFunction is mapped to.		
<b>Type or Unit</b>	Reference to AUTOSAR OS Task		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Scheduler Module, MainFunction(s) of the BSW modules		
<b>Dependency</b>	The AUTOSAR OS Task has already been available and is configured. Therefore, the configuration attributes of the task do not need to mention again.		

<b>Name</b>	SCHM_POSITION_IN_TASK		
<b>Description</b>	The parameter depicts the specific Position of the MainFunction within the OS Task.		
<b>Type or Unit</b>	Integer		
<b>Range</b>	0..255		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	AUTOSAR OS Task		
<b>Dependency</b>	The dependencies between the main processing functions mapped to the same task have been resolved by the configuration (process) beforehand. The result of this resolution is the order of the main processing functions within this task.		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
ScheduleTableMapping	0..*	--

## 10.2.4 ScheduleTableMapping

<b>SWS Item</b>	<b>INTEGR095:</b>
<b>Container Name</b>	ScheduleTableMapping
<b>Description</b>	Assigns a MainFunction to a OS Schedule Table
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_SCHEDULETABLE_REF		
<b>Description</b>	The parameter depicts the reference to the OS ScheduleTable the MainFunction is mapped to.		
<b>Type or Unit</b>	Reference to AUTOSAR OS ScheduleTable		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Scheduler Module, MainFunction(s) of the BSW modules		
<b>Dependency</b>	The AUTOSAR OS ScheduleTable has already been available and is configured. Therefore, the configuration attributes of the task do not need to mention again.		

<b>Name</b>	SCHM_MODE_REF		
<b>Description</b>	The parameter depicts the reference to a particular mode (not specified further, yet), this particular ScheduleTable is valid.		
<b>Type or Unit</b>	Reference to a particular mode		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Scheduler Module, MainFunction(s) of the BSW modules		
<b>Dependency</b>	The mode has already been available and is configured. Therefore, the configuration attributes of the task do not need to mention again.		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
none	--	--

### 10.2.5 ModuleConfiguration

<b>SWS Item</b>	<b>INTEGR087:</b>
<b>Container Name</b>	ModuleConfiguration
<b>Description</b>	Contains module specific configuration items
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_MODULE_LITERAL (<ModulePrefix>)		
<b>Description</b>	The parameter depicts a symbolic name (a literal) for the unambiguous identification of a BSW module; in fact it is the <ModulePrefix> specified for each BSW module in the Basic Software Module List (see [1]).		
<b>Type or Unit</b>	String		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW		
<b>Dependency</b>	--		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
InstanceConfiguration	1..*	-

## 10.2.6 InstanceConfiguration

<b>SWS Item</b>	<b>INTEGR088:</b>
<b>Container Name</b>	InstanceConfiguration
<b>Description</b>	Contains instance specific configuration items
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_INSTANCE_LITERAL		
<b>Description</b>	The parameter depicts a symbolic name for the unambiguous identification of a BSW module instance.		
<b>Type or Unit</b>	String		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module		
<b>Dependency</b>	SCHM_INSTANCE_ID		

<b>Name</b>	SCHM_INSTANCE_ID		
<b>Description</b>	The parameter depicts a value for the unambiguous identification of a BSW module instance.		
<b>Type or Unit</b>	uint8		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module		
<b>Dependency</b>	SCHM_INSTANCE_LITERAL		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
ExclusiveArea-Configuration	0..*	For each <i>ExclusiveArea</i> in the context of each BSW module.
ActivationPoint-Configuration	0..*	For each <i>ActivationPoint</i> in the context of each BSW module.
CancelationPoint-Configuration	0..*	For each <i>CancelationPoint</i> in the context of each BSW module.
SporadicEvent-Configuration	0..*	For each <i>SporadicEvent</i> in the context of each BSW module.

### 10.2.7 ExclusiveAreaConfiguration

<b>SWS Item</b>	<b>INTEGR086:</b>
<b>Container Name</b>	ExclusiveAreaConfiguration
<b>Description</b>	Contains configuration items of critical sections ( <i>ExclusiveArea</i> ) in the context of a BSW module
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_EXCLUSIVE_AREA_LITERAL		
<b>Description</b>	The parameter depicts a symbolic name for the unambiguous identification of an <i>ExclusiveArea</i> .		
<b>Type or Unit</b>	String		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	SCHM_EXCLUSIVE_AREA_ID		

<b>Name</b>	SCHM_EXCLUSIVE_AREA_ID		
<b>Description</b>	The parameter depicts a value for the unambiguous identification of an <i>ExclusiveArea</i> .		
<b>Type or Unit</b>	uint8		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	SCHM_EXCLUSIVE_AREA_LITERAL		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
none	--	--

### 10.2.8 ActivationPointConfiguration

<b>SWS Item</b>	<b>INTEGR089:</b>
<b>Container Name</b>	ActivationPointConfiguration
<b>Description</b>	Contains configuration items of <i>ActivationPoints</i> in the context of a BSW module
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_ACTIVATION_LITERAL		
<b>Description</b>	The parameter depicts a symbolic name for the unambiguous identification of an <i>ActivationPoint</i> .		
<b>Type or Unit</b>	String		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	SCHM_ACTIVATION_ID		

<b>Name</b>	SCHM_ACTIVATION_ID		
<b>Description</b>	The parameter depicts a value for the unambiguous identification of an <i>ActivationPoint</i> .		
<b>Type or Unit</b>	uint8		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	SCHM_ACTIVATION_LITERAL		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
none	--	--

### 10.2.9 CancelationPointConfiguration

<b>SWS Item</b>	<b>INTEGR090:</b>
<b>Container Name</b>	CancelationPointConfiguration
<b>Description</b>	Contains configuration items of CancelationPoints in the context of a BSW module
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_CANCELATION_LITERAL		
<b>Description</b>	The parameter depicts a symbolic name for the unambiguous identification of a <i>CancelationPoint</i> .		
<b>Type or Unit</b>	String		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	SCHM_CANCELATION_ID		

<b>Name</b>	SCHM_CANCELATION_ID		
<b>Description</b>	The parameter depicts a value for the unambiguous identification of a <i>CancelationPoint</i> .		
<b>Type or Unit</b>	uint8		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	SCHM_CANCELATION_LITERAL		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
none	--	--

### 10.2.10 SporadicEventConfiguration

<b>SWS Item</b>	<b>INTEGR069:</b>
<b>Container Name</b>	SporadicEventConfiguration
<b>Description</b>	Contains configuration items of <i>SporadicEvents</i>
<b>Configuration Parameters</b>	

<b>Name</b>	SCHM_SPORADIC_LITERAL		
<b>Description</b>	The parameter depicts a symbolic name for the unambiguous identification of a <i>SporadicEvent</i> .		
<b>Type or Unit</b>	String		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	X	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	BSW Module instance		
<b>Dependency</b>	none		

<b>Name</b>	DELAY_TIME		
<b>Description</b>	The parameter depicts the minimum delay time until a main processing functions shall be activated.		
	This configuration parameter is an input for integration. It represents the base for the real, implementation specific values.		
<b>Type or Unit</b>	any physical unit representing a delay time		
<b>Range</b>	--		
<b>Configuration Class</b>	<b>Pre-compile</b>	x	Variant 1
	<b>Link time</b>	--	--
	<b>Post Build</b>	--	--
<b>Scope</b>	per <i>SporadicEvent</i>		
<b>Dependency</b>	none		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
none	--	--

### 10.3 Published Information

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

<i>SWS Item</i>		<b>INTEGR045:</b>
<i>Information elements</i>		
<i>Information element name</i>	<i>Type / Range</i>	<i>Information element description</i>
SCHM_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
SCHM_MODULE_ID	#define/ uint8	Module ID of this module from Module List
SCHM_AR_MAJOR_VERSION	#define/ uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
SCHM_AR_MINOR_VERSION	#define/ uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
SCHM_AR_PATCH_VERSION	#define/ uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
SCHM_SW_MAJOR_VERSION	#define/ uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
SCHM_SW_MINOR_VERSION	#define/ uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
SCHM_SW_PATCH_VERSION	#define/ uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

## **11 Changes to Release 1**

### **11.1 Deleted SWS Items**

Not applicable

### **11.2 Replaced SWS Items**

Not applicable

### **11.3 Changed SWS Items**

Not applicable

### **11.4 Added SWS Items**

Not applicable