Document Title	Specification of Compiler Abstraction
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	051
Document Classification	Standard
Document Status	Final
Part of AUTOSAR Release	4.2.1

Document Change History

Release	Changed by	Change Description
4.2.1	AUTOSAR Release Management	• The compiler symbol definitions are not allowed to contain any value behind the symbol • Rework the document structure in order to follow TMPS_SRS_SWS and replace hardcoded diagrams with artifacts • Remove all MISRA/ C/ C++ related statements and references • Correct the unresolved references that point in SRS_BSWGeneral
4.1.2	AUTOSAR Release Management	• Editorial changes • Removed chapter(s) on change documentation
4.1.1	AUTOSAR Administration	• Added abstraction macro CONSTP2FUNC for a constant pointer to a function • Improved consistency to Memory Mapping (several MemMap.h files) • Reworked Configuration Specification
4.0.3	AUTOSAR Administration	• Added macros ‚FUNC_P2CONST‘ and ‘FUNC_P2VAR’ • Added pointer class ‘REGSPACE’ (for register access) • Updated the compiler symbols list
3.1.5	AUTOSAR Administration	• Put more emphasize on SwComponentType’s name in SWS_COMPILER_00054, COMPILER044 • Corrected compiler used in the example (chapter 7.1.5) • Corrected include structure in the example (chapter 7.1.5)

Document Change History		
Release	**Changed by**	**Change Description**
3.1.4	AUTOSAR Administration	• Compiler Abstraction has been extended to be suitable for Software Components • "STATIC" declaration keyword has been removed • The declaration keyword "LOCAL_INLINE" has been added for implementation of "static inline"-functions • Legal disclaimer revised
3.1.1	AUTOSAR Administration	• Legal disclaimer revised
3.0.1	AUTOSAR Administration	• Keyword "_STATIC_" has been renamed to "STATIC" • Keyword "_INLINE_" has been renamed to "INLINE" • Keyword "TYPEDEF" has been added as empty memory qualifier for use in type definitions • Document meta information extended • Small layout adaptations made
2.1.15	AUTOSAR Administration	• Add: COMPILER058 • Add: COMPILER057 • Change: SWS_COMPILER_00040 • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
2.0	AUTOSAR Administration	• Initial Release

Disclaimer

This specification and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only.
For any other purpose, no part of the specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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

This document specifies macros for the abstraction of compiler specific keywords used for addressing data and code within declarations and definitions.

Mainly compilers for 16-bit platforms (e.g. Cosmic and Metrowerks for S12X or Tasking for ST10) are using special keywords to cope with properties of the microcontroller architecture caused by the limited 16 bit addressing range. Features like paging and extended addressing (to reach memory beyond the 64k border) are not chosen automatically by the compiler, if the memory model is not adjusted to ‘large’ or ‘huge’. The location of data and code has to be selected explicitly by special keywords. Those keywords, if directly used within the source code, would make it necessary to port the software to each new microcontroller family and would prohibit the requirement of platform independency of source code.

If the memory model is switched to ‘large’ or ‘huge’ by default (to circumvent these problems) the project will suffer from an increased code size.

This document specifies a three-step concept:

1. The file Compiler.h provides macros for the encapsulation of definitions and declarations.

2. Each single module has to distinguish between at least the following different memory classes and pointer classes. Each of these classes is represented by a define (e.g. EEP_CODE).

3. The file Compiler_Cfg.h allows to configure these defines with the appropriate compiler specific keywords according to the modules description and memory set-up of the build scenario.

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.

Acronym:	***Description:***
Large, huge	Memory model configuration of the microcontroller's compiler. By default, all access mechanisms are using extended/paged addressing. Some compilers are using the term 'huge' instead of 'far'.
Tiny, small	Memory model configuration of the microcontroller's compiler. By default, all access mechanisms are using normal addressing. Only data and code within the addressing range of the platform's architecture is reachable (e.g. 64k on a 16 bit architecture).
far	Compiler keyword for extended/paged addressing scheme (for data and code that may be outside the normal addressing scheme of the platform's architecture).
near	Compiler keyword for normal addressing scheme (for data and code that is within the addressing range of the platform's architecture).

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf

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

[3] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Cosmic C Cross Compiler User's Guide for Motorola MC68HC12,V4.5

[6] ARM ADS compiler manual

[7] GreenHills MULTI for V850 V4.0.5:
Building Applications for Embedded V800, V4.0, 30.1.2004

[8] TASKING for ST10 V8.5:
C166/ST10 v8.5 C Cross-Compiler User's Manual, V5.16
C166/ST10 v8.5 C Cross-Assembler, Linker/Locator, Utilities User's Manual, V5.16

[9] Wind River (Diab Data) for PowerPC Version 5.2.1:
Wind River Compiler for Power PC - Getting Started, Edition 2, 8.5.2004
Wind River Compiler for Power PC - User's Guide, Edition 2, 11.5.2004

[10] TASKING for TriCore TC1796 V2.0R1:
TriCore v2.0 C Cross-Compiler, Assembler, Linker User's Guide, V1.2

[11] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25):
Motorola HC12 Assembler, 2.6.2004
Motorola HC12 Compiler, 2.6.2004
Smart Linker, 2.4.2004

[12] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

[13] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [12] (SWS BSW General), which is also valid for Compiler Abstraction.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Compiler Abstraction.

4 Constraints and assumptions

4.1 Limitations

During specification of abstraction and validation of concept, the compilers listed in chapter 3.1 have been considered. If any other compiler requires keywords that cannot be mapped to the mechanisms described in this specification this compiler will not be supported by AUTOSAR. In this case, the compiler vendor has to adapt its compiler.

If the physically existing memory is larger than the logically addressable memory in either code space or data space and more than the logically addressable space is used, logical addresses have to be reused. The C language (and other languages as well) can not cope with this situation.

4.2 Applicability to car domains

No restrictions.

4.3 Applicability to safety related environments

No restrictions. The compiler abstraction file does not implement any functionality, only symbols and macros.

5 Dependencies to other modules

[SWS_COMPILER_00048]「 The SWS Compiler Abstraction is applicable for each AUTOSAR basic software module and application software components. Therefore, the implementation of the memory class (memclass) and pointer class (ptrclass) macro parameters (see SWS_COMPILER_00040) shall fulfill the implementation and configuration specific needs of each software module in a specific build scenario. 」(SRS_BSW_00328, SRS_BSW_00384)

5.1 File structure

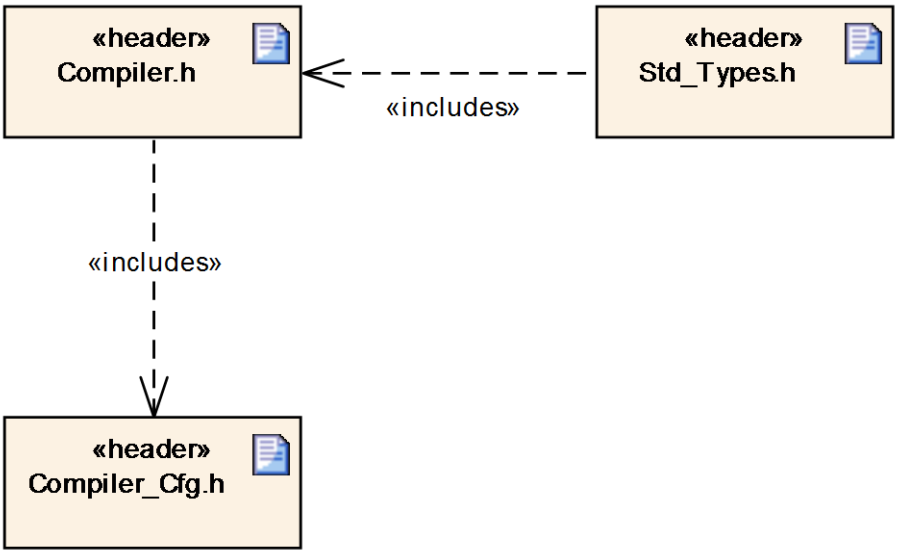

Figure 1: Include structure of Compiler.h

The following notes shall describe the connections to modules, which are indirectly linked to each other.

Note 1: The compiler abstraction is used to configure the reachability of elements (pointers, variables, function etc.).

Note 2: The memory mapping is used to perform the sectioning of memory. The user can define sections for optimizing the source code.

Note 3: The linker settings are responsible with the classification which elements are assigned to which memory section.

6 Requirements traceability

Document: AUTOSAR requirements on Basic Software, general

Requirement	***Satisfied by***
[SRS_BSW_00003] Version identification	SWS_COMPILER_00001
[SRS_BSW_00300] Module naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00301] Limit imported information	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00302] Limit exported information	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00304] AUTOSAR integer data types	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00305] Self-defined data types naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00306] Avoid direct use of compiler and platform specific keywords	supported by: COMPILER001, COMPILER006, SWS_COMPILER_00010, SWS_COMPILER_00012, COMPILER013, COMPILER015, COMPILER023, COMPILER026, COMPILER031, COMPILER032, COMPILER033, SWS_COMPILER_00035, SWS_COMPILER_00036, COMPILER039, COMPILER044, COMPILER046
[SRS_BSW_00307] Global variables naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00308] Definition of global data	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00309] Global data with read-only constraint	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00310] API naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00312] Shared code shall be reentrant	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00314] Separation of interrupt frames and service routines	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00318] Format of module version numbers	SWS_COMPILER_00001
[SRS_BSW_00321] Enumeration of module version numbers	SWS_COMPILER_00001
[SRS_BSW_00323] API parameter checking	Not applicable (Compiler Abstraction is not a BSW module)
[BSW00324] Do not use HIS I/O Library	Not applicable (non-functional requirement)
[SRS_BSW_00325] Runtime of interrupt service routines	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00326] Transition from ISRs to OS tasks	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00327] Error values naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00328] Avoid duplication of code	supported by: SWS_COMPILER_00048
[SRS_BSW_00329] Avoidance of generic interfaces	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00330] Usage of macros / inline functions instead of functions	Not applicable (Compiler Abstraction is not a BSW module)

Requirement	*Satisfied by*
[SRS_BSW_00331] Separation of error and status values	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00333] Documentation of callback function context	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00334] Provision of XML file	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00335] Status values naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00336] Shutdown interface	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00337] Classification of errors	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00338] Detection and Reporting of development errors	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00339] Reporting of production relevant error status	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00341] Microcontroller compatibility documentation	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00342] Usage of source code and object code	Not applicable (non-functional requirement)
[SRS_BSW_00343] Specification and configuration of time	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00344] Reference to link-time configuration	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00345] Pre-compile-time configuration	Chapter 10.2.1
[SRS_BSW_00346] Basic set of module files	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00347] Naming separation of different instances of BSW drivers	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00348] Standard type header	SWS_COMPILER_00003, SWS_COMPILER_00004, SWS_COMPILER_00052
[SRS_BSW_00350] Development error detection keyword	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00353] Platform specific type header	Not applicable (Compiler Abstraction is the C-language extension header)
[SRS_BSW_00355] Do not redefine AUTOSAR integer data types	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00357] Standard API return type	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00358] Return type of init() functions	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00359] Return type of callback functions	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00360] Parameters of callback functions	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00361] Compiler specific language extension header	SWS_COMPILER_00003, SWS_COMPILER_00004
[SRS_BSW_00369] Do not return development error codes via API	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00370] Separation of callback interface from API	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00371] Do not pass function pointers via API	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00373] Main processing function naming convention	Not applicable (Compiler Abstraction is not a BSW module)

Requirement	*Satisfied by*
[SRS_BSW_00374] Module vendor identification	SWS_COMPILER_00001
[SRS_BSW_00375] Notification of wake-up reason	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00376] Return type and parameters of main processing functions	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00377] Module specific API return types	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00378] AUTOSAR boolean type	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00379] Module identification	SWS_COMPILER_00001
[SRS_BSW_00380] Separate C-Files for configuration parameters	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00381] Separate configuration header file for pre-compile time parameters	SWS_COMPILER_00052
[SRS_BSW_00383] List dependencies of configuration files	Figure 1: Include structure of Compiler.h
[SRS_BSW_00384] List dependencies to other modules	SWS_COMPILER_00048
[SRS_BSW_00385] List possible error notifications	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00386] Configuration for detecting an error	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00387] Specify the configuration class of callback function	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00388] Introduce containers	Chapter 10.2
[SRS_BSW_00389] Containers shall have names	COMPILER044
[SRS_BSW_00390] Parameter content shall be unique within the module	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00391] Parameter shall have unique names	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00392] Parameters shall have a type	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00393] Parameters shall have a range	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00394] Specify the scope of the parameters	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00395] List the required parameters (per parameter)	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00396] Configuration classes	COMPILER044
[SRS_BSW_00397] Pre-compile-time parameters	COMPILER044
[SRS_BSW_00398] Link-time parameters	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00399] Loadable Post-build time parameters	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00004] Version check	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00400] Selectable Post-build time parameters	Not applicable (Compiler Abstraction is specific per build scenario)

Requirement	*Satisfied by*
[SRS_BSW_00401] Documentation of multiple instances of configuration parameters	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00402] Published information	SWS_COMPILER_00001
[SRS_BSW_00404] Reference to post build time configuration	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00405] Reference to multiple configuration sets	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00406] Check module initialization	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00407] Function to read out published parameters	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00408] Configuration parameter naming convention	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00409] Header files for production code error IDs	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00410] Compiler switches shall have defined values	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00411] Get version info keyword	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00412] Separate H-File for configuration parameters	SWS_COMPILER_00052
[SRS_BSW_00413] Accessing instances of BSW modules	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00414] Parameter of init function	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00415] User dependent include files	Not applicable (non-functional requirement)
[SRS_BSW_00416] Sequence of Initialization	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00417] Reporting of Error Events by Non-Basic Software	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00419] Separate C-Files for pre-compile time configuration parameters	Not applicable (Compiler Abstraction is specific per build scenario)
[BSW00420] Production relevant error event rate detection	Not applicable (Compiler Abstraction is not a BSW module)
[BSW00421] Reporting of production relevant error events	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00422] Debouncing of production relevant error status	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00424] BSW main processing function task allocation	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00425] Trigger conditions for schedulable objects	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00426] Exclusive areas in BSW modules	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00427] ISR description for BSW modules	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00428] Execution order dependencies of main processing functions	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00429] Restricted BSW OS functionality access	Not applicable (Compiler Abstraction is not a BSW module)

Requirement	*Satisfied by*
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00433] Calling of main processing functions	Not applicable (Compiler Abstraction is not a BSW module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00005] No hard coded horizontal interfaces within MCAL	Not applicable (non-functional requirement)
[SRS_BSW_00006] Platform independency	supported by: COMPILER001, COMPILER006, SWS_COMPILER_00010, SWS_COMPILER_00012, COMPILER013, COMPILER015, COMPILER023, COMPILER026, COMPILER031, COMPILER032, COMPILER033, SWS_COMPILER_00035, SWS_COMPILER_00036, COMPILER039, COMPILER044, COMPILER046
[SRS_BSW_00007] HIS MISRA C	Not applicable (Compiler Abstraction is the C-language extension header)
[SRS_BSW_00009] Module User Documentation	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00010] Memory resource documentation	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00101] Initialization interface	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00158] Separation of configuration from implementation	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00159] Tool-based configuration	Chapter 11.2.2
[SRS_BSW_00160] Human-readable configuration data	COMPILER044
[SRS_BSW_00161] Microcontroller abstraction	Not applicable (non-functional requirement)
[SRS_BSW_00162] ECU layout abstraction	Not applicable (non-functional requirement)
[SRS_BSW_00164] Implementation of interrupt service routines	Not applicable (non-functional requirement)
[SRS_BSW_00167] Static configuration checking	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00168] Diagnostic Interface of SW components	Not applicable (Compiler Abstraction is not a BSW module)
[SRS_BSW_00170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00171] Configurability of optional functionality	Not applicable (Compiler Abstraction is specific per build scenario)
[SRS_BSW_00172] Compatibility and documentation of scheduling strategy	Not applicable (Compiler Abstraction is not a BSW module)

7 Functional specification

7.1 General behavior

[SWS_COMPILER_00003] 「 For each compiler and platform an own compiler abstraction has to be provided. 」(SRS_BSW_00348, SRS_BSW_00361)

7.1.1 List of Compiler symbols

[SWS_COMPILER_00012]「 The following table defines target compiler symbols according to SWS_COMPILER_00010. For each compiler supported by AUTOSAR a symbol has to be defined.」(SRS_BSW_00306, SRS_BSW_00006)

Platform	***Compiler***	***Compiler symbol***
S12X	Code Warrior	_CODEWARRIOR_C_S12X_
S12X	Cosmic	_COSMIC_C_S12X_
TC1796/ TC1766	Tasking	_TASKING_C_TRICORE_
ST10	Tasking	_TASKING_C_ST10_
ST30	ARM Developer Suite	_ADS_C_ST30
V850	Greenhills	_GREENHILLS_C_V850_
MPC5554	Diab Data	_DIABDATA_C_ESYS_
TMS470	Texas Instruments	_TEXAS_INSTRUMENTS_C_TMS470_
ARM	Texas Instruments	_TEXAS_INSTRUMENTS_C_ARM_

Note: In order to avoid incompatibilities and/ or inconsistencies, the compiler symbol definitions are not allowed to contain any value behind the symbol.

7.1.2 Requirements on implementations using compiler abstraction

[SWS_COMPILER_00040] 「 Each AUTOSAR software module and application software component shall support the distinction of at least the following different memory classes and pointer classes.

It is allowed to add module specific memory classes and pointer classes as they are mapped and thus are configurable within the Compiler_Cfg.h file.

<PREFIX> is

- composed according <snp>[_<vi>_<ai>] for basic software modules where
  - <snp> is the *Section Name Prefix* which shall be the `BswModuleDescription`'s `shortName` converted in upper case letters if no SectionNamePrefix is defined for the MemorySection in the Basic Software Module Description or Software Component Description.
  - <snp> shall be the symbol of the Section NamePrefix associated to the MemorySection if a SectionNamePrefix is defined for the MemorySection.
  - <vi> is the `vendorId` of the BSW module
  - <ai> is the `vendorApiInfix` of the BSW module

  The sub part in squared brackets [_<vi>_<ai>] is omitted if no `vendorApiInfix` is defined for the Basic Software Module which indicates that it does not use multiple instantiation.
- the `shortName` of the software component type for software components (case sensitive)

<INIT_POLICY> is the initialization policy of variables. Possible values are:

- NO_INIT: Used for variables that are never cleared and never initialized.
- CLEARED: Used for variables that are cleared to zero after every reset.
- POWER_ON_CLEARED: Used for variables that are cleared to zero only after power on reset.
- INIT: Used for variables that are initialized with values after every reset.
- POWER_ON_INIT: Used for variables that are initialized with values only after power on reset.

Memory type	***Syntax of memory class (memclass) and pointer class (ptrclass) macro parameter***	***Comments***	***Located in***
Code	`<PREFIX>_CODE[_<PERIOD>]`	To be used for code. PERIOD is the typical period time value and unit of the ExecutableEntitys in this MemorySection. The name part [_<PERIOD>] is optional. units are: US microseconds MS milli second S second For example: 100US, 400US, 1MS, 5MS, 10MS, 20MS, 100MS, 1S Please note that deviations from this typical period time are possible due to integration decisions (e.g. RTEEvent To Task Mapping). Further, in special modes of the ECU the code may be scheduled with a higher or lower period.	Compiler_Cfg.h

Memory type	*Syntax of memory class (memclass) and pointer class (ptrclass) macro parameter*	*Comments*	*Located in*
Code	`<PREFIX>_<CN>_CODE`	To be used for callout code. <CN> is the callback name (including module reference) written in uppercase letters.	
Code	`<PREFIX>_CODE_FAST`	To be used for code that shall go into fast code memory segments. The FAST sections should be used when the execution does not happen in a well-defined period time but with the knowledge of high frequent access and /or high execution time, for example, a callback for a frequent notification.	
Code	`<PREFIX>_CODE_SLOW`	To be used for code that shall go into slow code memory segments. The SLOW sections should be used when the execution does not happen in a well-defined period time but with the knowledge of low frequent access, for example, a callback in case of seldom error.	
Constants	`<PREFIX>_CONST`	To be used for global or static constants.	
Constants	`<PREFIX>_CALIB`	To be used for calibration constants.	
Constants	`<PREFIX>_CONFIG_DATA`	To be used for module configuration constants.	
Constants	`<PREFIX>_CONST_SAVED_RECOVERY_ZONE<X>`	To be used for ROM buffers of variables saved in non-volatile memory.	
Pointer	`<PREFIX>_APPL_DATA`	To be used for references on application data (expected to be in RAM or ROM) passed via API	
Pointer	`<PREFIX>_APPL_CONST`	To be used for references on application constants (expected to be certainly in ROM, for instance pointer of Init-function) passed via API	
Pointer	`REGSPACE`	To be used for pointers to registers (e.g. `static volatile CONSTP2VAR(uint16, PWM_CONST, REGSPACE)`).	
Pointer	`<PREFIX>_APPL_CODE`	To be used for references on application functions. (e.g. call back function pointers). This section is **DEPRECATED** and shall not be used in fotore development. This memory class identifier has been replaced by <PREFIX>_<CN>_CODE.	
Variables	`<PREFIX>_VAR_<INIT_POLICY>`	To be used for all global or static variables.	

Memory type	*Syntax of memory class (memclass) and pointer class (ptrclass) macro parameter*	*Comments*	*Located in*
Variables	<PREFIX>_VAR_FAST_<INIT_POLICY>	To be used for all global or static variables that have at least one of the following properties: • accessed bitwise • frequently used • high number of accesses in source code Some platforms allow the use of bit instructions for variables located in this specific RAM area as well as shorter addressing instructions. This saves code and runtime.	
Variables	<PREFIX>_VAR_SLOW_<INIT_POLICY>	To be used for all infrequently accessed global or static variables.	
Variables	<PREFIX>_INTERNAL_VAR_<INIT_POLICY>	To be used for global or static variables which are accessible from a calibration tool.	
Variables	<PREFIX>_VAR_SAVED_ZONE<X>	To be used for RAM buffers of variables saved oin non-volatile memory.	
Variables	<PREFIX>_CALLOUT_CODE	To be used for references on application functions. (e.g. callout function pointers) This section is DEPRECATED and shall not be used in fotore development. This memory class identifier has been replaced by <PREFIX>_<CN>_CODE.	
Variables	<PREFIX>_VAR_NOINIT	To be used for all global or static variables that are never initialized. This section is **DEPRECATED** and shall not be used in fotore development. This memory class identifier has been replaced by <PREFIX>_VAR_<INIT_POLICY>.	
Variables	<PREFIX>_VAR_POWER_ON_INIT	To be used for all global or static variables that are initialized only after power on reset This section is **DEPRECATED** and shall not be used in fotore development. This memory class identifier has been replaced by <PREFIX>_VAR_<INIT_POLICY>.	

Memory type	*Syntax of memory class (memclass) and pointer class (ptrclass) macro parameter*	*Comments*	*Located in*
Variables	<PREFIX>_VAR_FAST	To be used for all global or static variables that have at least one of the following properties: • accessed bitwise • frequently used • high number of accesses in source code • • This section is **DEPRECATED** and shall not be used in fotore development. This memory class identifier has been replaced by <PREFIX>_VAR_FAST_<INIT_POLICY>.	
Variables	<PREFIX>_VAR	To be used for global or static variables that are initialized after every reset. This section is **DEPRECATED** and shall not be used in fotore development. This memory class identifier has been replaced by <PREFIX>_VAR_<INIT_POLICY>.	
Variables	AUTOMATIC	To be used for local non static variables	Compiler.h
Type Definitions	TYPEDEF	To be used in type definitions, where no memory qualifier can be specified.	Compiler.h

For the memory classes that have the form <PREFIX>_<NAME>, one can specify the part <NAME> in the the MemorySections of a Basic Software Module Description or Software Component Description as follows. This is especially required for generated code:

- <NAME> is the shortName (case sensitive) of the SwAddrMethod referred from the MemorySection if if the MemorySection has no memClassSymbol attribute defined.
- Only for Basic Software: <NAME> is the memClassSymbol (case sensitive) of the MemorySection if this attribute is defined.

」()

[SWS_COMPILER_00041]「 Each AUTOSAR software module and application software component shall wrap declaration and definition of code, variables, constants and pointer types using the following keyword macros: 」()

For instance:
native C-API:

```
Std_ReturnType Spi_SetupBuffers
(
    Spi_ChannelType       Channel,
    const Spi_DataType   *SrcDataBufferPtr,
    Spi_DataType         *DesDataBufferPtr,
    Spi_NumberOfDataType  Length
);
```

is encapsulated:

```
FUNC(Std_ReturnType, SPI_CODE) Spi_SetupBuffers
(
    Spi_ChannelType        Channel,
    P2CONST(Spi_DataType, AUTOMATIC, SPI_APPL_DATA) SrcDataBufferPtr,
    P2VAR(Spi_DataType, AUTOMATIC, SPI_APPL_DATA,)  DesDataBufferPtr,
    Spi_NumberOfDataType   Length
);
```

7.1.3 Contents of Compiler.h

[SWS_COMPILER_00004]「 The file name of the compiler abstraction shall be 'Compiler.h'. 」(SRS_BSW_00348, SRS_BSW_00361)

[SWS_COMPILER_00053]「 The file Compiler.h shall contain the definitions and macros specified in chapter 7.1.5. Those are fix for one specific compiler and platform. 」()

[SWS_COMPILER_00005]「 If a compiler does not require or support the usage of special keywords; the corresponding macros specified by this specification shall be provided as empty definitions or definitions without effect.
Example:

```
#define FUNC(type, memclass) type
/* not required for DIABDATA */
```

」()

[SWS_COMPILER_00010]「 The compiler abstraction shall define a symbol for the target compiler according to the following naming convention:
_<COMPILERNAME>_C_<PLATFORMNAME>_

Note 1: In order to avoid incompatibilities and/ or inconsistencies, the compiler symbol definitions are not allowed to contain any value behind the symbol.

Note 2: These defines can be used to switch between different implementations for different compilers, e.g.

- inline assembler fragments in drivers
- special pragmas for memory alignment control
- localization of function calls
- adaptions to memory models 」(SRS_BSW_00306, SRS_BSW_00006)

List of symbols: see SWS_COMPILER_00012

[SWS_COMPILER_00030] 「 "Compiler.h" shall provide information of the supported compiler vendor and the applicable compiler version. 」()

[SWS_COMPILER_00035] 「 The macro parameters memclass and ptrclass shall not be filled with the compiler specific keywords but with one of the configured values in SWS_COMPILER_00040. 」(SRS_BSW_00306, SRS_BSW_00006)

The rationale is that the module's implementation shall not be affected when changing a variable's, a pointer's or a function's storage class.

[SWS_COMPILER_00036] 「 C forbids the use of the far/near-keywords on function local variables (auto-variables). For this reason when using the macros below to allocate a pointer on stack, the memclass-parameter shall be set to AUTOMATIC. 」(SRS_BSW_00306, SRS_BSW_00006)

[SWS_COMPILER_00047] 「 The Compiler.h header file shall protect itself against multiple inclusions.
For instance:

```
#ifndef COMPILER_H
   #define COMPILER_H
    /* implementation of Compiler.h */
    ...
   #endif /* COMPILER_H */
```

There may be only comments outside of the ifndef - endif bracket. 」()

[SWS_COMPILER_00050] 「 It is allowed to extend the Compiler Abstraction header with vendor specific extensions. Vendor specific extended elements shall contain the AUTOSAR Vendor ID in the name. 」()

7.1.4 Contents of Compiler_Cfg.h

[SWS_COMPILER_00055] 「 The file Compiler_Cfg.h shall contain the module/component specific parameters (`ptrclass` and `memclass`) that are passed to the macros defined in Compiler.h. See SWS_COMPILER_00040 for memory types and required syntax. 」()

[SWS_COMPILER_00054] 「 Module specific extended elements shall contain the module abbreviation of the BSW module in the name. Application software component specific extended elements shall contain the Software Component Type's name. 」()

7.1.5 Comprehensive example

This example shows for a single API function where which macro is defined, used and configured.
Module: Eep
API function: Eep_Read
Platform: S12X
Compiler: Cosmic

File Eep.c:

```
#include "Std_Types.h"  /* This includes also Compiler.h */

FUNC(Std_ReturnType, EEP_CODE) Eep_Read
              (
                  Eep_AddressType EepromAddress,
                  P2VAR(uint8, AUTOMATIC, EEP_APPL_DATA) DataBufferPtr,
                  Eep_LengthType Length
              )
```

File Compiler.h:

```
#include "Compiler_Cfg.h"

#define AUTOMATIC
#define FUNC(rettype, memclass) rettype memclass
#define P2VAR(ptrtype, memclass, ptrclass)  ptrclass ptrtype * memclass
```

File Compiler_Cfg.h:

```
#define EEP_CODE
#define EEP_APPL_DATA @far  /* RAM blocks of NvM are in banked RAM */
```

What are the dependencies?
`EEP_APPL_DATA` is defined as ‘far’. This means that the pointers to the RAM blocks managed by the NVRAM Manager have to be defined as ‘far’ also. The application can locate RAM mirrors in banked RAM but also in non-banked RAM. The mapping of the RAM blocks to banked RAM is done in <Mip>_MemMap.h (see [12] for more information on <Mip>).

Because the pointers are also passed via Memory Interface and EEPROM Abstraction, their pointer and memory classes must also fit to `EEP_APPL_DATA`.

What would be different on a 32-bit platform?
Despite the fact that only the S12X has an internal EEPROM, the only thing that would change in terms of compiler abstraction are the definitions in Compiler_Cfg.h. They would change to empty defines:

```
#define EEP_CODE
#define EEP_APPL_DATA
```

7.1.6 Proposed process

To allow development and integration within a multi supplier environment a certain delivery process is indispensable. The following description can be seen as proposal:

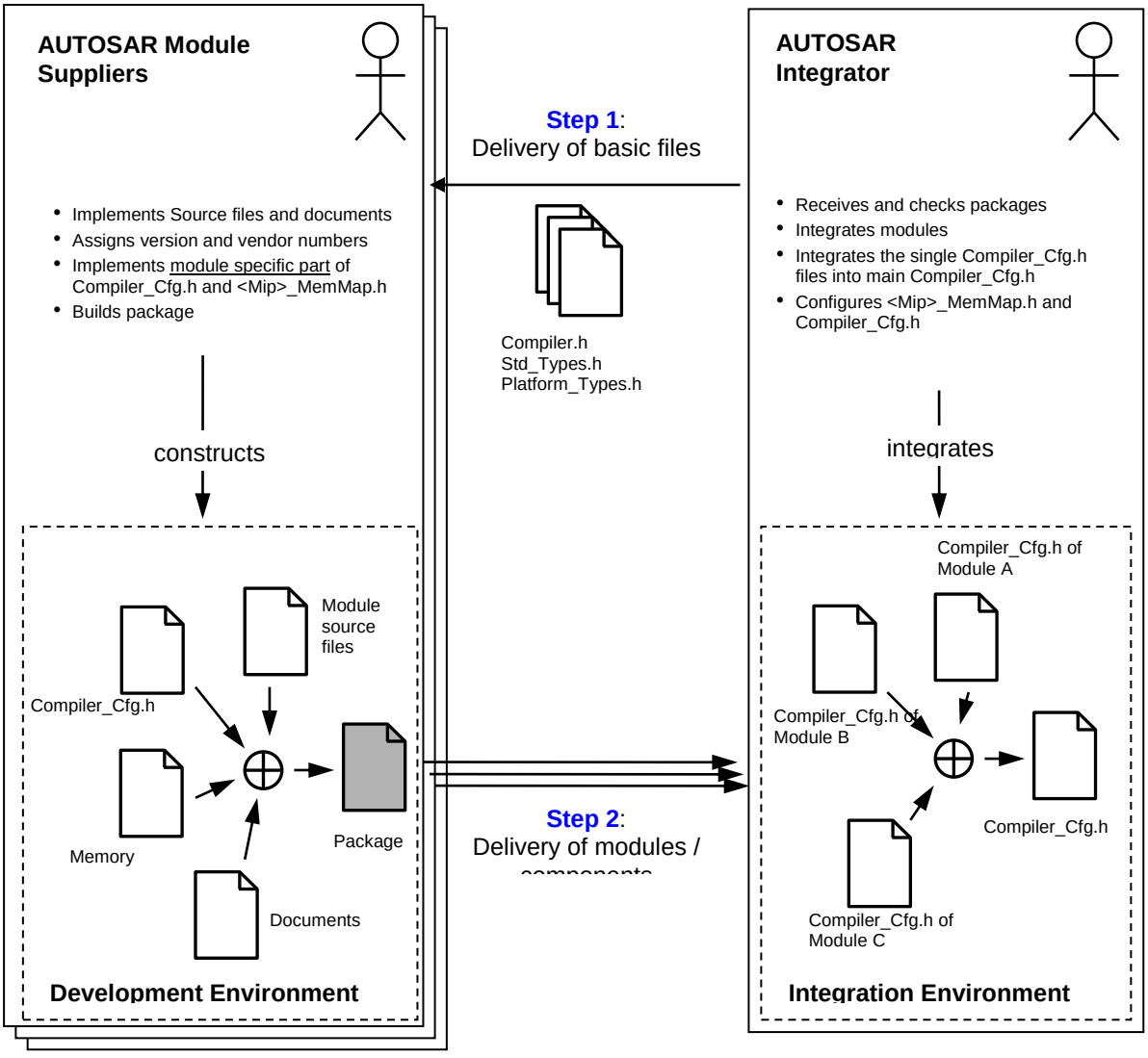

Figure 2: Proposal of integration-process

7.2 Development Errors

Not applicable.

7.3 Production Errors

Not applicable.

7.4 Extended Production Errors

Not applicable.

7.5 Error detection

Not applicable.

7.6 Error notification

Not applicable.

7.7 Version check

Not applicable.

7.8 Support for Debugging

Not applicable.

8 API specification

8.1 Imported types

Not applicable.

8.2 Macro definitions

These kind of items are the only API applicable to this module.

8.2.1 General definitions

8.2.1.1 Memory class AUTOMATIC

[SWS_COMPILER_00046]⌈

Define:	AUTOMATIC
Range:	"empty" --
Description:	The memory class AUTOMATIC shall be provided as empty definition, used for the declaration of local pointers.
Caveats:	SWS_COMPILER_00040

⌋()

8.2.1.2 Memory class TYPEDEF

[SWS_COMPILER_00059]⌈

Define:	TYPEDEF
Range:	"empty" --
Description:	The memory class TYPEDEF shall be provided as empty definition. This memory class shall be used within type definitions, where no memory qualifier can be specified. This can be necessary for defining pointer types, with e.g. P2VAR, where the macros require two parameters. First parameter can be specified in the type definition (distance to the memory location referenced by the pointer), but the second one (memory allocation of the pointer itself) cannot be defined at this time. Hence, memory class TYPEDEF shall be applied.
Caveats:	SWS_COMPILER_00040

⌋()

8.2.1.3 NULL_PTR

[SWS_COMPILER_00051]⌈

Define:	NULL_PTR
Range:	void pointer `((void *)0)`
Description:	The compiler abstraction shall provide the NULL_PTR define with a void pointer to zero definition.
Caveats:	SWS_COMPILER_00040

⌋()

8.2.1.4 INLINE

[SWS_COMPILER_00057]⌈

Define:	INLINE
Range:	inline/"empty" --
Description:	The compiler abstraction shall provide the INLINE define for abstraction of the keyword inline.
Caveats:	SWS_COMPILER_00040

⌋()

8.2.1.5 LOCAL_INLINE

[SWS_COMPILER_00060]⌈

Define:	LOCAL_INLINE
Range:	static inline/"empty" --
Description:	The compiler abstraction shall provide the LOCAL_INLINE define for abstraction of the keyword inline in functions with "static" scope.
Caveats:	Different compilers may require a different sequence of the keywords "static" and "inline" if this is supported at all.

⌋()

8.2.2 Function definitions

The following tables do not contain requirements. They just give an overview of used function keywords and their syntax within different compilers. This analysis is required for a correct and complete specification of methods and keywords and as rationale for those people who doubt the necessity of a compiler abstraction in AUTOSAR. These tables are not the complete overview of all existing compilers and platforms and their usage in AUTOSAR. However, the tables show examples that cover most use cases, from which the concepts are derived.

On platforms with memory exceeding the addressable range of the architecture (e.g. S12X with 512k of Flash) the compiler needs to know if a called function is reachable within normal addressing commands ('near') or extended/paged addressing commands ('far').

Compiler analysis for near functions:

Compiler	***Required syntax***
Cosmic, S12X	`@near void MyNearFunction(void);` Call of a near function results in a local page call or to a call into direct page. Dependent of compiler settings the compiler controls only the calling convention or allocation and calling convention.
Metrowerks, S12X	`void __near MyNearFunction(void);` Call of a near function results in a local page call or to a call into direct page.
IAR, HCS12 C/C++	`void __non_banked MyNearFunction (void);`

Tasking, ST10	`void _near MyNearFunction (void);`
	`_near void MyNearFunction (void);`
	Call of a near function results in a local segment code access (relevant in large model).

Tasking, TC1796	`void MyNearFunction (void);` (No keywords required)
Greenhills, V850	`void MyNearFunction (void);` (No keywords required)
ADS, ST30	`void MyNearFunction (void);` (No keywords required)
DIABDATA, MPC5554	`void MyNearFunction (void);` (No keywords required)

Compiler analysis for far functions:

Compiler	***Required syntax***
Cosmic, S12X	`@far void MyFarFunction(void);` Dependent of compiler settings the compiler controls only the calling convention or allocation and calling convention.
Metrowerks, S12X	`void __far MyFarFunction(void);`
IAR, HCS12 C/C++	`void __banked MyFarFunction (void);`
Tasking, ST10	`void _huge MyFarFunction (void);` `_huge void MyFarFunction (void);`
Tasking, TC1796	`void MyFarFunction (void);` (No keywords required)
Greenhills, V850	`void MyFarFunction (void);` (No keywords required)
ADS, ST30	`void MyFarFunction (void);` (No keywords required)
DIABDATA, MPC5554	`void MyFarFunction (void);` (No keywords required)

8.2.2.1 FUNC

[SWS_COMPILER_00001]⌈

Macro name:	FUNC	
Syntax:	`#define FUNC(rettype, memclass)`	
Parameters (in):	`retype`	return type of the function
	`memclass`	classification of the function itself
Parameters (out):	`None`	--
Return value:	`None`	--
Description:	The compiler abstraction shall define the FUNC macro for the declaration and definition of functions that ensures correct syntax of function declarations as required by a specific compiler.	
Caveats:	--	
Configuration:	--	

⌋ ()

Example (Cosmic, S12X):
```
#define <PREFIX>_CODE          @near
#define FUNC(rettype, memclass) memclass rettype
```

Required usage for function declaration and definition:
```
FUNC(void, <PREFIX>_CODE) ExampleFunction (void);
```

[SWS_COMPILER_00058]⌈ In the parameter list of this macro no further Compiler Abstraction macros shall be nested. Instead, use a previously defined type as return type or use FUNC_P2CONST/FUNC_P2VAR.

⌋ ()

Example:
```
 typedef P2VAR(uint8, AUTOMATIC, <PREFIX>_VAR) NearDataType;
 FUNC(NearDataType, <PREFIX>_CODE)
FarFuncReturnsNearPtr(void);
```

8.2.2.2 FUNC_P2CONST

[SWS_COMPILER_00061]Γ

Macro name:	FUNC_P2CONST	
Syntax:	`#define FUNC_P2CONST(rettype, ptrclass, memclass)`	
Parameters (in):	`rettype`	return type of the function
	`ptrclass`	defines the classification of the pointer's distance
	`memclass`	classification of the function itself
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	The compiler abstraction shall define the FUNC_P2CONST macro for the declaration and definition of functions returning a pointer to a constant. This shall ensure the correct syntax of function declarations as required by a specific compiler.	
Caveats:	--	
Configuration:	--	

」()

Example (Cosmic, S12X):

```
#define <PREFIX>_PBCFG        @far
#define <PREFIX>_CODE         @near
#define FUNC_P2CONST(rettype, ptrclass, memclass)\
const ptrclass rettype * memclass
```

Required usage for function declaration and definition:

```
FUNC_P2CONST(uint16, <PREFIX>_PBCFG, <PREFIX>_CODE)
ExampleFunction (void);
```

[SWS_COMPILER_00062]Γ In the parameter list of the FUNC_P2CONST, no further Compiler Abstraction macros shall be nested.

」()

8.2.2.3 FUNC_P2VAR

[SWS_COMPILER_00063]⌈

Macro name:	FUNC_P2VAR	
Syntax:	`#define FUNC_P2VAR(rettype, ptrclass, memclass)`	
Parameters (in):	`rettype`	return type of the function
	`ptrclass`	defines the classification of the pointer's distance
	`memclass`	classification of the function itself
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	The compiler abstraction shall define the FUNC_P2VAR macro for the declaration and definition of functions returning a pointer to a variable. This shall ensure the correct syntax of function declarations as required by a specific compiler.	
Caveats:	--	
Configuration:	--	

⌋ ()

Example (Cosmic, S12X):

```
#define <PREFIX>_PBCFG        @far
#define <PREFIX>_CODE         @near
#define FUNC_P2VAR(rettype, ptrclass, memclass)\
ptrclass rettype * memclass
```

Required usage for function declaration and definition:

```
FUNC_P2VAR(uint16, <PREFIX>_PBCFG, <PREFIX>_CODE)
ExampleFunction (void);
```

[SWS_COMPILER_00064]⌈ In the parameter list of the macro FUNC_P2VAR, no further Compiler Abstraction macros shall be nested.

⌋ ()

8.2.3 Pointer definitions

The following tables do not contain requirements. They just give an overview of used pointer keywords and their syntax within different compilers. This analysis is required for a correct and complete specification of methods and keywords and as rationale for those people who doubt the necessity of a compiler abstraction in AUTOSAR. These tables are not the complete overview of all existing compilers and platforms and their usage in AUTOSAR. However, the tables show examples that cover most use cases, from which the concepts are derived.

On platforms with memory exceeding the addressable range of the architecture (e.g. S12X with 512k of Flash) the compiler needs to know if data referenced by a pointer is accessible by normal addressing commands ('near') or extended/paged addressing commands ('far').

Compiler analysis for near pointers pointing to variable_data in RAM (use case: pointer to data buffer where data has to be copied to):

Compiler	***Required syntax***
Cosmic, S12X	`@near uint8* MyNearPointer;`
Metrowerks, S12X	`uint8* __near MyNearPointer;`
IAR, HCS12 C/C++	`uint8* __data16 MyNearPointer;`
Tasking, ST10	`_near uint8* MyNearPointer;`
Tasking, TC1796	`uint8* MyNearPointer;` (No keywords required)
Greenhills, V850	`uint8* MyNearPointer` (No keywords required)
ADS, ST30	`uint8* MyNearPointer` (No keywords required)
DIABDATA, MPC5554	`uint8* MyNearPointer` (No keywords required)

Compiler analysis for far pointers pointing to variable data in RAM:

Compiler	***Required syntax***
Cosmic, S12X	`@far uint8* MyFarPointer;`
Metrowerks, S12X	`uint8* __far MyFarPointer;`
IAR, HCS12 C/C++	(Information not available yet)

Tasking, ST10	`_far uint8* MyFarPointer;  /*14 bit arithmetic*/`
	`_huge uint8* MyFarPointer;  /*24 bit arithmetic*/`
	`_shuge uint8* MyFarPointer; /*16 bit arithmetic*/`
	`/* My personal note: CRAZY */`

Tasking, TC1796	`uint8* MyFarPointer;` (No keywords required)
Greenhills, V850	`uint8* MyFarPointer` (No keywords required)
ADS, ST30	`uint8* MyFarPointer` (No keywords required)
DIABDATA, MPC5554	`uint8* MyFarPointer` (No keywords required)

Compiler analysis for near pointers pointing to constant data in RAM (use case pointer to data buffer where data has to be read from):

Compiler	***Required syntax***
Cosmic, S12X	`@near uint8* MyNearPointer;` (Results in access of direct memory area)
Metrowerks, S12X	`const uint8* __near MyNearPointer;` (Results in access of direct memory area)
IAR, HCS12 C/C++	`const uint8*  MyNearPointer;` (Results in access of direct memory area)
Tasking, ST10	`const _near uint8* MyNearPointer;`
Tasking, TC1796	`const _near uint8* MyNearPointer;`
Greenhills, V850	`const uint8* MyNearPointer` (No additional keywords required)
ADS, ST30	`const uint8* MyNearPointer` (No additional keywords required)
DIABDATA, MPC5554	`const uint8* MyNearPointer` (No additional keywords required)

Compiler analysis for far pointers pointing to constant data in RAM:

Compiler	***Required syntax***
Cosmic, S12X	`@far uint8* MyFarPointer;`
Metrowerks, S12X	`const uint8* __far MyFarPointer;`
IAR, HCS12 C/C++	(Information not available yet)
Tasking, ST10	`const _far uint8* MyFarPointer;`
Tasking, TC1796	`uint8* MyFarPointer;` (No keywords required)
Greenhills, V850	`const uint8* MyFarPointer` (No additional keywords required)
ADS, ST30	`const uint8* MyFarPointer` (No additional keywords required)
DIABDATA, MPC5554	`const uint8* MyFarPointer` (No additional keywords required)

Compiler analysis for near pointers pointing to data in ROM (use case pointer to display data in ROM passed to SPI Driver):

Compiler	***Required syntax***
Cosmic, S12X	`const uint8* MyNearPointer;` `(Without near keyword because this is by default near!)`
Metrowerks, S12X	`const uint8* __near MyNearPointer;`
IAR, HCS12 C/C++	`const uint8* MyNearPointer;` `(Without near keyword because this is by default near!)`
Tasking, ST10	`const _near uint8* MyNearPointer;`
Tasking, TC1796	`const uint8* MyNearPointer;` (No keywords required)
Greenhills, V850	`const uint8* MyNearPointer` (No additional keywords required)
ADS, ST30	`const uint8* MyNearPointer` (No additional keywords required)
DIABDATA, MPC5554	`const uint8* MyNearPointer` (No additional keywords required)

Compiler analysis for far pointers pointing to constant data in ROM:

Compiler	***Required syntax***
Cosmic, S12X	not possible
Metrowerks, S12X	`const uint8* __far MyFarPointer;`
IAR, HCS12 C/C++	Access function and the banked constant data are located in the same bank: `const uint8* MyFarPointer;` but caller shall use the `__address_24_of` macro Access function is located in non-banked memory: PPAGE register has to be handled manually Access function and the banked constant data are located in different banks: Not possible
Tasking, ST10	`const _far uint8* MyFarPointer;`
Tasking, TC1796	`const uint8* MyFarPointer;` (No keywords required)
Greenhills, V850	`const uint8* MyFarPointer` (No additional keywords required)
ADS, ST30	`const uint8* MyFarPointer` (No additional keywords required)
DIABDATA, MPC5554	`const uint8* MyFarPointer` (No additional keywords required)

The HW architecture of the S12X supports different paging mechanisms with different limitations e.g. supported instruction set or pointer distance. Therefore the IAR, HCS12 C/C++ and the Cosmic, S12X compilers are limited in the usage of generic pointers applicable for the whole memory area because of the expected code overhead.

Conclusion: These vendors should adapt their compilers, because a generic SW architecture as described by AUTOSAR cannot be adjusted in every case to the platform specific optimal solution.

Compiler analysis for pointers, where the symbol of the pointer itself is placed in near-memory:

Compiler	***Required syntax***
Cosmic, S12X	`uint8* @near MyPointerInNear;`
Metrowerks, S12X	`__near uint8* MyPointerInNear;`
Tasking, ST10	`uint8* _near MyPointerInNear;`
Tasking, TC1796	`uint8* MyPointerInNear;` (No keywords required)
Greenhills, V850	`uint8* MyPointerInNear` (No keywords required)
ADS, ST30	`uint8* MyPointerInNear` (No keywords required)
DIABDATA, MPC5554	`uint8* MyPointerInNear` (No keywords required)

Compiler analysis for pointers, where the symbol of the pointer itself is placed in far-memory:

Compiler	***Required syntax***
Cosmic, S12X	`uint8* @far MyPointerInFar;`
Metrowerks, S12X	`__far uint8* MyPointerInFar;`
Tasking, ST10	`uint8* _far MyPointerInFar;`
Tasking, TC1796	`uint8* MyPointerInFar;` (No keywords required)
Greenhills, V850	`uint8* MyPointerInFar` (No keywords required)
ADS, ST30	`uint8* MyPointerInFar` (No keywords required)
DIABDATA, MPC5554	`uint8* MyPointerInFar` (No keywords required)

The examples above lead to the conclusion, that for definition of a pointer it is not sufficient to specify only one memory class. Instead, a combination of two memory classes, one for the pointer's 'distance' and one for the pointer's symbol itself, is possible, e.g.:

```
/* Tasking ST10, far-pointer in near memory
 * (both content and pointer in RAM)
 */
_far uint8* _near MyFarPointerInNear;
```

Compiler analysis for function pointers:

Compiler	***Required syntax***
Cosmic, S12X	`@near void (* const Irq_InterruptVectorTable[])(void)` Call of a near function results in an interpage call or to a call into direct page:
Metrowerks, S12X	`void (*const __near Irq_InterruptVectorTable[]) (void)` Call of a near function results in an interpage call or to a call into direct page: Near functions and far functions are not compatible because of other ret-statements:
IAR, HCS12 C/C++	`__non_banked void (* const Irq_InterruptVectorTable[])(void)` Casting from __non_banked to __banked is performed through zero extension: Casting from __banked to __non_banked is an illegal operation.
Tasking, ST10	`_far void (*NvM_AsyncCbkPtrType) (NvM_ModuleIdType ModuleId, NvM_ServiceIdType ServiceId )` Call of a near function results in a local segment code access (relevant in large model):
Tasking, TC1796	`void (*NvM_AsyncCbkPtrType) (NvM_ModuleIdType ModuleId, NvM_ServiceIdType ServiceId )` (No additional keywords required)
Greenhills, V850	`void (*NvM_AsyncCbkPtrType) (NvM_ModuleIdType ModuleId,`

Compiler	Required syntax
	NvM_ServiceIdType ServiceId) (No additional keywords required)
ADS, ST30	void (*NvM_AsyncCbkPtrType) (NvM_ModuleIdType ModuleId, NvM_ServiceIdType ServiceId) (No additional keywords required)
DIABDATA, MPC5554	void (*NvM_AsyncCbkPtrType) (NvM_ModuleIdType ModuleId, NvM_ServiceIdType ServiceId) (No additional keywords required)

8.2.3.1 P2VAR

[SWS_COMPILER_00006]Γ

Macro name:	P2VAR	
Syntax:	`#define P2VAR(ptrtype, memclass, ptrclass)`	
Parameters (in):	`ptrtype`	type of the referenced variable
	`memclass`	classification of the pointer's variable itself
	`ptrclass`	defines the classification of the pointer's distance
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	The compiler abstraction shall define the P2VAR macro for the declaration and definition of pointers in RAM, pointing to variables. The pointer itself is modifiable (e.g. ExamplePtr++). The pointer's target is modifiable (e.g. *ExamplePtr = 5).	
Caveats:	--	
Configuration:	--	

」()

Example (Metrowerks, S12X):

```
#define P2VAR(ptrtype, memclass, ptrclass) \
                ptrclass ptrtype * memclass
```

Required usage for pointer declaration and definition:

```
#define SPI_APPL_DATA @far
#define SPI_VAR_FAST  @near

P2VAR(uint8, SPI_VAR_FAST, SPI_APPL_DATA) Spi_FastPointerToApplData;
```

8.2.3.2 P2CONST

[SWS_COMPILER_00013]⌈

Macro name:	P2CONST	
Syntax:	`#define P2CONST(ptrtype, memclass, ptrclass)`	
Parameters (in):	`ptrtype`	type of the referenced constant
	`memclass`	classification of the pointer's variable itself
	`ptrclass`	defines the classification of the pointer's distance
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	The compiler abstraction shall define the P2CONST macro for the declaration and definition of pointers in RAM pointing to constants The pointer itself is modifiable (e.g. ExamplePtr++). The pointer's target is not modifiable (read only).	
Caveats:	--	
Configuration:	--	

⌋()

Example (Metrowerks, S12X):
```
#define P2CONST(ptrtype, memclass, ptrclass) \
                const ptrtype memclass * ptrclass
```

Example (Cosmic, S12X):
```
#define P2CONST(ptrtype, memclass, ptrclass) \
                const ptrtype ptrclass * memclass
```

Example (Tasking, ST10):
```
#define P2CONST(ptrtype, memclass, ptrclass) \
                const ptrclass ptrtype * memclass
```

Required usage for pointer declaration and definition:
```
#define EEP_APPL_CONST @far
#define EEP_VAR  @near

P2CONST(Eep_ConfigType, EEP_VAR, EEP_APPL_CONST) Eep_ConfigurationPtr;
```

8.2.3.3 CONSTP2VAR

[SWS_COMPILER_00031]⌈

Macro name:	CONSTP2VAR	
Syntax:	`#define CONSTP2VAR (ptrtype, memclass, ptrclass)`	
Parameters (in):	`ptrtype`	type of the referenced variable
	`memclass`	classification of the pointer's constant itself
	`ptrclass`	defines the classification of the pointer's distance
Parameters (out):	`None`	--
Return value:	`None`	--
Description:	The compiler abstraction shall define the CONSTP2VAR macro for the declaration and definition of constant pointers accessing variables. The pointer itself is not modifiable (fix address). The pointer's target is modifiable (e.g. *ExamplePtr = 18).	
Caveats:	--	
Configuration:	--	

⌋()

Example (Tasking, ST10):

```
#define CONSTP2VAR (ptrtype, memclass, ptrclass) \
                ptrclass ptrtype * const memclass
```

Required usage for pointer declaration and definition:

```
/* constant pointer to application data */
CONSTP2VAR (uint8, NVM_VAR, NVM_APPL_DATA)
NvM_PointerToRamMirror = Appl_RamMirror;
```

8.2.3.4 CONSTP2CONST

[SWS_COMPILER_00032]⌈

Macro name:	CONSTP2CONST	
Syntax:	`#define CONSTP2CONST(ptrtype, memclass, ptrclass)`	
Parameters (in):	`ptrtype`	type of the referenced constant
	`memclass`	classification of the pointer's constant itself
	`ptrclass`	defines the classification of the pointer's distance
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	The compiler abstraction shall define the CONSTP2CONST macro for the declaration and definition of constant pointers accessing constants. The pointer itself is not modifiable (fix address). The pointer's target is not modifiable (read only).	
Caveats:	--	
Configuration:	--	

⌋()

Example (Tasking, ST10):
```
#define  CONSTP2CONST (ptrtype, memclass, ptrclass) \
                const memclass ptrtype * const ptrclass
```

Required usage for pointer declaration and definition:
```
#define CAN_PBCFG_CONST @gpage
#define CAN_CONST       @near

/* constant pointer to the constant postbuild configuration
data */
CONSTP2CONST (Can_PBCfgType, CAN_CONST, CAN_PBCFG_CONST)
Can_PostbuildCfgData = CanPBCfgDataSet;
```

8.2.3.5 P2FUNC

[SWS_COMPILER_00039]Γ

Macro name:	P2FUNC	
Syntax:	`#define P2FUNC(rettype, ptrclass, fctname)`	
Parameters (in):	`rettype`	return type of the function
	`ptrclass`	defines the classification of the pointer's distance
	`fctname`	function name respectively name of the defined type
Parameters (out):	`None`	--
Return value:	`None`	--
Description:	The compiler abstraction shall define the P2FUNC macro for the type definition of pointers to functions.	
Caveats:	--	
Configuration:	--	

」()

Example (Metrowerks, S12X):
```
define P2FUNC(rettype, ptrclass, fctname)\
                rettype (*ptrclass fctname)
```

Example (Cosmic, S12X):
```
#define P2FUNC(rettype, ptrclass, fctname) \
                ptrclass rettype (*fctname)
```

Required usage for pointer type declaration:
```
#define EEP_APPL_CONST @far
#define EEP_VAR  @near

typedef P2FUNC (void, NVM_APPL_CODE, NvM_CbkFncPtrType)
(void);
```

8.2.3.6 CONSTP2FUNC

[SWS_COMPILER_00065]⌈

Macro name:	CONSTP2FUNC	
Syntax:	`#define CONSTP2FUNC(rettype, ptrclass, fctname)`	
Parameters (in):	`rettype`	return type of the function
	`ptrclass`	defines the classification of the pointer's distance
	`fctname`	function name respectively name of the defined type
Parameters (out):	`None`	--
Return value:	`None`	--
Description:	The compiler abstraction shall define the CONSTP2FUNC macro for the type definition of constant pointers to functions.	
Caveats:	--	
Configuration:	--	

⌋ ()

Example (PowerPC):
```
#define CONSTP2FUNC(rettype, ptrclass, fctname)\
                    rettype (* const fctname)
```

Example (CodeWarrior, S12X):
```
#define CONSTP2FUNC(rettype, ptrclass, fctname)\
                    rettype (* const ptrclass fctname)
```

8.2.4 Constant definitions

8.2.4.1 CONST

[SWS_COMPILER_00023]⌈

Macro name:	CONST	
Syntax:	`#define CONST(consttype, memclass)`	
Parameters (in):	`consttype`	type of the constant
	`memclass`	classification of the constant itself
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	The compiler abstraction shall define the CONST macro for the declaration and definition of constants.	
Caveats:	--	
Configuration:	--	

⌋ ()

Example (Cosmic, S12X):
```
#define CONST(type, memclass) memclass const type
```

Required usage for declaration and definition:

```
#define NVM_CONST @gpage

CONST(uint8, NVM_CONST) NvM_ConfigurationData;
```

8.2.5 Variable definitions

8.2.5.1 VAR

[SWS_COMPILER_00026]

Macro name:	VAR	
Syntax:	`#define VAR(vartype, memclass)`	
Parameters (in):	`vartype`	type of the variable
	`memclass`	classification of the variable itself
Parameters (out):	`None`	--
Return value:	`None`	--
Description:	The compiler abstraction shall define the VAR macro for the declaration and definition of variables.	
Caveats:	--	
Configuration:	--	

()

Example (Tasking, ST10):

```
#define VAR(type, memclass) memclass type
```

Required usage for declaration and definition:

```
#define NVM_FAST_VAR _near

VAR(uint8, NVM_FAST_VAR) NvM_VeryFrequentlyUsedState;
```

8.3 Type definitions

Not applicable.

8.4 Function definitions

Not applicable.

8.5 Call-back notifications

Not applicable.

8.6 Scheduled functions

Not applicable.

8.7 Expected Interfaces

8.7.1 Mandatory Interfaces

Not applicable.

8.7.2 Optional Interfaces

Not applicable.

8.7.3 Configurable interfaces

Not applicable.

8.8 Service Interfaces

8.8.1 Scope of this Chapter

Not applicable.

8.8.2 Overview

Not applicable.

8.8.3 Specification of the Ports and Port Interfaces

8.8.3.1 General Approach

Not applicable.

8.8.3.2 Data Types

Not applicable.

8.8.3.3 Port Interface

Not applicable.

8.8.4 Definition of the Service

Not applicable.

8.8.5 Configuration of the DET

Not applicable.

9 Sequence diagrams

Not applicable.

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. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

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

Chapter 10.3 specifies published information of this module.

The Compiler Abstraction has no separate configuration interface by means of specifying a separate parameter definition. Instead, configuration of the Memory Mapping has been extended (see [13]) by the parameters described in this chapter.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:
Layered Software Architecture [3]
Specification Of ECU Configuration [4]

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

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 8 and Chapter 9.

10.2.1 Variants

Variant PC (**P**re **C**ompile): This is the only variant because all configuration parameters are pre-compile time parameters, which influence the compilation process.
Each of the different memory classes (memclass) and pointer classes (ptrclass) is represented by a define.

10.2.2 Module-Specific Memory Classes

[13] defines module-specific memory classes in the container ‘MemMapAddressingModeSet’. This container has been extended by the parameter ‘MemMapCompilerAddressingMode’.

[SWS_COMPILER_00066]「 The parameter ‘MemMapCompilerAddressingMode’ shall contain the implementation behind a module-specific memory class symbol.」()

10.2.3 Global Memory Classes

Furthermore, there are global memory classes that are valid for all modules. These can be configured in the container ‘MemMapGenericCompilerClass’.

[SWS_COMPILER_00067]「 Global memory classes (e.g. REGSPACE) shall be configured in the container ‘MemMapGenericCompilerClass’.」()

[SWS_COMPILER_00068]「 The parameter ‘MemMapGenericCompilerAddressingMode’ shall contain the implementation behind a global memory class symbol.」()

[SWS_COMPILER_00042]「 The file Compiler.h is specific for each build scenario. Therefore there is no standardized configuration interface specified.」()

10.3 Published Information

Not applicable.

11 Changes in Release 4.0

11.1 Deleted SWS Items

SWS Item	*Rationale*

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
-		

11.3 Changed SWS Items

SWS Item	*Rationale*

11.4 Added SWS Items

SWS Item	*Rationale*

12 Not applicable requirements

[SWS_COMPILER_00999]「 These requirements are not applicable to this specification. 」 (SRS_BSW_00300, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00305, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00310, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00323, SRS_BSW_00325, SRS_BSW_00327, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00344, SRS_BSW_00346, SRS_BSW_00350, SRS_BSW_00353, SRS_BSW_00357, SRS_BSW_00358, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00369, SRS_BSW_00371, SRS_BSW_00373, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00380, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00004, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00407, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00413, SRS_BSW_00414, SRS_BSW_00415, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00005, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00158, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00171, SRS_BSW_00172)