Document Title	Specification of Memory Mapping
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	128

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R21-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2021-11-25	R21-11	AUTOSAR Release Management	• POWER_ON_INIT behaviour does not match ComputerRuntimeInitialization • Deprecate compiler abstraction • Description regarding alignment is too strict for some targets
2020-11-30	R20-11	AUTOSAR Release Management	• No content changes
2019-11-28	R19-11	AUTOSAR Release Management	• Clarify NO-INIT policy • Clarify caseness of VendorApiInfix • Clarify usage of core scope • Update of referenced pictures • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Support splitting of modules in allocatable memory parts • Clarify handling of configuration data • Additional minor corrections / clarifications / editorial changes; For details please refer to the Change Documentation
2017-12-08	4.3.1	AUTOSAR Release Management	• Amend explanatory text • Editorial changes

2016-11-30	4.3.0	AUTOSAR Release Management	• Support dedicated allocation of pointer variables • Remove obsolete specification content • Amend examples • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Support core scope specific memory allocation • Clean up requirement tracing • editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Support partitioning of BSW for safety systems • Remove obsolete memory sections in Recommendation A • Clarifications about the handling of SIZE and ALIGNMENT • editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Clarify usage of <X> in recovery and saved data zone • editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Clarify usage of default section
2013-03-15	4.1.1	AUTOSAR Administration	• Consistent naming pattern for memory allocation keywords • pre-define M1 values for the option attribute of MemorySection and SwAddrMethod • added configuration for Compiler Abstraction • support BSW module specific MemMap header files • recommended memory allocation keywords are reworked
2011-12-22	4.0.3	AUTOSAR Administration	• Consistent naming pattern for memory allocation keywords is introduced • Refine definition the <PREFIX> part in memory allocation keywords

2009-12-18	4.0.1	AUTOSAR Administration	• ECU Configuration Parameters for MemMap defined • Define generation of MemMap header files • New standardised Memory Allocation Keywords for new initialisation policy CLEARED added • Refinement of <SIZE> suffix of Memory Allocation Keywords to <ALIGNMENT> suffix, • Clarify link MetaModel attribute values, – Define MemorySectionType and SectionInitializationPolicy for the standardised Memory Allocation Keywords – Define that <NAME> used for Memory Allocation Keywords is the MemorySection shortName • Application hint for usage of INLINE and LOCAL_INLINE added • Handling structs, arrays and unions redefined
2010-02-02	3.1.4	AUTOSAR Administration	• Typo errors are corrected throughout the document • Memory Mapping section has been extended for application SWC • Common Published information has been updated • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	Legal disclaimer revised

2006-11-28	2.1	AUTOSAR Administration	• In MEMMAP004,all size postfixes for memory segment names were listed, the keyword 'BOOLEAN was added, taking into account the particular cases where boolean data need to be mapped in a particular segment. • In MEMMAP004 and SWS_MemMap_00021,tables are defining the mapping segments associated to #pragmas instructions, adding some new segments to take into account some implementation cases • Document meta information extended • Small layout adaptations made
2006-05-16	2.0	AUTOSAR Administration	Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

This document specifies mechanisms for the mapping of code and data to specific memory sections via memory mapping files. For many ECUs and microcontroller platforms it is of utmost necessity to be able to map code, variables and constants module wise to specific memory sections. Selection of important use cases:

Avoidance of waste of RAM
If different variables (8, 16 and 32 bit) are used within different modules on a 32 bit platform, the linker will leave gaps in RAM when allocating the variables in the RAM. This is because the microcontroller platform requires a specific alignment of variables and some linkers do not allow an optimization of variable allocation.
This wastage of memory can be circumvented if the variables are mapped to specific memory sections depending on their size. This minimizes unused space in RAM.

Usage of specific RAM properties
Some variables (e.g. the RAM mirrors of the NVRAM Manager) must not be initialized after a power-on reset. It shall be possible to map them to a RAM section that is not initialized after a reset.
For some variables (e.g. variables that are accessed via bit masks) it improves both performance and code size if they are located within a RAM section that allows for bit manipulation instructions of the compiler. Those RAM sections are usually known as 'Near Page' or 'Zero Page'.

Usage of specific ROM properties
In large ECUs with external flash memory there is the requirement to map modules with functions that are called very often to the internal flash memory that allows for fast access and thus higher performance. Modules with functions that are called rarely or that have lower performance requirements are mapped to external flash memory that has slower access.

Usage of the same source code of a module for boot loader and application
If a module shall be used both in boot loader and application, it is necessary to allow the mapping of code and data to different memory sections.
A mechanism for mapping of code and data to memory sections that is supported by all compilers listed in chapter 3.1 is the usage of pragmas. As pragmas are very compiler specific, a mechanism that makes use of those pragmas in a standardized way has to be specified.

Support of Memory Protection
The usage of hardware memory protection requires a separation of the modules variables into different memory areas. Internal variables are mapped into protected memory, buffers for data exchange are mapped into unprotected memory.

Support of partitioning
In some cases it is necessary to separate partition assigned memory. Therefore an additional separation of the module variables into different memory (partition-)areas is needed if the BSW Module shall support a split over several Partitions.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Memory Mapping specification that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
BSW	Basic Software
ISR	Interrupt Service Routine
NVRAM	Non-Volatile RAM

Table 2.1: Abbreviations and Acronyms

3 Related documentation

3.1 Input documents

References

[1] Glossary
AUTOSAR_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[4] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[5] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[6] Methodology for Classic Platform
AUTOSAR_TR_Methodology

[7] Specification of RTE Software
AUTOSAR_SWS_RTE

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

[9] ARM ADS compiler manual

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

[11] TASKING for ST10 V8.5
C166/ST10 v8.5 C Cross-Compiler User's Manual, V5.16

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

3.2 Related standards and norms

Not applicable.

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for SWS Memory Mapping.

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.

A dedicated pack-control of structures is not supported. Hence global set-up passed via compiler / linker parameters has to be used.

A dedicated alignment control of code, variables and constants is not supported. Hence affected objects shall be assigned to different sections or a global setting passed via compiler / linker parameters has to be used.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

[SWS_MemMap_00020] ⌈The SWS Memory Mapping is applicable for each AUTOSAR basic software module and software component. Therefore the implementation of memory mapping files shall fulfill the implementation and configuration specific needs of each software module in a specific build scenario. See also [SWS_MemMap_00038], [SWS_MemMap_00003], [SWS_MemMap_00018] and [SWS_MemMap_00001].⌋*(SRS_BSW_00384, SRS_BSW_00351)*

5.1 File structure

5.1.1 Code file structure

Not applicable.

5.1.2 Header file structure

[SWS_MemMap_00028] ⌈The Memory Mapping shall provide a BSW memory mapping header file if any of the BSW Module Descriptions is describing a `DependencyOnArtifact` as `requiredArtifact.DependencyOnArtifact.category` = `MEMMAP` In this case the file name of the BSW memory mapping header file name is defined by the attribute value `requiredArtifact.DependencyOnArtifact.artifactDescriptor.shortLabel` in the BSW Module Description.⌋*(SRS_BSW_00465, SRS_BSW_00415, SRS_BSW_00351, SRS_BSW_00464)*

Please note that [SWS_MemMap_00028] does support that either several BSW Module Descriptions contributing to the same file (e.g MemMap.h for legacy code) or that the same BSW Module Description specifies a set of memory mapping header files with differnt names for example in case of a BSW Module Description of an ICC2 cluster.

For instance:

```
<REQUIRED-ARTIFACTS>
    <DEPENDENCY-ON-ARTIFACT>
        <SHORT-NAME>MemMap</SHORT-NAME>
        <CATEGORY>MEMMAP</CATEGORY>
        <ARTIFACT-DESCRIPTOR>
            <SHORT-LABEL>MemMap.h</SHORT-LABEL>
            <CATEGORY>SWHDR</CATEGORY>
        </ARTIFACT-DESCRIPTOR>
    </DEPENDENCY-ON-ARTIFACT>
</REQUIRED-ARTIFACTS>
```

Results in the generation of the requested Memory Allocation Key Words in the file MemMap.h

[SWS_MemMap_00032] ⌈For each basic software module description which is part of the input configuration a basic software module specific memory mapping header file `{Mip}_MemMap.h` shall be provided by the Memory Mapping if the BSW Module Descriptions is NOT describing a `DependencyOnArtifact` as `requiredArtifact`. `DependencyOnArtifact.category` = `MEMMAP`. Hereby `{Mip}` is composed according `<Msn>[_<vi>_<ai>]` for basic software modules where

- `<Msn>` is the `shortName` (case sensitive) of the `BswModuleDescription`
- `<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.⌋*(SRS_BSW_00465, SRS_BSW_00415, SRS_BSW_00351, SRS_BSW_00464)*

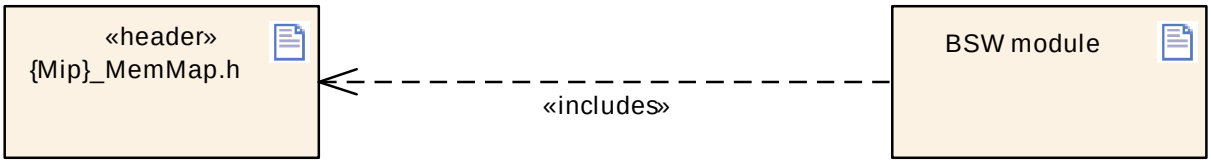

Figure 5.1: Basic Software Module specific memory mapping header file

Please note:
The approach of basic software module specific memory mapping header files implements the pattern of a user specific file split as specified in [SRS_BSW_00415]. The concrete name pattern defined in [SWS_MemMap_00032] is deviating from the naming scheme of [SRS_BSW_00415] since the module and user relationship is interpreted from the opposite way around.

[SWS_MemMap_00029] ⌈For each software component type which is part of the input configuration a software component type specific memory mapping header file `{componentTypeName}_MemMap.h` shall be provided by the Memory Mapping.⌋*(SRS_BSW_00465, SRS_BSW_00415, SRS_BSW_00351, SRS_BSW_00464)*

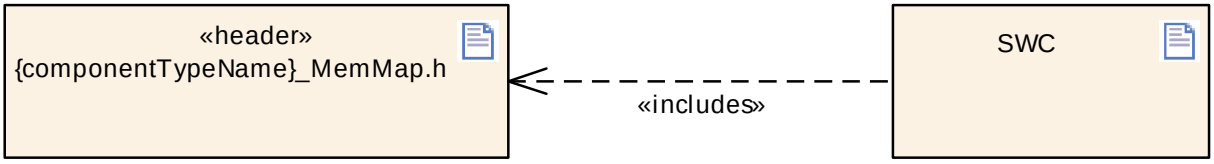

Figure 5.2: Software Component type specific memory mapping header file

6 Requirements traceability

The following tables references the requirements specified in [3] and links to the fulfillment of these. Please note that if column 'Satisfied by' is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_MemMap_00999]
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_MemMap_00999]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_MemMap_00999]
[SRS_BSW_00006]	The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	[SWS_MemMap_00003] [SWS_MemMap_00005] [SWS_MemMap_00006] [SWS_MemMap_00010] [SWS_MemMap_00036]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_MemMap_00999]
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.	[SWS_MemMap_00999]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_MemMap_00999]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_MemMap_00999]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_MemMap_00999]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_MemMap_00999]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_MemMap_00999]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_MemMap_00999]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_MemMap_00999]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_MemMap_00999]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_MemMap_00999]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_MemMap_00999]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_MemMap_00999]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_MemMap_00999]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_MemMap_00999]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_MemMap_00999]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_MemMap_00999]
[SRS_BSW_00305]	Data types naming convention	[SWS_MemMap_00999]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_MemMap_00003] [SWS_MemMap_00005] [SWS_MemMap_00006] [SWS_MemMap_00010] [SWS_MemMap_00015] [SWS_MemMap_00016] [SWS_MemMap_00018] [SWS_MemMap_00023] [SWS_MemMap_00036]
[SRS_BSW_00307]	Global variables naming convention	[SWS_MemMap_00999]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_MemMap_00999]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_MemMap_00999]
[SRS_BSW_00310]	API naming convention	[SWS_MemMap_00999]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_MemMap_00999]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_MemMap_00999]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_MemMap_00999]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_MemMap_00999]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_MemMap_00999]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_MemMap_00999]
[SRS_BSW_00327]	Error values naming convention	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_MemMap_00001] [SWS_MemMap_00005]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_MemMap_00999]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_MemMap_00999]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_MemMap_00999]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_MemMap_00999]
[SRS_BSW_00335]	Status values naming convention	[SWS_MemMap_00999]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_MemMap_00999]
[SRS_BSW_00337]	Classification of development errors	[SWS_MemMap_00999]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_MemMap_00999]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_MemMap_00999]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_MemMap_00999]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_MemMap_00999]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_MemMap_00999]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_MemMap_00999]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_MemMap_00999]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_MemMap_00999]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00351]	Encapsulation of compiler specific methods to map objects	[SWS_MemMap_00002] [SWS_MemMap_00003] [SWS_MemMap_00005] [SWS_MemMap_00006] [SWS_MemMap_00007] [SWS_MemMap_00010] [SWS_MemMap_00011] [SWS_MemMap_00013] [SWS_MemMap_00015] [SWS_MemMap_00016] [SWS_MemMap_00018] [SWS_MemMap_00020] [SWS_MemMap_00022] [SWS_MemMap_00023] [SWS_MemMap_00026] [SWS_MemMap_00027] [SWS_MemMap_00028] [SWS_MemMap_00029] [SWS_MemMap_00032] [SWS_MemMap_00033] [SWS_MemMap_00034] [SWS_MemMap_00035] [SWS_MemMap_00036] [SWS_MemMap_00037] [SWS_MemMap_00038] [SWS_MemMap_00039] [SWS_MemMap_00040] [SWS_MemMap_00041] [SWS_MemMap_00042]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_MemMap_00999]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_MemMap_00999]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_MemMap_00999]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_MemMap_00999]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_MemMap_00999]
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_MemMap_00002]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_MemMap_00999]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_MemMap_00999]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_MemMap_00999]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_MemMap_00999]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_MemMap_00999]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_MemMap_00999]
[SRS_BSW_00380]	Configuration parameters being stored in memory shall be placed into separate c-files	[SWS_MemMap_00999]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_MemMap_00999]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_MemMap_00020]
[SRS_BSW_00385]	List possible error notifications	[SWS_MemMap_00999]
[SRS_BSW_00386]	The BSW shall specify the configuration for detecting an error	[SWS_MemMap_00999]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_MemMap_00999]
[SRS_BSW_00389]	Containers shall have names	[SWS_MemMap_00999]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_MemMap_00999]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_MemMap_00999]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_MemMap_00999]
[SRS_BSW_00394]	The Basic Software Module specifications shall specify the scope of the configuration parameters	[SWS_MemMap_00999]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_MemMap_00999]
[SRS_BSW_00396]	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	[SWS_MemMap_00999]
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts	[SWS_MemMap_00999]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_MemMap_00999]
[SRS_BSW_00399]	Parameter-sets shall be located in a separate segment and shall be loaded after the code	[SWS_MemMap_00999]
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	[SWS_MemMap_00999]
[SRS_BSW_00401]	Documentation of multiple instances of configuration parameters shall be available	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00402]	Each module shall provide version information	[SWS_MemMap_00999]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_MemMap_00999]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_MemMap_00999]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_MemMap_00999]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_MemMap_00999]
[SRS_BSW_00408]	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	[SWS_MemMap_00999]
[SRS_BSW_00409]	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	[SWS_MemMap_00999]
[SRS_BSW_00410]	Compiler switches shall have defined values	[SWS_MemMap_00999]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_MemMap_00999]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_MemMap_00999]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_MemMap_00999]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_MemMap_00028] [SWS_MemMap_00029] [SWS_MemMap_00032]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_MemMap_00999]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_MemMap_00999]
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	[SWS_MemMap_00999]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the DEM	[SWS_MemMap_00999]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_MemMap_00999]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_MemMap_00999]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_MemMap_00999]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_MemMap_00999]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_MemMap_00999]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_MemMap_00999]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_MemMap_00999]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_MemMap_00999]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_MemMap_00006] [SWS_MemMap_00038]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_MemMap_00999]
[SRS_BSW_00439]	Enable BSW modules to handle interrupts	[SWS_MemMap_00999]
[SRS_BSW_00440]	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	[SWS_MemMap_00999]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_MemMap_00022]
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	[SWS_MemMap_00999]
[SRS_BSW_00448]	Module SWS shall not contain requirements from Other Modules	[SWS_MemMap_00999]
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ ReturnType	[SWS_MemMap_00999]
[SRS_BSW_00450]	A Main function of a un-initialized module shall return immediately	[SWS_MemMap_00999]
[SRS_BSW_00451]	Hardware registers shall be protected if concurrent access to these registers occur	[SWS_MemMap_00999]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_MemMap_00999]
[SRS_BSW_00453]	BSW Modules shall be harmonized	[SWS_MemMap_00999]
[SRS_BSW_00454]	An alternative interface without a parameter of category DATA_REFERENCE shall be available.	[SWS_MemMap_00999]
[SRS_BSW_00456]	A Header file shall be defined in order to harmonize BSW Modules	[SWS_MemMap_00999]
[SRS_BSW_00457]	Callback functions of Application software components shall be invoked by the Basis SW	[SWS_MemMap_00999]
[SRS_BSW_00458]	Classification of production errors	[SWS_MemMap_00999]
[SRS_BSW_00459]	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	[SWS_MemMap_00999]
[SRS_BSW_00460]	Reentrancy Levels	[SWS_MemMap_00999]
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module	[SWS_MemMap_00999]

Requirement	Description	Satisfied by
[SRS_BSW_00462]	All Standardized Autosar Interfaces shall have unique requirement Id / number	[SWS_MemMap_00999]
[SRS_BSW_00463]	Naming convention of callout prototypes	[SWS_MemMap_00999]
[SRS_BSW_00464]	File names shall be considered case sensitive regardless of the filesystem in which they are used	[SWS_MemMap_00028] [SWS_MemMap_00029] [SWS_MemMap_00032]
[SRS_BSW_00465]	It shall not be allowed to name any two files so that they only differ by the cases of their letters	[SWS_MemMap_00028] [SWS_MemMap_00029] [SWS_MemMap_00032]
[SRS_BSW_00466]	Classification of extended production errors	[SWS_MemMap_00999]
[SRS_BSW_00467]	The init / deinit services shall only be called by BswM or EcuM	[SWS_MemMap_00999]
[SRS_BSW_00469]	Fault detection and healing of production errors and extended production errors	[SWS_MemMap_00999]
[SRS_BSW_00470]	Execution frequency of production error detection	[SWS_MemMap_00999]
[SRS_BSW_00471]	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	[SWS_MemMap_00999]
[SRS_BSW_00472]	Avoid detection of two production errors with the same root cause.	[SWS_MemMap_00999]
[SRS_BSW_00473]	Classification of transient faults	[SWS_MemMap_00999]
[SRS_BSW_00477]	The functional interfaces of AUTOSAR BSW modules shall be specified in C99	[SWS_MemMap_00003] [SWS_MemMap_00018] [SWS_MemMap_00023]
[SRS_BSW_00478]	Timing limits of main functions	[SWS_MemMap_00999]
[SRS_BSW_00479]	Interfaces for handling request from external devices	[SWS_MemMap_00999]
[SRS_BSW_00480]	NullPointer Errors shall follow a naming rule	[SWS_MemMap_00999]
[SRS_BSW_00481]	Invalid configuration set selection errors shall follow a naming rule	[SWS_MemMap_00999]
[SRS_BSW_00482]	Get Version Informationfunction shall follow a naming rule	[SWS_MemMap_00999]

7 Functional specification

7.1 General issues

The memory mapping files include the compiler and linker specific keywords for memory allocation into header and source files. These keywords control the assignment of variables and functions to specific sections. Thereby implementations are independent from compiler and microcontroller specific properties. The assignment of the sections to dedicated memory areas / address ranges is not the scope of the memory mapping file and is typically done via linker control files.

[SWS_MemMap_00001] ⌈For each build scenario (e.g. Boot loader, ECU Application) an own set of memory mapping files has to be provided.⌋*(SRS_BSW_00328)*

[SWS_MemMap_00002] ⌈The memory mapping file name shall be `{Mip}_MemMap.h` for basic software modules and `{componentTypeName}_MemMap.h` for software components where `{Mip}` is the Module implementation prefix and `{componentTypeName}` is the name of the software component type.⌋*(SRS_BSW_00361, SRS_BSW_00351)*

Please note that the information of `{Mip}` is taken from the Basic Software Module Description of the related BSW module as described in [SWS_MemMap_00028] and [SWS_MemMap_00032].

[SWS_MemMap_00010] ⌈If a compiler/linker does not require specific commands to implement the functionality of SWS Memory Mapping, the Memory Allocation Keyword defines might be undefined without further effect.⌋*(SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351)*

[SWS_MemMap_00036] ⌈If a compiler/linker does not support mandatory functionality for the kind of MemorySection used by the BSW module or software component the Memory Allocation Keyword shall be defined to raise an error.⌋*(SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351)*

Example 7.1

```
#ifdef EEP_START_SEC_VAR_CLEARED_16
    #undef EEP_START_SEC_VAR_CLEARED_16
#endif
```

As described in [SWS_MemMap_00029] the number of files depends on the number of `SwComponentType`s in the input configuration. To determine the number of `MemorySection`s the applicable `SwcImplementation`s have to be known. These are described in an AUTOSAR environment with the `SwcToImplMapping` in the `SystemMapping` and / or via ECU Configuration values `RteImplementationRef` in a `RteSwComponentType` container.
Knowing the `SwcImplementation`s provides as well the number of `MemorySec-`

tions which have to be identified for [SWS_MemMap_00027]. For more details about the content of a SwcImplementation see document [4] and [5].

Further on the total number of used MemorySections depends as well on the number of used BSW modules. These can be determined by the M1 instance of the EcucValueCollection which refers to the MemMap's EcucModuleConfigurationValues. This EcucValueCollection refers as well to EcucModuleConfigurationValues of other Bsw Modules which refer again to BswImplementations via moduleDescription references. Knowing the BswImplementations provides as well the number of MemorySections which have to be identified for [SWS_MemMap_00026]. For more details about the content of a BswImplementation see document [5].

In [6] further information is provided how Memory Mapping is used in the AUTOSAR Methodology.

7.2 Mapping of variables and code

7.2.1 Requirements on implementations using memory mapping header files for BSW Modules and Software Components

[SWS_MemMap_00038] gives a recommendation to the granularity in which the different types of variables and code should be allocated in a C implementation. The referenced subsection 7.2.1.3 and subsection 7.2.1.4 defines the recommended names for those memory allocation keywords. Nevertheless a implementation may deviate from this recommendations, e.g. to implement supplementary requirements.

[SWS_MemMap_00038] ⌈

Each AUTOSAR basic software module and software component should support the configuration of at least the following different memory types as described in

- Table 7.2: Section Type VAR
- Table 7.3: Section Type VAR_FAST
- Table 7.4: Section Type VAR_SLOW
- Table 7.5: Section Type INTERNAL_VAR
- Table 7.6: Section Type VAR_SAVED_ZONE
- Table 7.7: Section Type CONST_SAVED_RECOVERY_ZONE
- Table 7.8: Section Type CONST
- Table 7.9: Section Type CALIB
- Table 7.10: Section Type CONFIG_DATA

- Table 7.11: Section Type CODE
- Table 7.12: Section Type CALLOUT_CODE
- Table 7.13: Section Type CODE_FAST
- Table 7.14: Section Type CODE_SLOW

It is allowed to add module specific sections as they are mapped and thus are configurable within the module's configuration file.

The shortcut `{ALIGNMENT}` means the typical variable alignment. In order to avoid memory gaps variables are allocated separately according their size for the kind of memory sections where a high amount of variables is expected, e.g. VAR. Hereby it is the task of the implementer to ensure the proper granularity by defining memory sections with different `{ALIGNMENT}` postfixes for variables of different element sizes as described below.

It is the integrator's job to ensure via appropriate memory mapping configuration (i.e. using the proper alignment #pragmas or omitting them at all to let the compiler decide) that the platform specific alignment requirements of objects of the respective *size* are honored. Thereby the effective alignment can deviate from the `{ALIGNMENT}` post-fix.

`BOOLEAN`, used for variables and constants of size 1 bit

`8`, used for variables and constants which typically have to be aligned to 8 bit. For instance used for variables and constants of size 8 bit or used for composite data types: arrays, structs and unions containing elements of maximum 8 bits.

`16`, used for variables and constants which typically have to be aligned to 16 bit. For instance used for variables and constants of size 16 bit or used for composite data types: arrays, structs and unions containing elements of maximum 16 bits.

`32`, used for variables and constants which typically have to be aligned to 32 bit. For instance used for variables and constants of size 32 bit or used for composite data types: arrays, structs and unions containing elements of maximum 32 bits.

`PTR`, used for variables and constants whose value is the address of another variable, so called pointers.

`UNSPECIFIED`, used for variables, constants, structure, array and unions when *size* (alignment) does not fit the criteria of 8,16, 32 bit or PTR. For instance used for variables and constants of unknown size

In case structures and unions, it shall be allowed to use an alignment larger than the bit size of the elements. For instance to facilitate copy instruction a structure may have minimum 2 byte alignment, even if members are byte aligned. In this case, it should be possible to use alignment 16 bit instead of 8 bit for this structure.

Note: The (embedded) application binary interface ((E)ABI) of some target architectures (e.g., TriCore) imposes additional alignment requirements on aggregate types type (e.g., structs) depending on the size of the structure. Those additional constraints

do not need to be taken in consideration when selecting the {ALIGNMENT} post-fix of the Memory Allocation Keyword for variables and constants of those aggregate types.

The shortcut {INIT_POLICY} means the initialization policy of variables. Possible INIT_POLICY postfixes are:

- CLEARED, used for not explicitly initialized variables.
- INIT, used for initialized variables. This are typically explicitly initialized variables, but it can be also used for not explicitly initialized variables to be able to mix up both types to deal with legacy code.
- POWER_ON_CLEARED, used for variables that are not explicitly initialized (cleared) during normal start-up. Instead these are cleared only after power on reset of the microcontroller or in case of battery backup memory the memory itself.

For more details and examples please refer to the table below.

Note: The postfixes NO_INIT and POWER_ON_INIT are still supported but deprecated and will be removed in one of the next releases.

Use INIT or CLEARED also for those variables which might be initialized at a later time in the program flow, e.g. by an initialization routine. POWER_ON_CLEARED shall be used for variables which shall survive resets only.

For optimizing the initialization at start-up, it is possible for any software vendor to apply an initialization policy refinement inside the SwAddrMethod name, e.g.:

- <PREFIX>_SEC_VAR_POWER_ON_CLEARED_RSTSAFE_QM_8, used to express reset safe variables.
- <PREFIX>_SEC_VAR_POWER_ON_CLEARED_NVRAM_QM_8, used to express that the section contains NVRAM buffers.
- <PREFIX>_SEC_VAR_POWER_ON_CLEARED_BATTERY_BACKUP_QM_8, used to express that the memory is a special battery backup device.
- <PREFIX>_SEC_VAR_INIT_INDETERMINATE_QM_8, used to express that the section contains NVRAM buffers.
- <PREFIX>_SEC_VAR_INIT_SELFINIT_QM_8, used to express that the memory is a special battery backup device.

Depending on the used SwAddrMethod one can derive options to map to individual ModeSets and so to different memory devices in the target project.

Note 1: For microcontrollers / processors which are equipped with Error Correction Codes (ECC), the hardware needs to initialize the according memory in case of under voltage due to lost ECC. This includes:

- Any 'normal' system RAM without external supply, which needs to be initialized when the microcontroller voltage drops below a threshold as the ECC codes become invalid. This usually happens in case of a cold power on reset.

- Any 'standby' supplied RAM, which needs to be initialized when the standby voltage drops below a threshold and the ECC codes become invalid.

As a consequence POWER_ON_CLEARED symbols cannot be stored inside of those memory areas.

Note 2: Please consider that microcontrollers / processors with embedded LBIST (Logical Build In Self Test), MBIST (Memory Build In Self Test) will initialize a specified amount of memory when those tests are executed. So these memory devices shall not be used for POWER_ON_CLEARED.⌋ *(SRS_BSW_00437, SRS_BSW_00351)*

Init Policy	Allowed for	Type	Example	Initialization Time	Behavior	Note
CLEARED	Not explicitly initialized variables	BSS	uint8 my_bss; /* =0 */	any reset	All objects are initialized to 0 or null pointer as per C standard (6.7.8 Initialization clause 10).	This is typically used for not explicitly initialized objects with a static storage duration.
INIT	Initialized variables	DATA	uint8 my_data=5;	any reset, copytable execution	All objects are initialized according to their initializer.	This is typically used for either initialized or not explicitly initialized objects with a static storage duration. Note: Depending on the used compiler it might not be possible to combine DATA and BSS initialization due to limited #pragmas.
		BSS	uint8 my_bss; /* =0 */		All objects are initialized to 0 or null pointer as per C standard (6.7.8 Initialization clause 10).	
POWER_ON_CLEARED	Power-on cleared variables	BSS	uint8 my_bss;	Cold PowerOn reset	All objects are initialized to 0 or null pointer, but only on Cold PowerOn reset or brownout reset. They are not overwritten on a regular warm reset (e.g. software reset, watchdog reset, external reset).	This deviates from the C standard as all objects with a static storage duration shall be initialized before program startup (5.1.2 Execution environments).

Table 7.1: Summary of Init Behavior

[SWS_MemMap_00022] ⌈The keywords to be used before inclusion of the memory mapping header file shall use the templates <PREFIX>_START_SEC_<NAME> or <PREFIX>_STOP_SEC_<NAME>

Where:

- <PREFIX> is composed according <snp>[_<vi>_<ai>] for basic software modules where
  - <snp> is the Section Name Prefix which shall be the Module Abbreviation from the BSW Module list (e.g. "EEP" or "CAN") in upper case letters of the BSW module. For the generation of the MemMap.h file following rules apply:
    - <snp> shall be the BswModuleDescription's shortName converted in upper case letters if no SectionNamePrefix is defined for the MemorySection.
    - <snp> shall be the symbol of the SectionNamePrefix associated to the MemorySection if a SectionNamePrefix is defined for the MemorySection.
  - <vi> is the vendorId of the BSW module, which shall be in upper case.
  - <ai> is the vendorApiInfix of the BSW module, which shall be in upper case.

  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.

  OR
- <PREFIX> is the shortName of the software component type for software components (case sensitive) if no SectionNamePrefix is defined for the MemorySection.

  OR
- <PREFIX> is the symbol of the SectionNamePrefix if a SectionNamePrefix is defined for the MemorySection.

  AND
- <NAME> is the shortName of the MemorySection described in Basic Software Module Description or a Software Component Description (case sensitive) if the MemorySection has no symbol attribute defined.

  OR
- <NAME> is the symbol of the MemorySection described in Basic Software Module Description or a Software Component Description (case sensitive) if the MemorySection has a symbol attribute defined.

⌋(*SRS_BSW_00441*, *SRS_BSW_00351*)

Please note if the Memory Allocation Keywords shall appear in capital letters in the code the related MemorySections in the Basic Software Module Description or Software Component Description have to be named with capital letters.

[SWS_MemMap_00037] ⌈The part <NAME> from [SWS_MemMap_00022] may contain the following ASIL keywords to indicate the restriction/qualifications: {safety} = QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D

The {safety} tag is optional and indicates the maximum possible safety level. Downscaling in the project is possible inside memory mapping header files. If no {safety} keyword is added the default shall be treated as QM (without any ASIL qualification).⌋ *(SRS_BSW_00351)*

[SWS_MemMap_00039] ⌈The part <NAME> from [SWS_MemMap_00022] shall contain the following core scope keywords to indicate the restriction/qualifications: {coreScope} =

- GLOBAL is code/data which can be executed/accessed by any core in case of multi-core ECUs.
- LOCAL code/data must be mapped by the integrator to a specific core (Core 0, Core 1, ...) in case of multi-core ECUs.

The {coreScope} tag can be considered optional in case it is GLOBAL. Means in case no {coreScope} keyword is added the default shall be treated as GLOBAL. Instead the {coreScope} tag LOCAL shall be considered as mandatory.

Background: Scope of LOCAL SwAddrMethod is visible and can not be mixed up with GLOBAL SwAddrMethods.

⌋*(SRS_BSW_00351)*

[SWS_MemMap_00042] ⌈For variable section types, the part <NAME> from [SWS_MemMap_00022] may contain an optional vendor specific {refinement} tag. It shall be used to refine the variable allocation or initialization behavior. The used values are vendor specific and free of choice.⌋*(SRS_BSW_00351)*

The usage of {coreScope} LOCAL is limited to the section types it is specified for. In addition for section types VAR, VAR_FAST, VAR_SLOW, INTERNAL_VAR the usage of {coreScope} is only perrmitted for {INIT_POLICY} equal to CLEARED or INIT. This restriction shall reduce the complexity of memory layouts and reduce the amount of memory holes due to typical allocation restrictions non initialized memory sections.

In this regard the [constr_1402] in the document [4] is defined.

Application hint: It's an integrator decision to map memory section with the GLOBAL property to a core specific memory section. For instance this can be utilized to optimize the performance if the majority of memory accesses occur from a specific core. Nevertheless such a mapping prerequisites, that the core specific memory is also accessible by the other cores.

Please note that the name part <NAME> according [SWS_MemMap_00022] is provided either by MemorySection.shortName or MemorySection.symbol. In order to provide the safety information the name part according [SWS_MemMap_00037] needs to be part of the MemorySection.shortName or MemorySection.symbol respectively. To provide the core scope qualification the name part according [SWS_MemMap_00039] needs to be part of the MemorySection.shortName or MemorySection.symbol.

Therefore the usual patterns for Memory Allocation Keywords are

```
{PREFIX}_START_SEC_CODE[_{codePeriode}][_{safety}][_{coreScope}]

{PREFIX}_STOP_SEC_CODE[_{codePeriode}][_{safety}][_{coreScope}]

{PREFIX}_START_SEC_VAR_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}

{PREFIX}_STOP_SEC_VAR_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}

{PREFIX}_START_SEC_CONST[_{accessPeriod}][_{safety}][_{coreScope}]

{PREFIX}_STOP_SEC_CONST[_{accessPeriod}][_{safety}][_{coreScope}]
```

Those are applied in the recommendations provided in subsection 7.2.1.3 and subsection 7.2.1.4.

7.2.1.1 Splitting of modules in allocatable memory parts

To increase the performance some multi core architectures work with core local memory areas. As a consequence the access speed to specific memory areas depends on the core where the code is executed. For instance a BSW module which is multi core capable by implementation of the Master/Satellite-approach is usually beneficial to split the interface of the BSW module from the "Master" functionality implementation. Another use case is to split a BSW module with several distinct features in different memory parts. Those memory parts are typically composed out of a set of sections (CODE, CONST, VAR) used or the implementation of the feature. This support that those memory parts can be assigned to set of physical controller memories being close to the main user of the feature.

[SWS_MemMap_00040] ⌈When a BSW module or Software Component is split into allocatable memory parts the <PREFIX> as described in [SWS_MemMap_00022] shall be sub-structured in the following way:

<PREFIX> = <snp>[_<vi>_<ai>]_<feature>⌋*(SRS_BSW_00351)*

[SWS_MemMap_00041] ⌈When a BSW module or Software Component is split into allocatable memory parts the resulting <PREFIX> as specified in [SWS_MemMap_00040] (inclusive [_<vi>_<ai>]) shall be described as a SectionNamePrefix and all belonging MemorySections.prefix needs to reference the SectionNamePrefix.⌋*(SRS_BSW_00351)*

Please note the example given in 7.3.5.

7.2.1.2 config constants versus non-config constants

There are basically two different kinds of constants in the implementation of an AUTOSAR BSW Module.

1. Constants which are used to implement a configurable behavior. For the different config classes of config data (i.e. everything that is placed in `<Mip>_Lcfg.c` and `<Mip>_PBcfg.c`) the syntax of Memory Allocation Keywords are:

   ```
   {PREFIX}_START_SEC_CONFIG_DATA_{configClass}[_{safety}]_{ALIGNMENT}

   {PREFIX}_STOP_SEC_CONFIG_DATA_{configClass}[_{safety}]_{ALIGNMENT}
   ```

   Note: `{configClass}` may only be `PREBUILD` or `POSTBUILD`. Thereby `PREBUILD` represents both `Pre-Compile time` and `Link time` configuration data.

   See table 7.10.

2. Constants which are used to implement a fixed value which is not related to the configuration methodology of AUTOSAR. For non-config constants (i.e. everything that is placed in `<Mip>.[ch]` or `<Mip>_<Implementation Extension>.[ch]`) the Syntax of Memory Allocation Keywords are:

   ```
   {PREFIX}_START_SEC_CONST[_{accessPeriod}][_{safety}][_{coreScope}]

   {PREFIX}_STOP_SEC_CONST[_{accessPeriod}][_{safety}][_{coreScope}]
   ```

   See table 7.8.

7.2.1.3 Data Sections

The table below defines recommended keywords for variable and constant sections:

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_{INIT_POLICY}[_{safety}][_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_{INIT_POLICY}[_{safety}][_{coreScope}]_{ALIGNMENT}`
Description	To be used for all global or static variables. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, coreLocal}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	VAR
Section Initialization Policy	`{INIT_POLICY}`

Status	–

Table 7.2: Section Type VAR

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_FAST_{INIT_POLICY}[_{safety}] [_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_FAST_{INIT_POLICY}[_{safety}] [_{coreScope}]_{ALIGNMENT}`
Description	To be used for all global or static variables. 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. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, coreLocal}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	VAR
Section Initialization Policy	`{INIT_POLICY}`
Status	–

Table 7.3: Section Type VAR_FAST

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_SLOW_{INIT_POLICY}[_{safety}] [_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_SLOW_{INIT_POLICY}[_{safety}] [_{coreScope}]_{ALIGNMENT}`

Description	To be used for all infrequently accessed global or static variables. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, core Local}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	VAR
Section Initialization Policy	`{INIT_POLICY}`
Status	–

Table 7.4: Section Type VAR_SLOW

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_INTERNAL_VAR_{INIT_POLICY}[_{safety}] [_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_INTERNAL_VAR_{INIT_POLICY}[_{safety}] [_{coreScope}]_{ALIGNMENT}`
Description	To be used for global or static variables those are accessible from a calibration tool. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, core Local}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	VAR
Section Initialization Policy	`{INIT_POLICY}`
Status	–

Table 7.5: Section Type INTERNAL_VAR

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_SAVED_ZONE{anyName Part}[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_SAVED_ZONE{anyName Part}[_{safety}]_{ALIGNMENT}`

Description	To be used for RAM buffers of variables saved in non volatile memory. {anyNamePart} denotes the specific content of the saved zone. In the related SwAddrMethod the sectionInitializationPolicy attribute shall be set to POWER-ON-CLEARED. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted.
Memory Section Type	VAR
Section Initialization Policy	`POWER-ON-CLEARED`
Status	–

Table 7.6: Section Type VAR_SAVED_ZONE

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CONST_SAVED_RECOVERY_ZONE{anyNamePart}[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CONST_SAVED_RECOVERY_ZONE{anyNamePart}[_{safety}]_{ALIGNMENT}`
Description	To be used for ROM buffers of variables saved in non volatile memory. {anyNamePart} denotes the specific content of the recovery zone. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted.
Memory Section Type	CONST
Section Initialization Policy	–
Status	–

Table 7.7: Section Type CONST_SAVED_RECOVERY_ZONE

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CONST[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CONST[_{safety}]_{ALIGNMENT}`

Description	To be used for global or static constants. {accessPeriod} is the typical period time value and unit of the ExecutableEntitys in this MemorySection. The name part _{accessPeriod} 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 on in special modes of the ECU the code may be scheduled with a higher or lower period. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted.
Memory Section Type	CONST
Section Initialization Policy	–
Status	–

Table 7.8: Section Type CONST

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CALIB[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CALIB[_{safety}]_{ALIGNMENT}`
Description	To be used for calibration constants. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted.
Memory Section Type	CALPRM
Section Initialization Policy	–
Status	–

Table 7.9: Section Type CALIB

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CONFIG_DATA[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CONFIG_DATA[_{safety}]_{ALIGNMENT}`

Description	Constants with attributes that show that they reside in one segment for module configuration. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part {configClass} shall contain the configClass with one of the strings PREBUILD or POSTBUILD. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the config Class with the possible values {configClassPreBuild, configClassPostBuild}.
Memory Section Type	CONFIG-DATA
Section Initialization Policy	–
Status	–

Table 7.10: Section Type CONFIG_DATA

7.2.1.4 Code Sections

There are different kinds of execution code sections. This code sections shall be identified with dedicated keywords. If a section is not supported by the integrator and micro controller then be aware that the keyword is ignored. The table below defines recommended keywords for code sections:

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CODE[_{codePeriod}][_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CODE[_{codePeriod}][_{safety}][_{coreScope}]`

Description	To be used for mapping code to application block, boot block, external flash etc. {codePeriod} is the typical period time value and unit of the ExecutableEntitys in this MemorySection. The name part _{codePeriod} 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 on in special modes of the ECU the code may be scheduled with a higher or lower period. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, core Local}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	CODE
Section Initialization Policy	–
Status	–

Table 7.11: Section Type CODE

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CALLOUT_CODE[_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CALLOUT_CODE[_{safety}][_{coreScope}]`
Description	To be used for mapping callouts of the BSW Modules which shall typically use the global linker settings for callouts. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, core Local}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	CODE
Section Initialization Policy	–

Status	–

Table 7.12: Section Type CALLOUT_CODE

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CODE_FAST[_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CODE_FAST[_{safety}][_{coreScope}]`
Description	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 times but with the knowledge of high frequent access and /or high execution time. For example, a callback for a frequent notification. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, coreLocal}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	CODE
Section Initialization Policy	–
Status	–

Table 7.13: Section Type CODE_FAST

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CODE_SLOW[_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CODE_SLOW[_{safety}][_{coreScope}]`
Description	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 times but with the knowledge of low frequent access. For example, a callback in case of seldom error. The name part _{safety} shall contain the safety integrity level with at most one of the strings QM, ASIL_A, ASIL_B, ASIL_C, ASIL_D. In case of QM the name part may be omitted. The name part _{coreScope} shall contain the core scope qualification with at most one of the strings GLOBAL, LOCAL. In case of GLOBAL the name part may be omitted. In the related SwAddrMethod one option attribute shall describe the safety integrity level with the possible values {safetyQM, safetyAsilA, safetyAsilB, safetyAsilC, safetyAsilD}. In case of safetyQM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the core scope qualification with at most one of the possible values {coreGlobal, coreLocal}. In case of coreGlobal the attribute may be omitted.
Memory Section Type	CODE
Section Initialization Policy	–

Status	–

Table 7.14: Section Type CODE_SLOW

[SWS_MemMap_00003] ⌈Each AUTOSAR basic software module and software component shall wrap declaration and definition of code, variables and constants using the following mechanism:

1. Definition of start symbol for module memory section
2. Inclusion of the memory mapping header file
3. Declaration/definition of code, variables or constants belonging to the specified section
4. Definition of stop symbol for module memory section
5. Inclusion of the memory mapping header file

For code which is invariably implemented as inline function the wrapping with Memory Allocation Keywords is not required.⌋*(SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351, SRS_BSW_00477)*

Application hint:
The implementations of AUTOSAR basic software modules or AUTOSAR software components are not allowed to rely on an implicit assignment of objects to default sections because properties of default sections are platform and tool dependent. Therefore this style of code implementation is not platform independent.

Application hint:
For code which is implemented with the `LOCAL_INLINE` macro of the `"Compiler.h"` the wrapping with Memory Allocation Keywords is required. In the case that the `LOCAL_INLINE` is set to the inline keyword of the compiler the related Memory Allocation Keywords shall not define any linker section assignments or change the addressing behavior because this is already set by the environment of the calling function where the code is inlined. In the case that the `LOCAL_INLINE` is set to empty the related Memory Allocation Keywords shall be configured like for regular code. For code which his implemented with the `INLINE` macro of the `"Compiler.h"` the wrapping with Memory Allocation Keywords is required at least for the code which is remaining if `INLINE` is set to empty.

Please note as well that in the Basic Software Module Description the `MemorySection` related to the used Memory Allocation Keywords has to document the usage of `INLINE` and `LOCAL_INLINE` in the option attribute. For further information see [5].

Additional option attribute values are predefined in document [4], [TPS_SWCT_01456].

The inclusion of the memory mapping header files within the code is a MISRA violation. As neither executable code nor symbols are included (only pragmas) this violation is an approved exception without side effects.

The start and stop symbols for section control are configured with section identifiers defined in the inclusion of memory mapping header file. For details on configuring sections see " Configuration specification".

Example 7.2

For example (BSW Module):

```
#define EEP_START_SEC_VAR_INIT_16
#include "Eep_MemMap.h"
static uint16 EepTimer = 100;
static uint16 EepRemainingBytes = 16;
#define EEP_STOP_SEC_VAR_INIT_16
#include "Eep_MemMap.h"
```

Example 7.3

For example (SWC):

```
#define Abc_START_SEC_CODE
#include "Abc_MemMap.h"
/* --- Write a Code here */
#define Abc_STOP_SEC_CODE
#include "Abc_MemMap.h"
```

[SWS_MemMap_00018] ⌈Each AUTOSAR basic software module and software component shall support, for all C-objects, the configuration of the assignation to one of the memory types (code, variables and constants).⌋*(SRS_BSW_00306, SRS_BSW_00351, SRS_BSW_00477)*

[SWS_MemMap_00023] ⌈Memory mapping header files shall not be included inside the body of a function.⌋*(SRS_BSW_00306, SRS_BSW_00351, SRS_BSW_00477)*

The goal of this requirement is to support compiler which do not support `#pragma` inside the body of a function. To force a special memory mapping of a function's static variable, this variable must be moved to file static scope.

Application hint concerning callout sections:

According [SWS_BSW_00135] an individual set of memory allocation keywords per callout function shall be used. This provides on one hand a high flexibility for the configuration of memory allocation. On the other hand this bears the risk of high configuration effort for the `MemMap` module because all individual memory sections have to be configured for the MemMap header file generation. To ease the integration of such callout sections it is recommended that in the Basic Software Module Description all `MemorySection`s which are describing callouts and which typically are treated with the same linker properties should refer to the identical `SwAddrMethod`. According the recommended memory sections in section 7.2.1.4 "code sections" the `SwAddrMethod` defined by AUTOSAR would have the reference path:

`/AUTOSAR_MemMap/SwAddrMethods/CALLOUT_CODE`

For instance:

```
<MEMORY-SECTION>
  <SHORT-NAME>COM_SOMECALLOUT_CODE</SHORT-NAME>
  <SW-ADDRMETHOD-REF DEST="SW-ADDR-METHOD">/
     AUTOSAR_MemMap/SwAddrMethods/CALLOUT_CODE</SW-
     ADDRMETHOD-REF>
</MEMORY-SECTION>
```

This enables the integrater either to configer all of the memory sections identical with the means of the MemMapGenericMapping and additionally to handle the special cases individually with the means of the MemMapSectionSpecificMapping. See as well the example 7.3.4 Callout sections

7.2.2 Requirements on memory mapping header files

[SWS_MemMap_00005] ⌈The memory mapping header files shall provide a mechanism to select different code, variable or constant sections by checking the definition of the module specific Memory Allocation Key Words for starting a section (see [SWS_MemMap_00038]). Code, variables or constants declared after this selection shall be mapped to this section.⌋*(SRS_BSW_00328, SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351)*

[SWS_MemMap_00026] ⌈Each BSW memory mapping header file shall support the Memory Allocation Keywords to start and to stop a section for each belonging MemorySection defined in a BswImplementation which is part of the input configuration.⌋*(SRS_BSW_00351)*

[SWS_MemMap_00033] ⌈All MemorySections defined in a BswImplementation belong to the {Mip}_MemMap.h memory mapping header file if the BswImplementation does NOT contain a DependencyOnArtifact as requiredArtifact.DependencyOnArtifact.category = MEMMAP⌋*(SRS_BSW_00351)*

Please note also [SWS_MemMap_00032].

[SWS_MemMap_00034] ⌈All MemorySection defined in a BswImplementation belong to the memory mapping header file defined by the attribute requiredArtifact.artifactDescriptor.shortLabel if the BswImplementation does contain exactly one DependencyOnArtifact as requiredArtifact.DependencyOnArtifact.category = MEMMAP⌋*(SRS_BSW_00351)*

Please note also [SWS_MemMap_00028].

[SWS_MemMap_00035] ⌈All MemorySection defined in a BswImplementation and associated with the identical SectionNamePrefix belong to the memory mapping header file defined by the attribute requiredArtifact.artifactDescriptor.shortLabel of the DependencyOnArtifact which is referenced by the SectionNamePrefix with a implementedIn reference.⌋*(SRS_BSW_00351)*

In this case the if the BswImplementation may contain several DependencyOnArtifact as with requiredArtifact. DependencyOnArtifact.category = MEMMAP This will be used to describe an ICC2 cluster with one BswModuleDescription. Please note also [SWS_MemMap_00028].

[SWS_MemMap_00027] ⌈The software component type specific memory mapping header file {componentTypeName}_MemMap.h shall support the Memory Allocation Keywords to start and to stop a section for each MemorySection defined in a SwcImplementation associated of this software component type.⌋*(SRS_BSW_00351)*

[SWS_MemMap_00015] ⌈The selected section shall be activated, if the section macro is defined before include of the memory mapping header file.⌋*(SRS_BSW_00306, SRS_BSW_00351)*

[SWS_MemMap_00016] ⌈The selection of a section shall only influence the linker's behavior for one of the three different object types code, variables or constants concurrently.⌋*(SRS_BSW_00306, SRS_BSW_00351)*

Application hint:
On one side the creation of combined sections (for instance code and constants) is not allowed. For the other side the set-up of the compiler / linker must be done in a way, that only the settings of the selected section type is changed. For instance the set-up of the code section shall not influence the configuration of the constant section and other way around.

Example 7.4

```
#ifdef EEP_START_SEC_VAR_INIT_16
    #undef EEP_START_SEC_VAR_INIT_16
    #define START_SECTION_DATA_INIT_16
#elif
/*
   additional mappings of modules sections into project
   sections
*/
...
#endif


#ifdef START_SECTION_DATA_INIT_16
    #pragma section data "sect_data16"
    #undef START_SECTION_DATA_INIT_16
    #undef MEMMAP_ERROR
#elif
/*
   additional statements for switching the project sections
*/
...
#endif
```

Application hint:
Those code or variables sections can be used for the allocation of objects from more

than one module.
Those code or variables sections can be used for the allocation of objects from different module specific code or variable sections of one module.

[SWS_MemMap_00006] ⌈The memory mapping header files shall provide a mechanism to deselect different code and variable sections by checking the definition of the module specific Memory Allocation Key Words for stopping a section (see [SWS_MemMap_00038]).

The selected section shall be deactivated if the section macro is defined before include of the memory mapping header file. Code or variables declared after this selection shall be mapped to an section collecting those inaccurate non-handled objects from BSW Module or software component implementation.[1]⌋*(SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00437, SRS_BSW_00351)*

Example 7.5

```
#ifdef EEP_STOP_SEC_CODE
    #undef EEP_STOP_SEC_CODE
    #define STOP_SECTION_COMMON_CODE
#elif
/*
   additional mappings of modules sections into project
   sections
*/
...
#endif


/* additional module specific mappings */
...

#ifdef STOP_SECTION_COMMON_CODE
    #pragma section code restore
    #undef STOP_SECTION_COMMON_CODE
    #undef MEMMAP_ERROR
#elif
/*
   additional statements for switching the project sections
*/
#endif
```

[SWS_MemMap_00007] ⌈The memory mapping header files shall check if they have been included with a valid memory mapping symbol and in a valid sequence (no START preceded by a START, no STOP without the corresponding START). This shall be done by a preprocessor check.⌋*(SRS_BSW_00351)*

Example 7.6

[1] Since its error prone to determined expected properties for memory which is not explicitly handled by Memory Allocation Key Words usually those objects are treated in away to cause linker errors. The default sections might be used to catch those non-handled objects.

```
#define MEMMAP_ERROR

/*
   mappings of modules sections into project sections and
   statements for switching the project sections
*/

...
#elif STOP_SECTION_COMMON_CODE
    #pragma section code restore
    #undef STOP_SECTION_COMMON_CODE
    #undef MEMMAP_ERROR
#endif

#ifdef MEMMAP_ERROR
  #error "Eep_MemMap.h, wrong pragma command"
#endif
```

[SWS_MemMap_00011] ⌈The memory mapping header files shall undefine the module or software component specific Memory Allocation Key Words for starting or stopping a section.⌋*(SRS_BSW_00351)*

Example 7.7

```
#ifdef EEP_STOP_SEC_CODE
    #undef EEP_STOP_SEC_CODE
```

[SWS_MemMap_00013] ⌈The memory mapping header files shall use if-else structures to reduce the compilation effort.⌋*(SRS_BSW_00351)*

Example 7.8

For instance:

```
#define MEMMAP_ERROR
...
/* module and ECU specific section mappings */
#if defined START_SECTION_COMMON_CODE
    #pragma section ftext
    #undef START_SECTION_COMMON_CODE
    #undef MEMMAP_ERROR
#elif defined START_SECTION_UNBANKED_CODE
    #pragma section code text
    #undef START_SECTION_UNBANKED_CODE
    #undef MEMMAP_ERROR
#elif defined ...
...

#endif
```

7.3 Examples

The examples in this section shall illustrate the relationship between the Basic Software Module Descriptions, Software Component Descriptions, the ECU configuration of the Memory Mapping and the Memory Mapping header files.

7.3.1 Code Section

The following example shows `ApplicationSwComponentType` "MySwc" which contains in its `SwcInternalBehavior` a `RunnableEntity` "Run1". The `RunnableEntity` "Run1" references the `SwAddrMethod` "CODE" which `sectionType` attribute is set to `code`. This expresses the request to allocate the `RunnableEntity` code into a code section with the name "CODE".

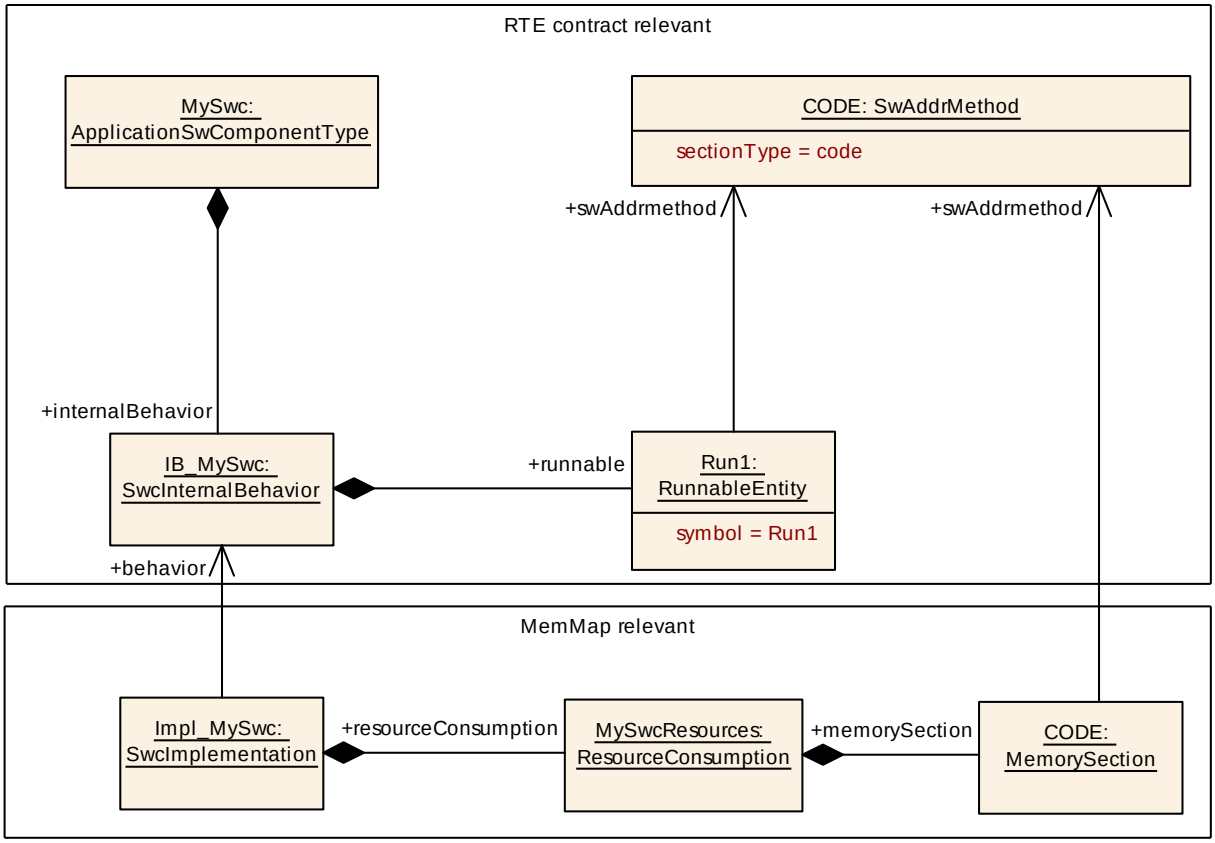

Figure 7.1: Example of `ApplicationSwComponentType` with code section

According the SWS RTE [7] the Runnable Entity prototype in the Application Header File of the software component is emitted as:

Example 7.9

Runnable Entity prototype in Application Header File Rte_MySwc.h according SWS_Rte_7194

```
1 #define MySwc_START_SEC_CODE
```

```
#include "MySwc_MemMap.h"

FUNC(void, MySwc_CODE) Run1 (void);

#define MySwc_STOP_SEC_CODE
#include "MySwc_MemMap.h"
```

Please note that the same Memory Allocation Keywords have to be used for the function definition of "Run1" and all other functions of the Software Component which shall be located to same MemorySection.

The SwcImplementation "Impl_MySwc" associated with the ApplicationSwComponentType "MySwc" defines that it uses a MemorySection named CODE. The MemorySection "CODE" refers to SwAddrMethod "CODE". This indicates that the module specific (abstract) memory section CODE share a common addressing strategy defined by SwAddrMethod "CODE".

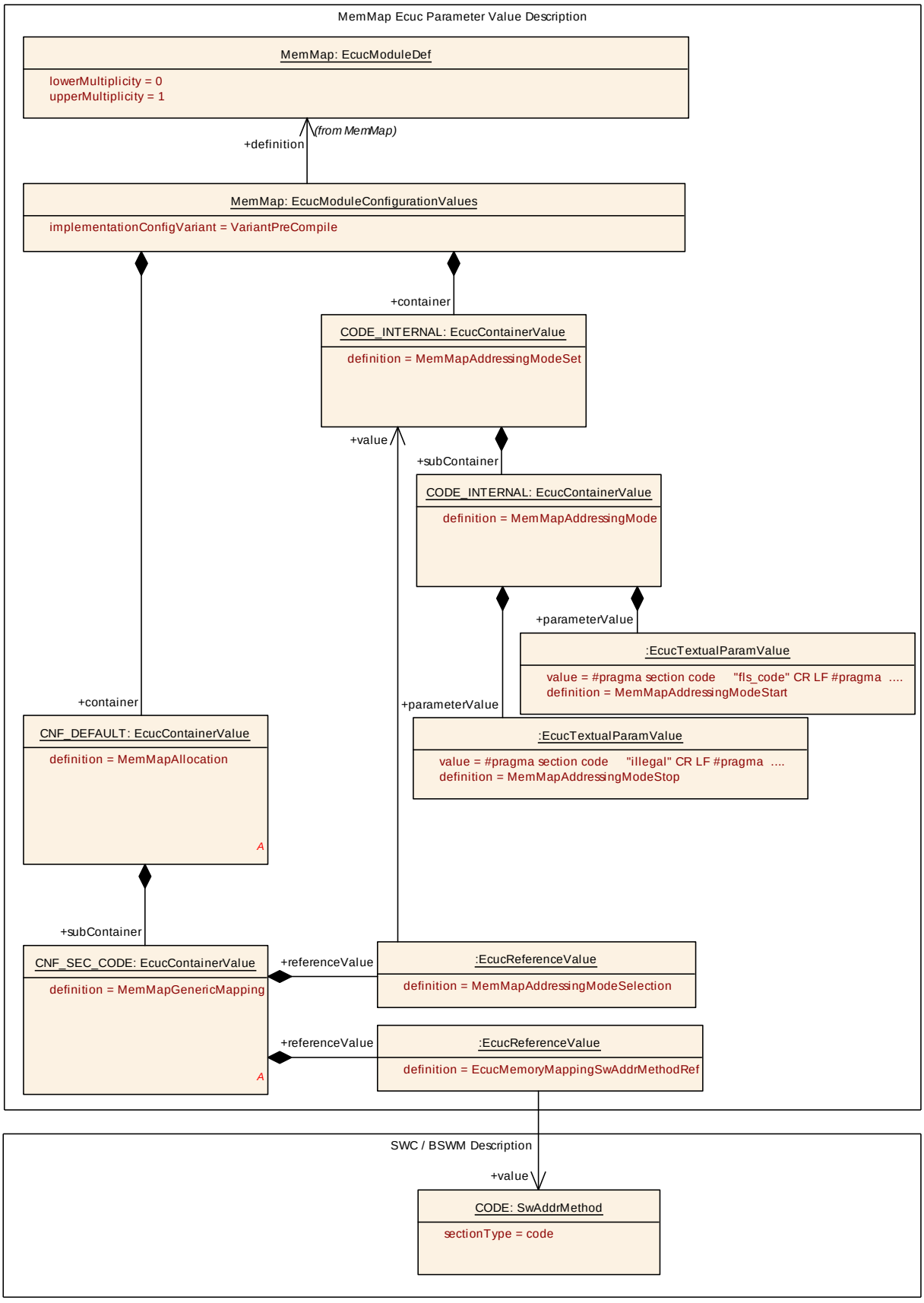

Figure 7.2: Example of `MemMap` configuration for a code section

With the means of the MemMapGenericMapping "CNF_SEC_CODE" Memory Mapping is configured that all module specific (abstract) memory sections referring to SwAddrMethod "CODE" are using the MemMapAddressingModeSet "CODE_INTERNAL". MemMapAddressingModeSet "CODE_INTERNAL" defines the proper statements to start and to stop the mapping of code to the specific linker sections by the usage of the related Memory Allocation Keywords.

With this information of the Memory Allocation Header for the Software Component can be generated like:

Example 7.10

Header file MySwc_MemMap.h according [SWS_MemMap_00022]

```
1
2 #ifdef MySwc_START_SEC_CODE
3 #pragma section_code "fls_code"
4 #pragma ...
5     #undef MySwc_START_SEC_CODE
6
7 #ifdef MySwc_STOP_SEC_CODE
8 #pragma section_code "illegal"
9     #undef MySwc_STOP_SEC_CODE
```

7.3.2 Fast Variable Section

The following example shows ApplicationSwComponentType "MySwc" which contains in its SwcInternalBehavior two VariableDataPrototypes "FooBar" and "EngSpd"'.
The VariableDataPrototype "FooBar" references a ImplementationDataType which is associated to a SwBaseType defining baseTypeSize = 8. This denotes a variable size of 8 bit for the data implementing "FooBar".
The VariableDataPrototype "EngSpd" references a ImplementationDataType which is associated to a SwBaseType defining baseTypeSize = 16. This denotes a variable size of 16 bit for the data implementing "EngSpd".

Both VariableDataPrototypes references the SwAddrMethod "VAR_FAST_INIT" which sectionType attribute is set to "var" and the memoryAllocationKeywordPolicy is set to addrMethodShortNameAndAlignment.

This denotes that the variables implementing the associated VariableDataPrototypes have to be sorted according their size into different MemorySections.

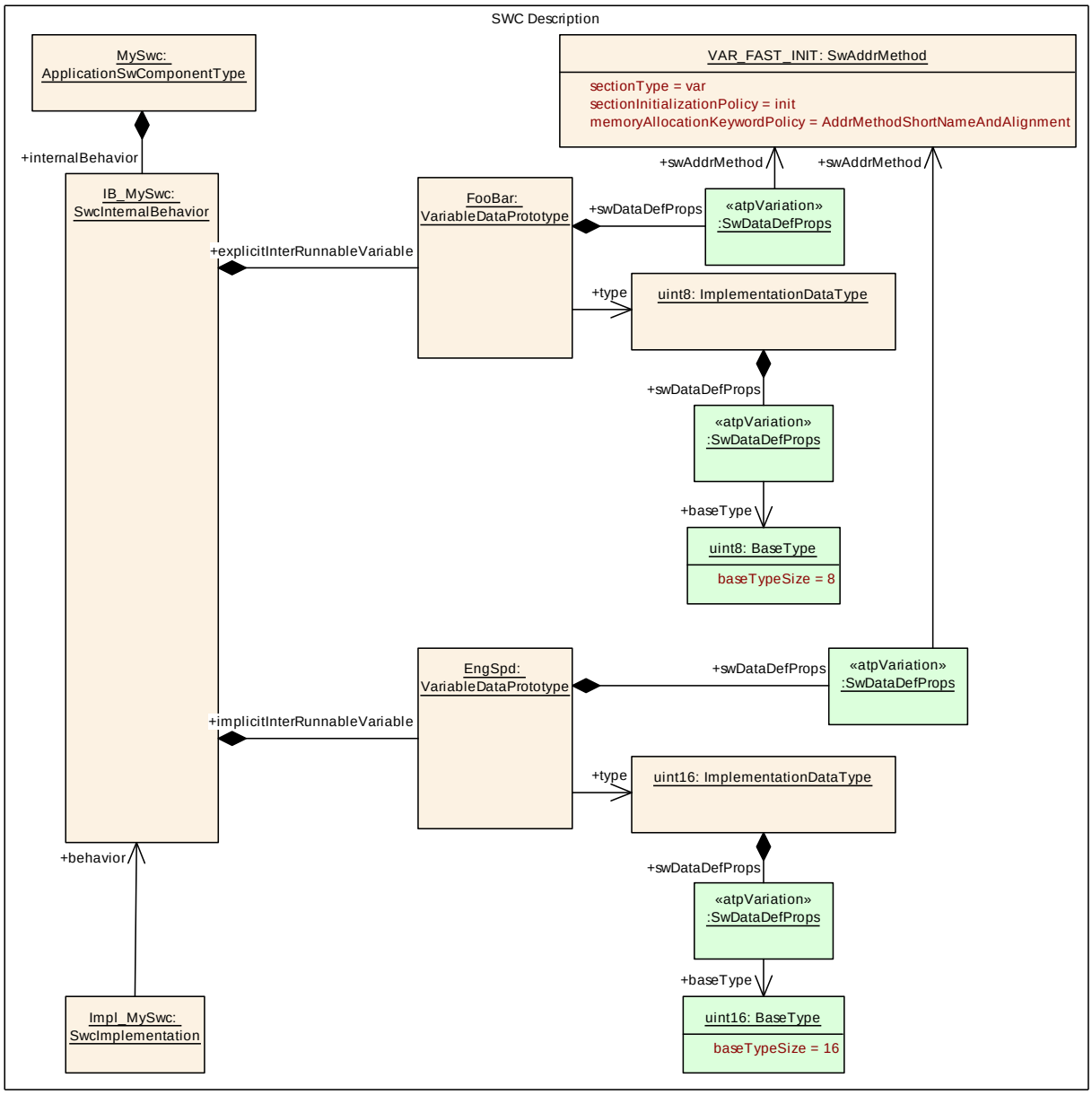

Figure 7.3: Example of `ApplicationSwComponentType` with `VariableDataPrototype`s

Please note that in this example both `VariableDataPrototype`s have to be implemented by RTE. The RTE again has to provide a BSW Module description defining the used `MemorySection`s. Further on the RTE might allocate additional buffer for instance to implement implicit communication behavior. In this example the RTE uses four different `MemorySection`s "VAR_FAST_INIT_8", "VAR_FAST_INIT_16"', "VAR_FAST_INIT_TASK_BUF_8" and "VAR_FAST_INIT_TASK_BUF_16" to sort variables according their size and to allocate additional buffers.

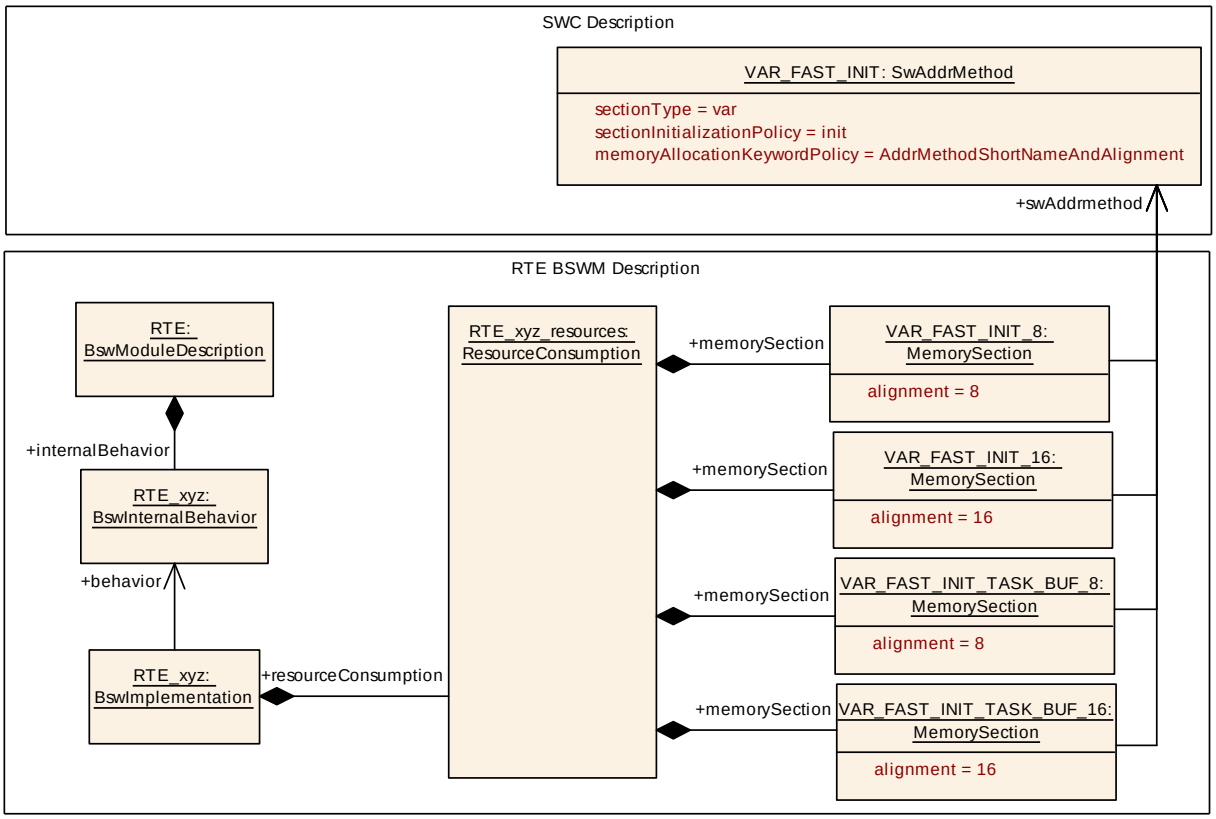

Figure 7.4: Example of Basic Software Module Description of RTE

All of these MemorySections are associated with the SwAddrMethod "VAR_FAST_INIT" This indicates that the module specific (abstract) memory sections "VAR_FAST_INIT_8", "VAR_FAST_INIT_16", "VAR_FAST_INIT_TASK_BUF_8" and "VAR_FAST_INIT_TASK_BUF_16" share a common addressing strategy defined by SwAddrMethod "VAR_FAST_INIT".

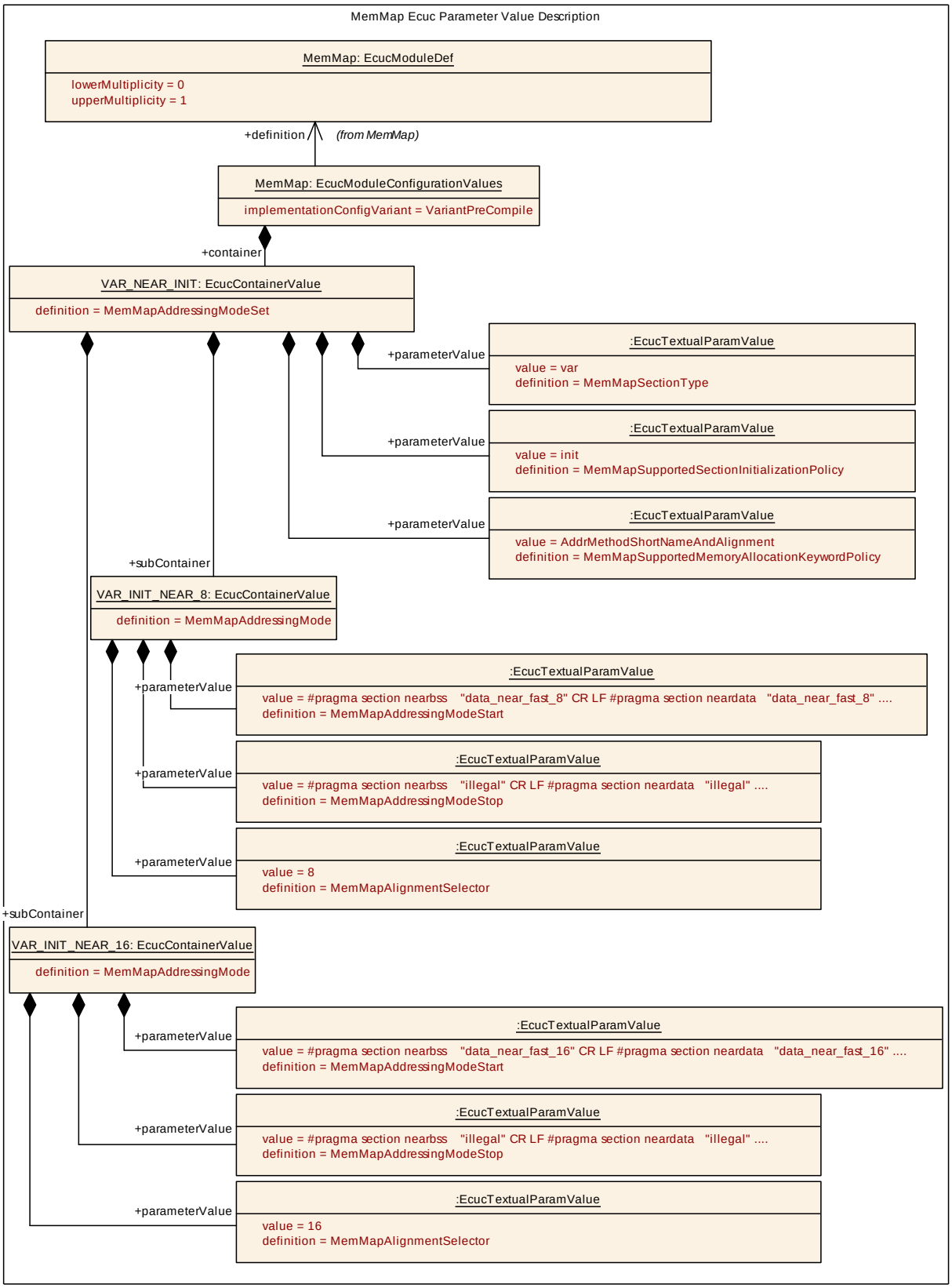

Figure 7.5: Example of `MemMap` configuration for a data section

The ECU Configuration of Memory Mapping defines a MemMapAddressingModeSet "VAR_NEAR_INIT" This supports the sectionType = var, sectionInitializationPolicy = "INIT" and memoryAllocationKeywordPolicy = addrMethodShortNameAndAlignment. In this example MemMapAddressingModes are shown for the alignment 8 and 16 (MemMapAlignmentSelector = 8 and MemMapAlignmentSelector = 16).

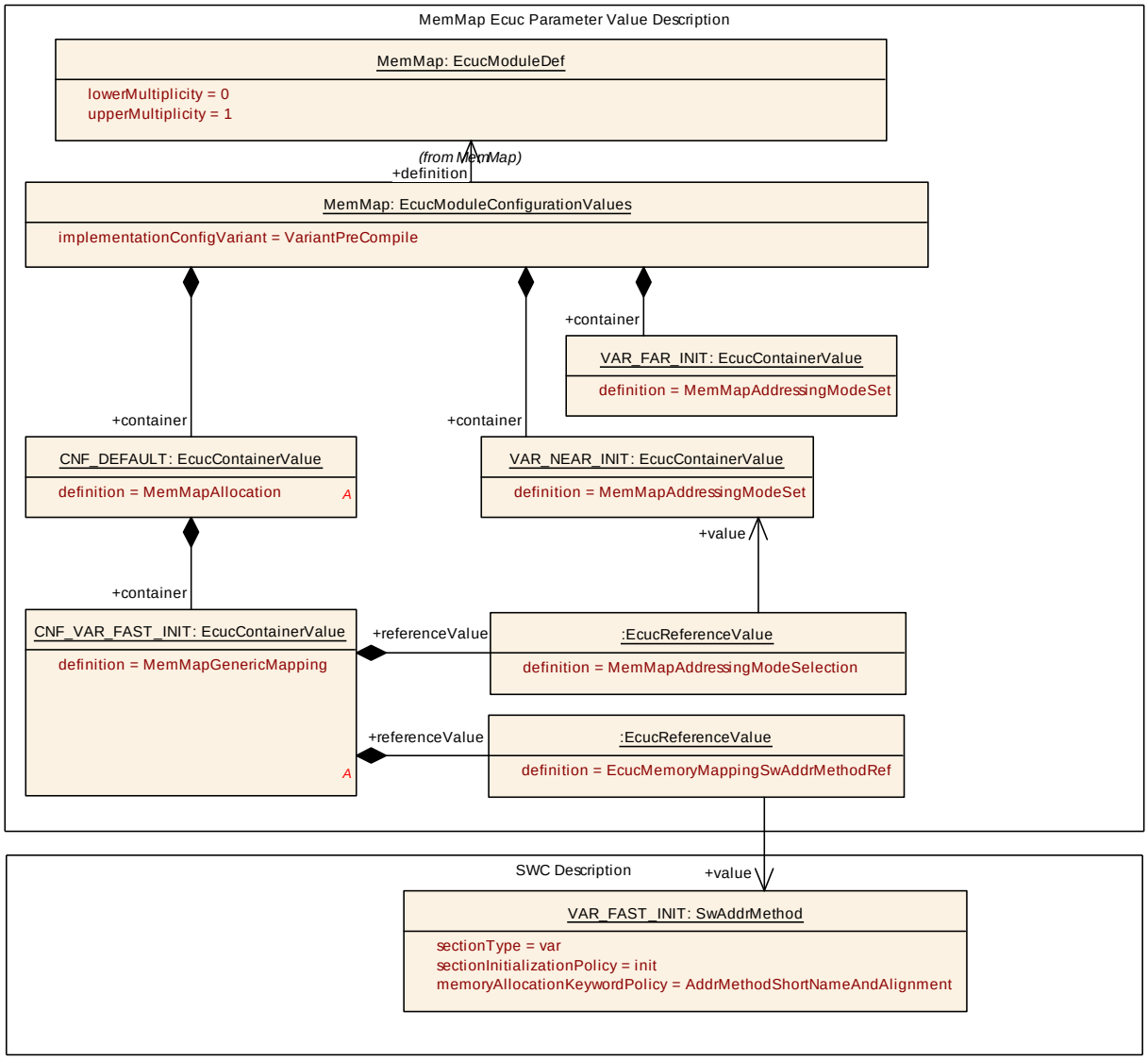

Figure 7.6: Example of MemMap configuration for a MemMapGenericMapping

With the means of the MemMapGenericMapping "CNF_VAR_FAST_INIT" Memory Mapping is configured that all module specific (abstract) memory sections referring to SwAddrMethod "VAR_FAST_INIT" are using the MemMapAddressingModeSet "VAR_NEAR_INIT". MemMapAddressingModeSet "VAR_NEAR_INIT" defines the proper statements to start and to stop the mapping of variables with different alignments (in this example 8 and 16) to the specific linker sections by the usage of the related Memory Allocation Keywords.

With this information of the Memory Allocation Header for the BSW can be generated like:

Example 7.11

MemMap Header file Rte_MemMap.h

```
#ifdef RTE_START_SEC_VAR_FAST_INIT_8
#pragma section nearbss    "data_near_fast_8"
#pragma section neardata   "data_near_fast_8"
....
#pragma ...
    #undef RTE_START_SEC_VAR_FAST_INIT_8

#ifdef RTE_STOP_SEC_VAR_FAST_INIT_8
#pragma section_code "illegal"
    #undef RTE_STOP_SEC_VAR_FAST_INIT_8

#ifdef RTE_START_SEC_VAR_FAST_INIT_16
#pragma section nearbss    "data_near_fast_16"
#pragma section neardata   "data_near_fast_16"
....
#pragma ...
    #undef RTE_START_SEC_VAR_FAST_INIT_16

#ifdef RTE_STOP_SEC_VAR_FAST_INIT_16
#pragma section_code "illegal"
    #undef RTE_STOP_SEC_VAR_FAST_INIT_16

#ifdef RTE_START_SEC_VAR_FAST_INIT_TASK_BUF_8
#pragma section nearbss    "data_near_fast_8"
#pragma section neardata   "data_near_fast_8"
....
#pragma ...
    #undef RTE_START_SEC_VAR_FAST_INIT_TASK_BUF_8

#ifdef RTE_STOP_SEC_VAR_FAST_INIT_TASK_BUF_8
#pragma section_code "illegal"
    #undef RTE_STOP_SEC_VAR_FAST_INIT_TASK_BUF_8

#ifdef RTE_START_SEC_VAR_FAST_INIT_TASK_BUF_16
#pragma section nearbss    "data_near_fast_16"
#pragma section neardata   "data_near_fast_16"
....
#pragma ...
    #undef RTE_START_SEC_VAR_FAST_INIT_TASK_BUF_16

#ifdef RTE_STOP_SEC_VAR_FAST_INIT_TASK_BUF_16
#pragma section_code "illegal"
    #undef RTE_STOP_SEC_VAR_FAST_INIT_TASK_BUF_16
```

7.3.3 Code Section in ICC2 cluster

The following Basic Software Module Description would result in the support of the Memory Allocation Keywords in the MemMap header file:

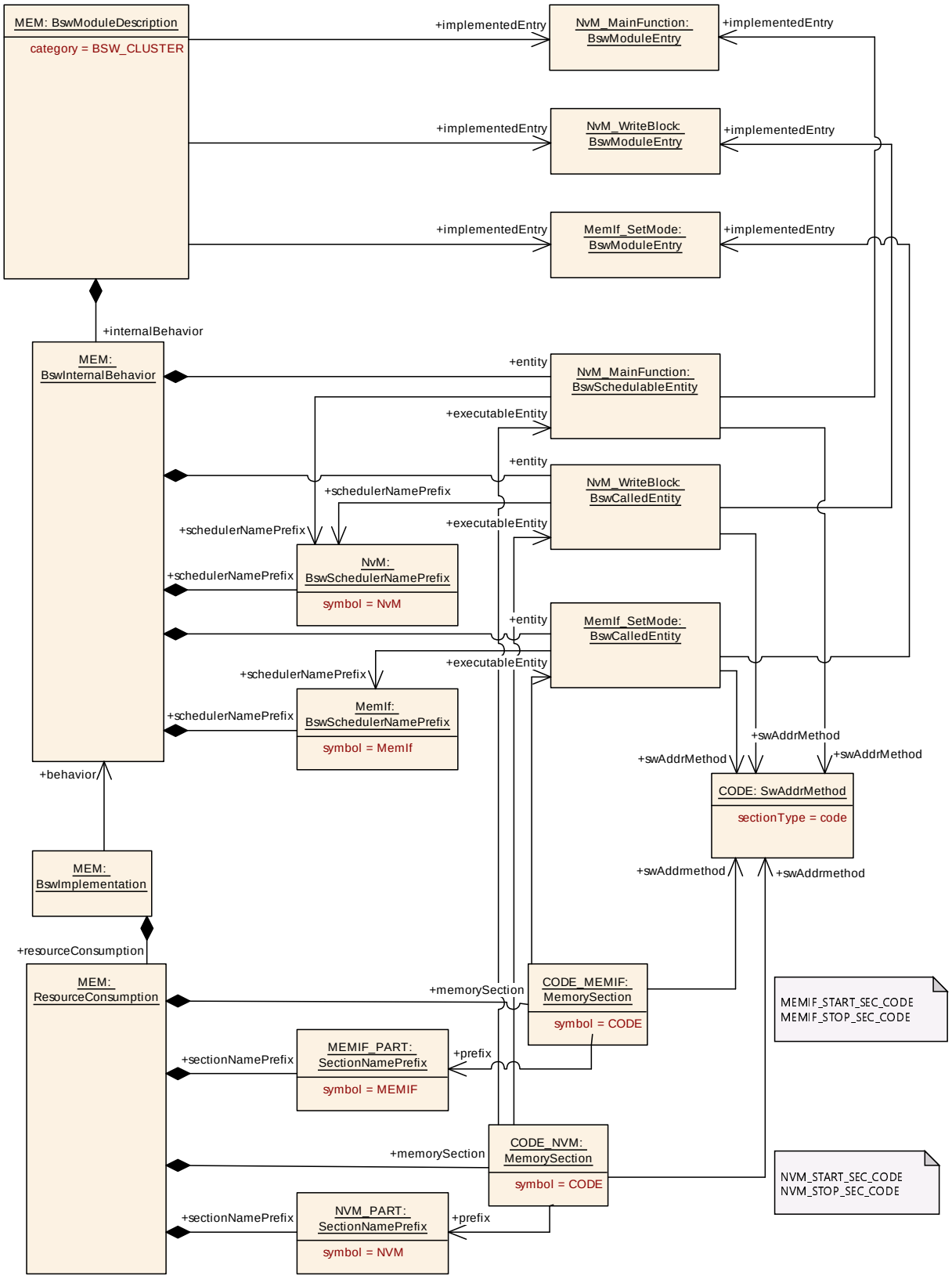

Figure 7.7: Example of BSW Module Description of an ICC2 cluster

Example 7.12

MemMap Header file

```
#ifdef NVM_START_SEC_CODE
...
#ifdef NVM_STOP_SEC_CODE
...
#ifdef MEMIF_START_SEC_CODE
...
#ifdef MEMIF_STOP_SEC_CODE
```

7.3.4 Callout sections

The following Basic Software Module Description would result in the support of the Memory Allocation Keywords in the MemMap header file:

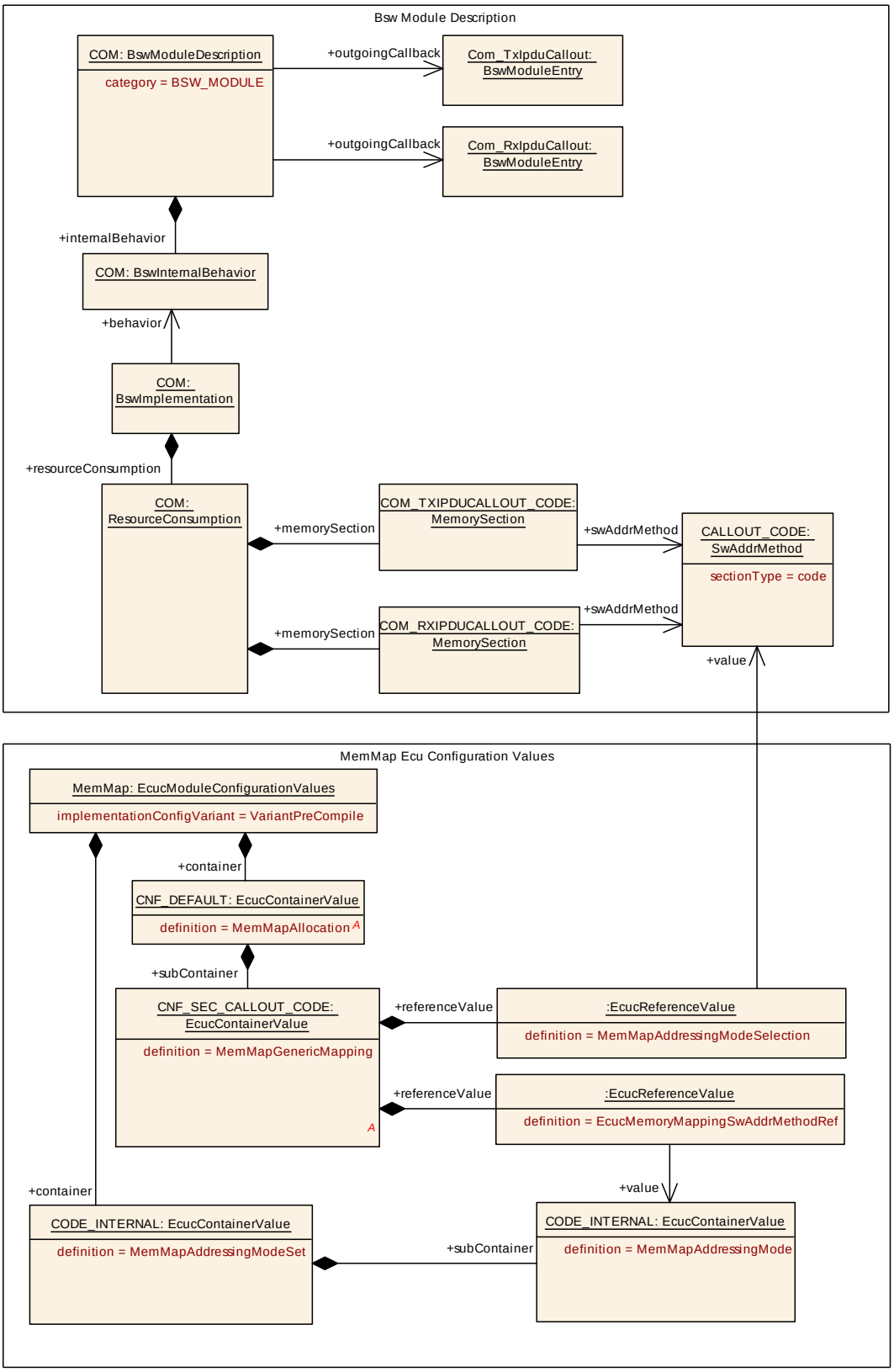

Figure 7.8: Example of description and configuration for callout code

Example 7.13

MemMap Header file

```
#ifdef COM_START_SEC_COM_TXIPDUCALLOUT_CODE
...
#ifdef COM_STOP_SEC_COM_TXIPDUCALLOUT_CODE
...
#ifdef COM_START_SEC_COM_RXIPDUCALLOUT_CODE
...
#ifdef COM_STOP_SEC_COM_RXIPDUCALLOUT_CODE
```

Nevertheless both memory sections are implemented identical since both are referencing the identical SwAddrMethod and the MemMapGenericMapping is used to configure the MemMap module.

7.3.5 Allocatable Memory Parts

The following example shows an Adc driver which is internally split into an interface part and a kernel part. Usually the kernel part is allocated to memory with high performance for the micro controller core handling the interrupts. In opposite the interface part is usually allocated to memory with a good average performance for all micro controller cores using the Adc module. The shown configuration would result in the support of following Memory Allocation Keywords in the Adc_MemMap.h header file:

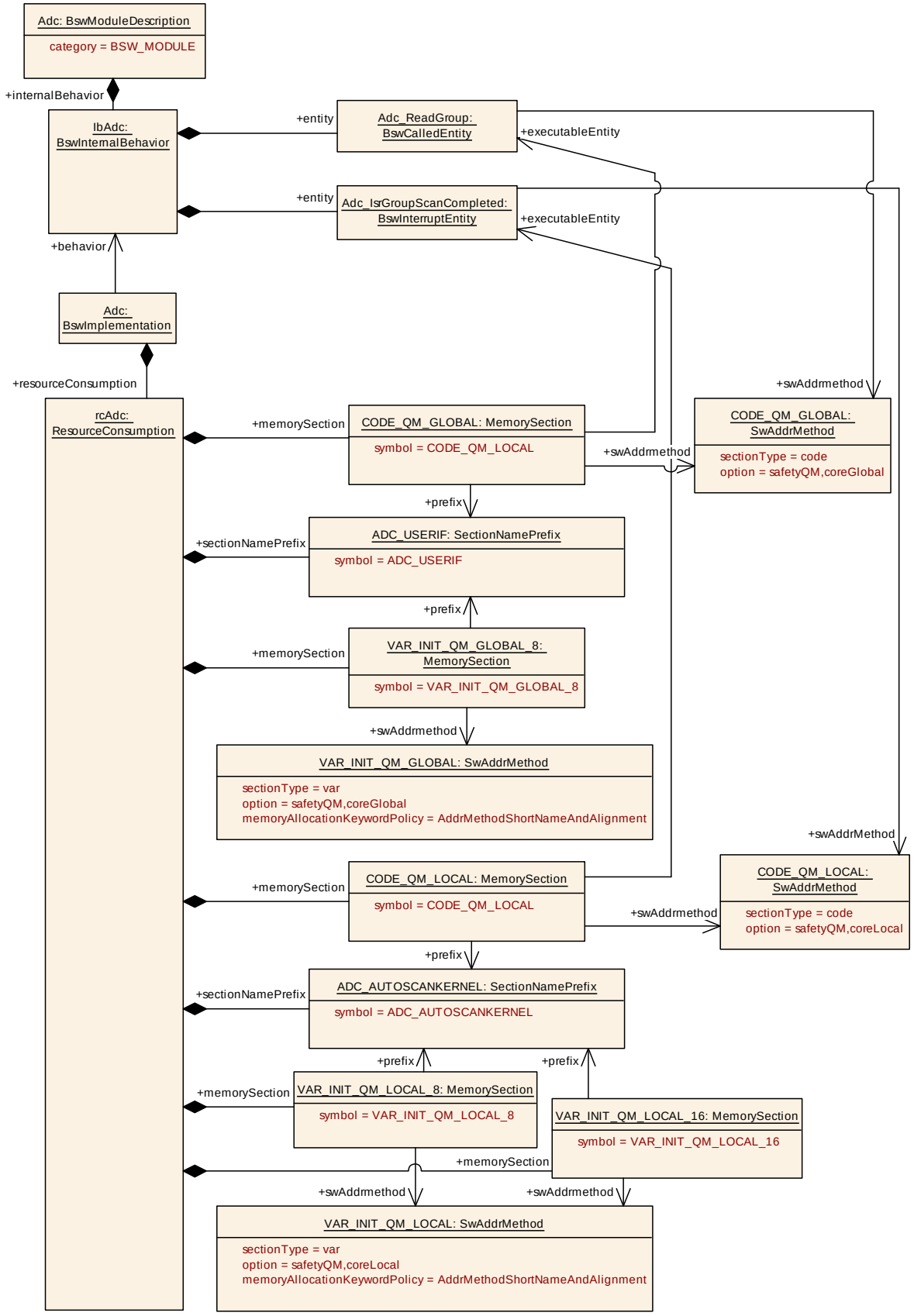

Figure 7.9: Example of description and configuration for allocatable memory parts

Example 7.14

`Adc_MemMap.h` header file

```
#ifdef ADC_USERIF_START_SEC_CODE_QM_GLOBAL
...
#ifdef ADC_USERIF_STOP_SEC_CODE_QM_GLOBAL
...
#ifdef ADC_USERIF_START_SEC_VAR_INIT_QM_GLOBAL_8
...
#ifdef ADC_USERIF_STOP_SEC_VAR_INIT_QM_GLOBAL_8
...
#ifdef ADC_AUTOSCANKERNEL_START_SEC_CODE_QM_LOCAL
...
#ifdef ADC_AUTOSCANKERNEL_STOP_SEC_CODE_QM_LOCAL
...
#ifdef ADC_AUTOSCANKERNEL_START_SEC_VAR_INIT_QM_LOCAL_8
...
#ifdef ADC_AUTOSCANKERNEL_STOP_SEC_VAR_INIT_QM_LOCAL_8
...
#ifdef ADC_AUTOSCANKERNEL_START_SEC_VAR_INIT_QM_LOCAL_16
...
#ifdef ADC_AUTOSCANKERNEL_STOP_SEC_VAR_INIT_QM_LOCAL_16
```

Nevertheless both memory sections are implemented identical since both are referencing the identical `SwAddrMethod` and the `MemMapGenericMapping` is used to configure the `MemMap` module.

8 API specification

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 section 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave section 10.1 in the specification to guarantee comprehension.

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

Chapter 10.3 specifies published information of the module `MemMap`.

10.1 How to read this chapter

For details refer to the chapter 10.1 "Introduction to configuration specification" in SWS_BSWGeneral [2].

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe chapter 7 Functional specification.

10.2.1 MemMap

Module SWS Item	ECUC_MemMap_00001	
Module Name	MemMap	
Module Description	Configuration of the Memory Mapping and Compiler Abstraction module.	
Post-Build Variant Support	false	
Supported Config Variants	VARIANT-PRE-COMPILE	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
MemMapAddressingModeSet	0..*	Defines a set of addressing modes which might apply to a SwAddrMethod.

Container Name	Multiplicity	Scope / Dependency
MemMapAllocation	0..*	Defines which MemorySection of a BSW Module or a Software Component is implemented with which MemMapAddressingModeSet. This can either be specified for a set of MemorySections which refer to an identical SwAddrMethod (MemMapGenericMapping) or for individual MemorySections (MemMapSectionSpecificMapping). If both are defined for the same MemorySection the MemMapSectionSpecificMapping overrules the MemMapGenericMapping.
MemMapGenericCompiler MemClass	0..*	The shortName of the container defines the name of the generic Compiler memclass which is global for all using modules, e.g. REGSPACE. The configures the Compiler Abstraction. **Tags:** atp.Status=obsolete

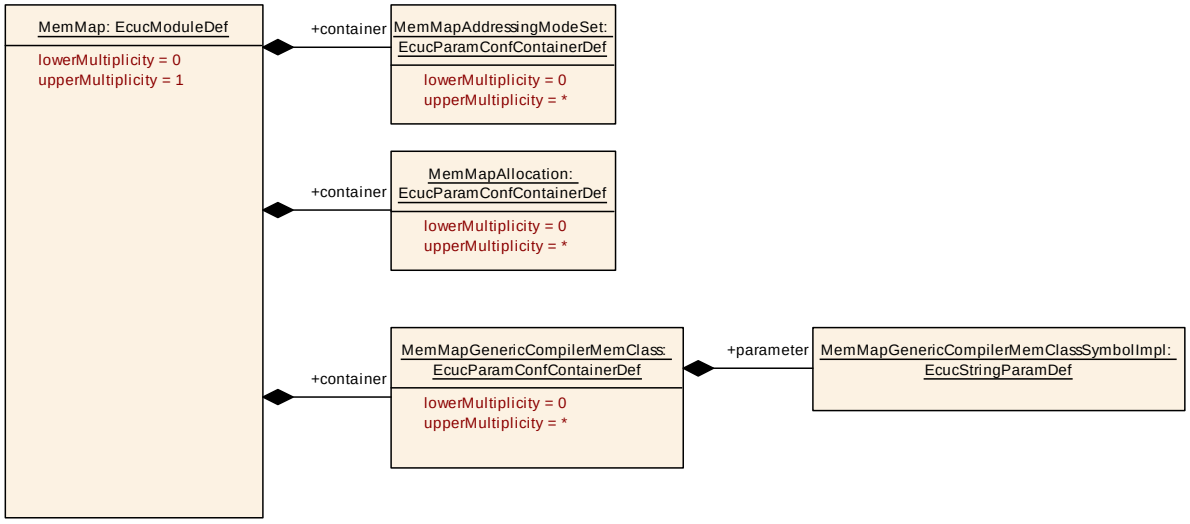

Figure 10.1: Overview about MemMap

10.2.2 MemMapAddressingModeSet

SWS Item	[ECUC_MemMap_00002]
Container Name	MemMapAddressingModeSet
Parent Container	MemMap
Description	Defines a set of addressing modes which might apply to a SwAddrMethod.
Configuration Parameters	

Name	MemMapCompilerMemClassSymbolImpl [ECUC_MemMap_00018] (Obsolete)		
Parent Container	MemMapAddressingModeSet		
Description	Defines the implementation behind a MemClassSymbol and configures the Compiler Abstraction. **Tags:** atp.Status=obsolete		
Multiplicity	1		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapSupportedAddressingMethodOption [ECUC_MemMap_00009]		
Parent Container	MemMapAddressingModeSet		
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddrMethods. The attribute option of a swAddrMethod mapped via MemMapGenericMapping to this MemMapAddressingModeSet shall be equal to one of the configured MemMapSupportedAddressMethodOption's		
Multiplicity	0..*		
Type	EcucStringParamDef		
Default Value			
Regular Expression	[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapSupportedMemoryAllocationKeywordPolicy [ECUC_MemMap_00017]		
Parent Container	MemMapAddressingModeSet		
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddrMethods. The attribute MemoryAllocationKeywordPolicy of a swAddrMethod mapped via MemMapGenericMapping to this MemMapAddressingModeSet shall be equal to one of the configured MemMapSupportedMemoryAllocationKeywordPolicy's		
Multiplicity	0..*		
Type	EcucEnumerationParamDef		
Range	MEMMAP_ALLOCATION_KEYWORD_POLICY_ADDR_METHOD_SHORT_NAME	The Memory Allocation Keyword is build with the short name of the SwAddrMethod. This is the default value if the atttribute does not exist in the SwAddrMethod.	
	MEMMAP_ALLOCATION_KEYWORD_POLICY_ADDR_METHOD_SHORT_NAME_AND_ALIGNMENT	The Memory Allocation Keyword is build with the the short name of the SwAddrMethod and the alignment attribute of the MemorySection. This requests a separation of objects in memory dependent from the alignment and is not applicable for RunnableEntitys and BswSchedulableEntitys.	
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapSupportedSectionInitializationPolicy [ECUC_MemMap_00008]		
Parent Container	MemMapAddressingModeSet		
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddrMethods. The sectionIntializationPolicy attribute value of a swAddrMethod mapped via MemMapGenericMapping to this MemMapAddressingModeSet shall be equal to one of the configured MemMapSupportedSectionIntializationPolicy's. Please note that SectionInitializationPolicyType describes the intended initialization of MemorySections. The following values are standardized in AUTOSAR Methodology (see chapter 7.2.1): • INIT • CLEARED • POWER-ON-CLEARED Note: The values NO-INIT and POWER-ON-INIT are still supported but deprecated and will be removed in one of the next releases. Note: The values are defined similar to the representation of enumeration types in the XML schema to ensure backward compatibility.		
Multiplicity	0..*		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapSupportedSectionType [ECUC_MemMap_00007]		
Parent Container	MemMapAddressingModeSet		
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddrMethods. The attribute sectionType of a swAddrMethod mapped via MemMapGenericMapping or MemMapSectionSpecificMapping to this MemMapAddressingModeSet shall be equal to one of the configured MemMapSupportedSectionType's.		
Multiplicity	0..*		
Type	EcucEnumerationParamDef		
Range	MEMMAP_SECTION_TYPE_CAL_PRM	To be used for calibratable constants of ECU-functions.	
	MEMMAP_SECTION_TYPE_CODE	To be used for mapping code to application block, boot block, external flash etc.	
	MEMMAP_SECTION_TYPE_CONFIG_DATA	Constants with attributes that show that they reside in one segment for module configuration.	
	MEMMAP_SECTION_TYPE_CONST	To be used for global or static constants.	
	MEMMAP_SECTION_TYPE_EXCLUDE_FROM_FLASH	Values existing in the ECU but not dropped down in the binary file. No upload should be needed to obtain access to the ECU data. The ECU will never be touched by the instrumentation tool, with the exception of upload. These are memory areas which are not overwritten by downloading the executable.	
	MEMMAP_SECTION_TYPE_VAR	To be used for global or static variables. The expected initialization is specified with the attribute sectionInitializationPolicy.	
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
MemMapAddressingMode	1..*	Defines a addressing mode with a set of #pragma statements implementing the start and the stop of a section.

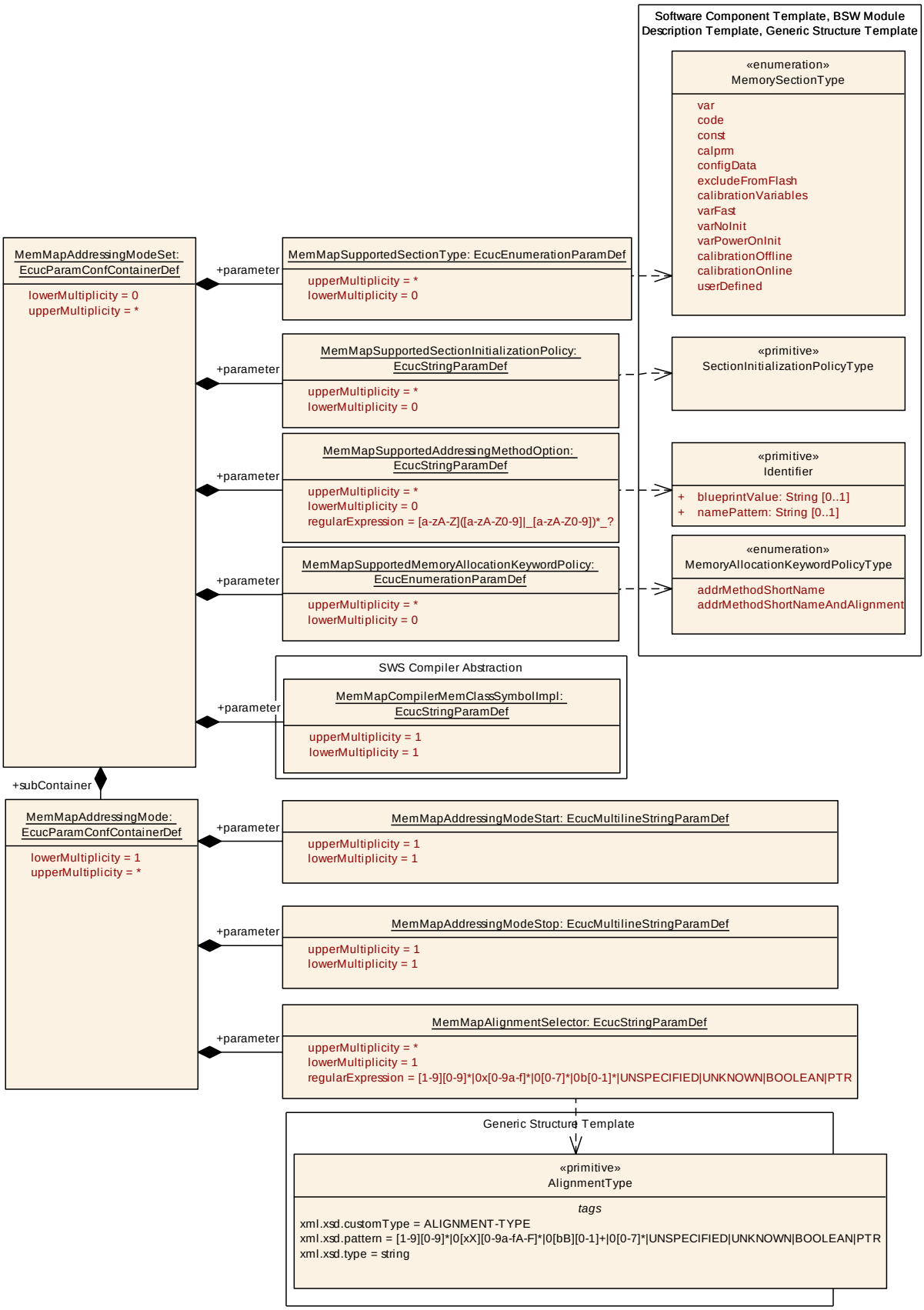

Figure 10.2: Overview about MemMapAddressingModeSet

10.2.3 MemMapAddressingMode

SWS Item	[ECUC_MemMap_00003]
Container Name	MemMapAddressingMode
Parent Container	MemMapAddressingModeSet
Description	Defines a addressing mode with a set of #pragma statements implementing the start and the stop of a section.
Configuration Parameters	

Name	MemMapAddressingModeStart [ECUC_MemMap_00004]		
Parent Container	MemMapAddressingMode		
Description	Defines a set of #pragma statements implementing the start of a section.		
Multiplicity	1		
Type	EcucMultilineStringParamDef		
Default Value			
Regular Expression			
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	MemMapAddressingModeStop [ECUC_MemMap_00005]		
Parent Container	MemMapAddressingMode		
Description	Defines a set of #pragma statements implementing the start of a section.		
Multiplicity	1		
Type	EcucMultilineStringParamDef		
Default Value			
Regular Expression			
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	MemMapAlignmentSelector [ECUC_MemMap_00006]		
Parent Container	MemMapAddressingMode		
Description	Defines a the alignments for which the MemMapAddressingMode applies. The to be used alignment is defined in the alignment attribute of the MemorySection. If the MemMapAlignmentSelector fits to alignment attribute of the MemorySection the set of #pragmas of the related MemMapAddressingMode shall be used to implement the start and the stop of a section. Please note that the same MemMapAddressingMode can be applicable for several alignments, e.g. "8" bit and "UNSPECIFIED".		
Multiplicity	1..*		
Type	EcucStringParamDef		
Default Value			
Regular Expression	[1-9][0-9]*|0x[0-9a-f]*|0[0-7]*|0b[0-1]*|UNSPECIFIED|UNKNOWN|BOOLEAN|PTR		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.4 MemMapAllocation

SWS Item	[ECUC_MemMap_00010]
Container Name	MemMapAllocation
Parent Container	MemMap
Description	Defines which MemorySection of a BSW Module or a Software Component is implemented with which MemMapAddressingModeSet. This can either be specified for a set of MemorySections which refer to an identical SwAddrMethod (MemMapGenericMapping) or for individual MemorySections (MemMapSectionSpecificMapping). If both are defined for the same MemorySection the MemMapSectionSpecificMapping overrules the MemMapGenericMapping.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
MemMapGeneric Mapping	0..*	Defines which SwAddrMethod is implemented with which MemMapAddressingModeSet. The pragmas for the implementation of the MemorySelectorKeywords are taken from the MemMapAddressingModeStart and MemMapAddressingModeStop parameters of the MemMapAddressingModeSet for the individual alignments. That this mapping becomes valid requires matching MemMapSupportedSectionType's, MemMapSupportedSectionInitializationPolicy's and MemMapSupportedAddressingMethodOption's. The MemMapGenericMapping applies only if it is not overruled by an MemMapSectionSpecificMapping
MemMapMapping Selector	0..*	The container holds a section criteria reusable for MemMapGenericMappings.
MemMapSectionSpecific Mapping	0..*	Defines which MemorySection of a BSW Module or a Software Component is implemented with which MemMapAddressingModeSet. The pragmas for the implementation of the MemorySelectorKeywords are taken from the MemMapAddressingModeStart and MemMapAddressingModeStop parameters of the MemMapAddressingModeSet for the specific alignment of the MemorySection. The MemMapSectionSpecificMapping precedes a mapping defined by MemMapGenericMapping.

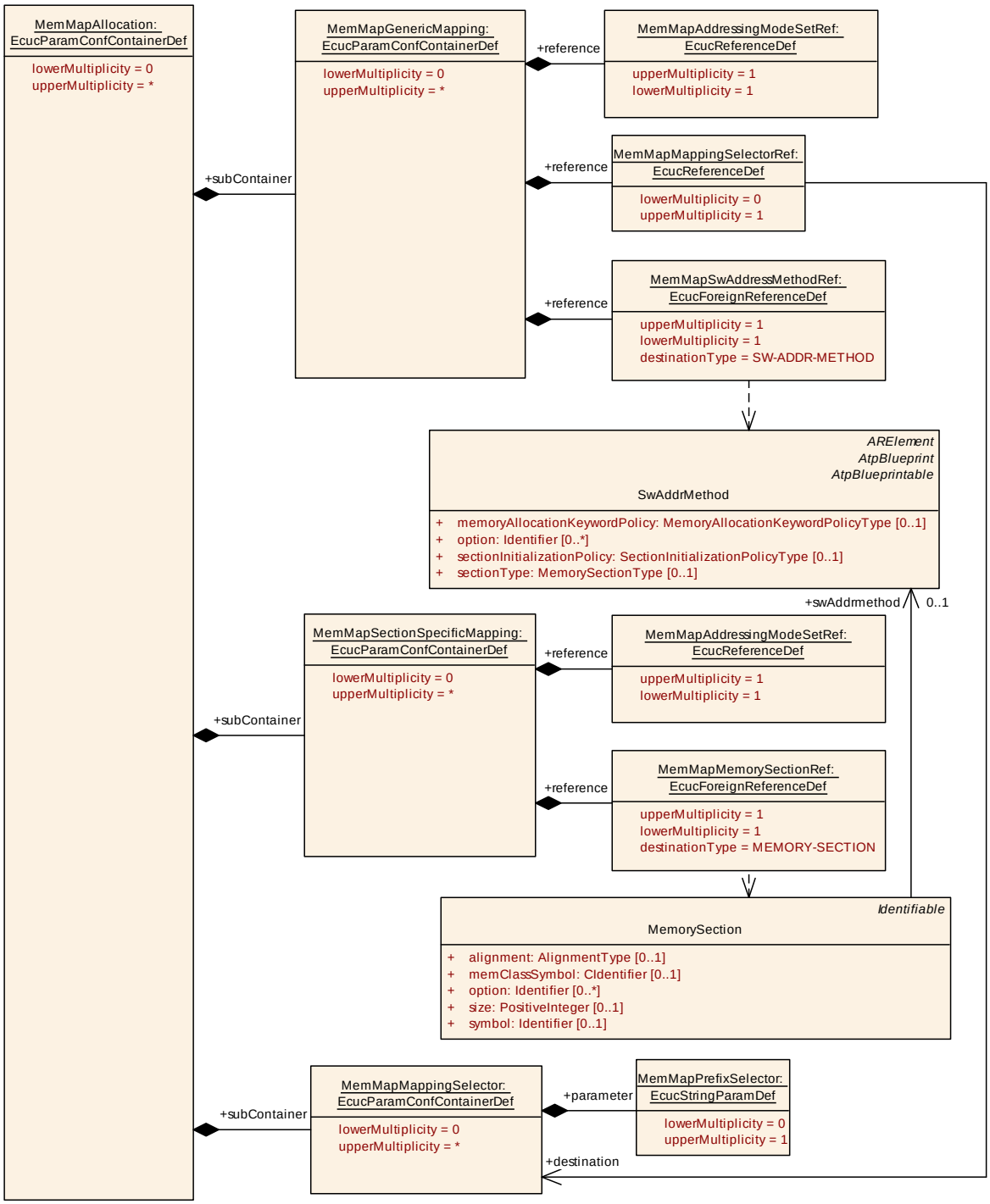

Figure 10.3: Overview about MemMapAllocation

10.2.5 MemMapGenericMapping

SWS Item	[ECUC_MemMap_00011]
Container Name	MemMapGenericMapping
Parent Container	MemMapAllocation

Description	Defines which SwAddrMethod is implemented with which MemMapAddressingModeSet. The pragmas for the implementation of the MemorySelectorKeywords are taken from the MemMapAddressingModeStart and MemMapAddressingModeStop parameters of the MemMapAddressingModeSet for the individual alignments. That this mapping becomes valid requires matching MemMapSupportedSectionType's, MemMapSupportedSectionInitializationPolicy's and MemMapSupportedAddressingMethodOption's. The MemMapGenericMapping applies only if it is not overruled by an MemMapSectionSpecificMapping
Configuration Parameters	

Name	MemMapAddressingModeSetRef [ECUC_MemMap_00012]		
Parent Container	MemMapGenericMapping		
Description	Reference to the MemMapAddressingModeSet which applies to the MemMapGenericMapping.		
Multiplicity	1		
Type	Reference to MemMapAddressingModeSet		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapMappingSelectorRef [ECUC_MemMap_00023]		
Parent Container	MemMapGenericMapping		
Description	Reference to a MemMapPrefixSelector. The owning MemMapGenericMapping is only effective for those memories where the MemMapMappingSelector matches.		
Multiplicity	0..1		
Type	Reference to MemMapMappingSelector		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapSwAddressMethodRef [ECUC_MemMap_00013]		
Parent Container	MemMapGenericMapping		
Description	Reference to the SwAddrMethod which applies to the MemMapGenericMapping.		
Multiplicity	1		
Type	Foreign reference to SW-ADDR-METHOD		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.6 MemMapSectionSpecificMapping

SWS Item	[ECUC_MemMap_00014]
Container Name	MemMapSectionSpecificMapping
Parent Container	MemMapAllocation
Description	Defines which MemorySection of a BSW Module or a Software Component is implemented with which MemMapAddressingModeSet. The pragmas for the implementation of the MemorySelectorKeywords are taken from the MemMapAddressingModeStart and MemMapAddressingModeStop parameters of the MemMapAddressingModeSet for the specific alignment of the MemorySection. The MemMapSectionSpecificMapping precedes a mapping defined by MemMapGenericMapping.
Configuration Parameters	

Name	MemMapAddressingModeSetRef [ECUC_MemMap_00015]		
Parent Container	MemMapSectionSpecificMapping		
Description	Reference to the MemMapAddressingModeSet which applies to the MemMapModuleSectionSpecificMapping.		
Multiplicity	1		
Type	Reference to MemMapAddressingModeSet		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	MemMapMemorySectionRef [ECUC_MemMap_00016]		
Parent Container	MemMapSectionSpecificMapping		
Description	Reference to the MemorySection which applies to the MemMapSectionSpecificMapping.		
Multiplicity	1		
Type	Foreign reference to MEMORY-SECTION		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.7 MemMapMappingSelector

SWS Item	[ECUC_MemMap_00021]
Container Name	MemMapMappingSelector
Parent Container	MemMapAllocation
Description	The container holds a section criteria reusable for MemMapGenericMappings.
Configuration Parameters	

Name	MemMapPrefixSelector [ECUC_MemMap_00022]		
Parent Container	MemMapMappingSelector		
Description	The parameter MemMapPrefixSelector defines a regular expression which shall be applied to the <PREFIX> part of the memory allocation keywords. The mapping using this selector is only effective for those memories where the <PREFIX> part of the memory allocation keyword matches the regular expression. Note: This is in particular intended the restrict the usage of of a MemMapAddressingModeSet for a sub set of BSW Modules or Software Components or a subset of allocatable memory parts inside BSW Modules or Software Components.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.8 MemMapGenericCompilerMemClass

SWS Item	[ECUC_MemMap_00019] (Obsolete)
Container Name	MemMapGenericCompilerMemClass
Parent Container	MemMap
Description	The shortName of the container defines the name of the generic Compiler memclass which is global for all using modules, e.g. REGSPACE. The configures the Compiler Abstraction. **Tags:** atp.Status=obsolete
Configuration Parameters	

Name	MemMapGenericCompilerMemClassSymbolImpl [ECUC_MemMap_00020] (Obsolete)		
Parent Container	MemMapGenericCompilerMemClass		
Description	Defines the implementation behind the generic MemClassSymbol and configures the Compiler Abstraction. **Tags:** atp.Status=obsolete		
Multiplicity	1		
Type	EcucStringParamDef		
Default Value			
Regular Expression			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.3 Published Information

For details refer to the chapter 10.3 Published Information in SWS_BSWGeneral [2].

11 Analysis

This chapter does not contain requirements. It just gives an overview to used keywords and their syntax within different compilers. This analysis is required for a correct and complete specification of methods and keywords and is based on the documents [8], [9], [10], [11] and [12].

11.1 Memory allocation of variables

Compiler analysis for starting/stopping a memory section for variables:

Compiler	Required syntax

Cosmic, S12X	Initialized variables:
	`#pragma section {name}`
	`#pragma section {}`
	
	Non Initialized variables:
	`#pragma section {[name]}`
	`#pragma section []`

Metrowerks, S12X	`#pragma DATA_SEG (<Modif> <Name>|"DEFAULT")`
	<Modif>: Some of the following strings may be used:
	`SHORT, __SHORT_SEG,`
	`DIRECT, __DIRECT_SEG,`
	`NEAR, __NEAR_SEG,`
	`FAR, __FAR_SEG,`
	`DPAGE, __DPAGE_SEG,`
	`RPAGE, __RPAGE_SEG`
	Pragma shall be used in definition and declaration.

Tasking, ST10	`#pragma class mem=name`
	`#pragma combine mem=ctype`
	`#pragma align mem=atype`
	`#pragma noclear`
	`#pragma default_attributes`
	`#pragma clear`
	
	atype is one of the following align types:
	B Byte alignment
	W Word alignment
	P Page alignment
	S Segment alignment
	C PEC addressable
	I IRAM addressable
	
	ctype is one of the following combine types:
	L private ('Local')
	P Public
	C Common
	G Global
	S Sysstack

Compiler	Required syntax
	U Usrstack A address Absolute section AT constant address (decimal, octal or hexadecimal number)

Tasking, TC1796	`#pragma pack 0 / 2`
	Packing of structs. Shall be visible at type declaration
	`#pragma section type "string"`
	`#pragma noclear`
	`#pragma clear`
	`#pragma for_extern_data_use_memory`
	`#pragma for_initialized_data_use_memory`
	`#pragma for_uninitialized_data_use_memory`

GreenHills, V850	`#pragma align (n)`
	`#pragma alignvar (n)`
	`#pragma ghs section sect="name"`
	`#pragma ghs section sect =default`
	Section Keyword:
	`data, sdata, tdata, zdata, bss, sbss, zbss`

ADS, ST30	`#pragma arm section [sort_type[[=]"name"]]`
	`[,sort_type="name"]*`
	`sort_type="rwdata, zidata`
	Alignment control via key words:
	`__packed, __align()`

DIABDATA, MPC5554	`#pragma section class_name [init_name] [uninit_name]`
	`[address_mode] [access]`
	`#pragma section class_name`
	Pragma shall be used before declaration.
	`class_name` for variables:
	`BSS, DATA, SDATA`

Table 11.1: Memory allocation of variables

11.2 Memory allocation of constant variables

Compiler analysis for starting/stopping a memory section for constant variables:

Compiler	Required syntax

Cosmic, S12X	Initialized variables:
	`#pragma section const {name}`
	`#pragma section const {}`

Metrowerks, S12X	`#pragma CONST_SEG (<Modif> <Name>|"DEFAULT")`
	<Modif>: Some of the following strings may be used:
	`PPAGE, __PPAGE_SEG,`
	`GPAGE, __GPAGE_SEG,`
	Pragma shall be used in definition and declaration.

Compiler	Required syntax

Tasking, ST10	`#pragma class mem=name`
	`#pragma align mem=atype`
	`#pragma combine mem=ctype`
	`#pragma default_attributes`
	
	atype is one of the following align types:
	B Byte alignment
	W Word alignment
	P Page alignment
	S Segment alignment
	C PEC addressable
	I IRAM addressable
	
	ctype is one of the following combine types:
	L private ('Local')
	P Public
	C Common
	G Global
	S Sysstack
	U Usrstack
	A address Absolute section AT constant address
	(decimal, octal or hexadecimal number)

Tasking, TC1796	`#pragma pack 0 / 2`
	Packing of structs. Shall be visible at type declaration
	
	`#pragma section type  "string"`
	`#pragma for_constant_data_use_memory`

GreenHills, V850	`#pragma ghs section sect="name"`
	`#pragma ghs section sect =default`
	Section Keyword:
	`rodata, rozdata, rosdata`

ADS, ST30	`#pragma arm section [sort_type[[=]"name"]]`
	`[,sort_type="name"]*`
	`sort_type="rodata`
	
	Alignment control via key words:
	`__packed, __align()`

DIABDATA, MPC5554	`#pragma section class_name [init_name]`
	`[uninit_name] [address_mode] [access]`
	`#pragma section class_name`
	Pragma shall be used before declaration.
	
	`class_name` for constant variables:
	`CONST, SCONST, STRING`

Table 11.2: Memory allocation of constant variables

11.3 Memory allocation of code

Compiler analysis for starting/stopping a memory section for code:

Compiler	Required syntax

Cosmic, S12X	Initialized variables:
	`#pragma section ( name )`
	`#pragma section ()`

Metrowerks, S12X	`#pragma CODE_SEG (<Modif> <Name> | "DEFAULT")`
	<Modif>: Some of the following strings may be used:
	`DIRECT, __DIRECT_SEG,`
	`NEAR, __NEAR_SEG,`
	`CODE, __CODE_SEG,`
	`FAR, __FAR_SEG,`
	`PPAGE, __PPAGE_SEG,`
	`PIC, __PIC_SEG,`
	Pragma shall be used in definition and declaration.

Tasking, ST10	`#pragma class mem=name`
	`#pragma combine mem=ctype`
	`#pragma default_attributes`
	
	ctype is one of the following combine types:
	L private ('Local')
	P Public
	C Common
	G Global
	S Sysstack
	U Usrstack
	A address Absolute section AT constant address

Tasking, TC1796	`#pragma section code "string"`
	`#pragma section code_init`
	`#pragma section const_init`
	`#pragma section vector_init`
	`#pragma section data_overlay`
	`#pragma section type[=]"name"`
	`#pragma section all`

GreenHills, V850	`#pragma ghs section sect="name"`
	`#pragma ghs section sect =default`
	Section Keyword: `text`

ADS, ST30	`#pragma arm section [sort_type[[=]"name"]]`
	`[,sort_type="name"]*`
	
	`sort_type="code`

DIABDATA, MPC5554	`#pragma section class_name [init_name]`
	`[uninit_name] [address_mode] [access]`
	`#pragma section class_name`
	Pragma shall be used before declaration.
	
	`class_name` for code:
	`CODE`

Table 11.3: Memory allocation of code

A Referenced Meta Classes

Class	**ApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ApplicationSwComponentType is used to represent the application software. **Tags:**atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: ApplicationSwComponentType

Class	**BaseTypeDirectDefinition**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	*ARObject*, *BaseTypeDefinition*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
baseType Encoding	BaseTypeEncoding String	0..1	attr	This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **Tags:**xml.sequenceOffset=90
baseTypeSize	PositiveInteger	0..1	attr	Describes the length of the data type specified in the container in bits. **Tags:**xml.sequenceOffset=70
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:**xml.sequenceOffset=110
memAlignment	PositiveInteger	0..1	attr	This attribute describes the alignment of the memory object in bits. E.g. "8" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. If the value is set to "0" the meaning shall be interpreted as "unspecified". **Tags:**xml.sequenceOffset=100
native Declaration	NativeDeclarationString	0..1	attr	This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example BaseType with shortName: "MyUnsignedInt" native Declaration: "unsigned short" Results in typedef unsigned short MyUnsignedInt; If the attribute is not defined the referring Implementation DataTypes will not be generated as a typedef by RTE. If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseType Size.

△

Class	BaseTypeDirectDefinition			
				△ This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems. **Tags:**xml.sequenceOffset=120

Table A.2: BaseTypeDirectDefinition

Class	BswImplementation			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Contains the implementation specific information in addition to the generic specification (BswModule Description and BswBehavior). It is possible to have several different BswImplementations referring to the same BswBehavior. **Tags:**atp.recommendedPackage=BswImplementations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
arRelease Version	RevisionLabelString	1	attr	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR.
behavior	BswInternalBehavior	1	ref	The behavior of this implementation. This relation is made as an association because • it follows the pattern of the SWCT • since ARElement cannot be splitted, but we want supply the implementation later, the Bsw Implementation is not aggregated in BswBehavior
preconfigured Configuration	EcucModule ConfigurationValues	*	ref	Reference to the set of preconfigured (i.e. fixed) configuration values for this BswImplementation. If the BswImplementation represents a cluster of several modules, more than one EcucModuleConfigurationValues element can be referred (at most one per module), otherwise at most one such element can be referred. **Tags:**xml.roleWrapperElement=true
recommended Configuration	EcucModule ConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.

▽

△

Class	**BswImplementation**			
vendorApiInfix	Identifier	0..1	attr	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <Module Name>_<vendorId>_ <vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1. See also SWS_BSW_00102.
vendorSpecific ModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module • several EcucModuleDefs used in this Bsw Implementation if it represents a cluster of modules • one or no EcucModuleDefs used in this Bsw Implementation if it represents a library **Tags:**xml.roleWrapperElement=true

Table A.3: BswImplementation

Class	**BswModuleDescription**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswOverview			
Note	Root element for the description of a single BSW module or BSW cluster. In case it describes a BSW module, the short name of this element equals the name of the BSW module. **Tags:**atp.recommendedPackage=BswModuleDescriptions			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bswModule Dependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDependency.shortName, bswModuleDependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
bswModule Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDocumentation, bswModuleDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=6

▽

△

Class	**BswModuleDescription**			
expectedEntry	BswModuleEntry	*	ref	Indicates an entry which is required by this module. Replacement of outgoingCallback / requiredEntry. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=expectedEntry.bswModuleEntry, expected Entry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
implemented Entry	BswModuleEntry	*	ref	Specifies an entry provided by this module which can be called by other modules. This includes "main" functions, interrupt routines, and callbacks. Replacement of providedEntry / expectedCallback. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implementedEntry.bswModuleEntry, implementedEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internalBehavior	BswInternalBehavior	*	aggr	The various BswInternalBehaviors associated with a Bsw ModuleDescription can be distributed over several physical files. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=internalBehavior.shortName xml.sequenceOffset=65
moduleId	PositiveInteger	0..1	attr	Refers to the BSW Module Identifier defined by the AUTOSAR standard. For non-standardized modules, a proprietary identifier can be optionally chosen. **Tags:**xml.sequenceOffset=5
providedClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module provides a client server entry which can be called from another parition or core.This entry is declared locally to this context and will be connected to the requiredClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedClientServerEntry.shortName, providedClientServerEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=45
providedData	VariableDataPrototype	*	aggr	Specifies a data prototype provided by this module in order to be read from another partition or core.The providedData is declared locally to this context and will be connected to the requiredData of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedData.shortName, provided Data.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=55

▽

△

Class	**BswModuleDescription**			
providedMode Group	ModeDeclarationGroup Prototype	*	aggr	A set of modes which is owned and provided by this module or cluster. It can be connected to the required ModeGroups of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with modes provided via ports by an associated ServiceSwComponentType, EcuAbstraction SwComponentType or ComplexDeviceDriverSw ComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=providedModeGroup.shortName, provided ModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
releasedTrigger	Trigger	*	aggr	A Trigger released by this module or cluster. It can be connected to the requiredTriggers of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with Triggers provided via ports by an associated ServiceSwComponentType, Ecu AbstractionSwComponentType or ComplexDeviceDriver SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=releasedTrigger.shortName, released Trigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=35
requiredClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module requires a client server entry which can be implemented on another parition or core.This entry is declared locally to this context and will be connected to the providedClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredClientServerEntry.shortName, requiredClientServerEntry.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
requiredData	VariableDataPrototype	*	aggr	Specifies a data prototype required by this module in oder to be provided from another partition or core.The required Data is declared locally to this context and will be connected to the providedData of another or the same module via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredData.shortName, required Data.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60

▽

△

Class	**BswModuleDescription**			
requiredMode Group	ModeDeclarationGroup Prototype	*	aggr	Specifies that this module or cluster depends on a certain mode group. The requiredModeGroup is local to this context and will be connected to the providedModeGroup of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredModeGroup.shortName, requiredModeGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
requiredTrigger	Trigger	*	aggr	Specifies that this module or cluster reacts upon an external trigger.This requiredTrigger is declared locally to this context and will be connected to the providedTrigger of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=requiredTrigger.shortName, requiredTrigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Table A.4: BswModuleDescription

Class	**DependencyOnArtifact**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Dependency on the existence of another artifact, e.g. a library.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
artifact Descriptor	AutosarEngineering Object	0..1	aggr	The specified artifact needs to exist.
usage	DependencyUsage Enum	1..*	attr	Specification for which process step(s) this dependency is required.

Table A.5: DependencyOnArtifact

Class	**EcucModuleConfigurationValues**
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate
Note	Head of the configuration of one Module. A Module can be a BSW module as well as the RTE and ECU Infrastructure. As part of the BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:**atp.recommendedPackage=EcucModuleConfigurationValuess

▽

△

Class	EcucModuleConfigurationValues			
Base	*AREelement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
container	EcucContainerValue	*	aggr	Aggregates all containers that belong to this module configuration. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=container.shortName, container.definition, container.variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10
definition	EcucModuleDef	0..1	ref	Reference to the definition of this EcucModule ConfigurationValues element. Typically, this is a vendor specific module configuration. **Stereotypes:** atpIdentityContributor **Tags:**xml.sequenceOffset=-10
ecucDefEdition	RevisionLabelString	0..1	attr	This is the version info of the ModuleDef ECUC Parameter definition to which this values conform to / are based on. For the Definition of ModuleDef ECUC Parameters the AdminData shall be used to express the semantic changes. The compatibility rules between the definition and value revision labels is up to the module's vendor.
implementation ConfigVariant	EcucConfiguration VariantEnum	0..1	attr	Specifies the kind of deliverable this EcucModule ConfigurationValues element provides. If this element is not used in a particular role (e.g. preconfigured Configuration or recommendedConfiguration) then the value shall be one of VariantPreCompile, VariantLink Time, VariantPostBuild.
module Description	BswImplementation	0..1	ref	Referencing the BSW module description, which this EcucModuleConfigurationValues element is configuring. This is optional because the EcucModuleConfiguration Values element is also used to configure the ECU infrastructure (memory map) or Application SW-Cs. However in case the EcucModuleConfigurationValues are used to configure the module, the reference is mandatory in order to fetch module specific "common" published information.
postBuildVariant Used	Boolean	0..1	attr	Indicates whether a module implementation has or plans to have (i.e., introduced at link or post-build time) new post-build variation points. TRUE means yes, FALSE means no. If the attribute is not defined, FALSE semantics shall be assumed.

Table A.6: EcucModuleConfigurationValues

Class	EcucValueCollection			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	This represents the anchor point of the ECU configuration description. **Tags:**atp.recommendedPackage=EcucValueCollections			
Base	*AREelement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**EcucValueCollection**			
ecucValue	EcucModule ConfigurationValues	*	ref	References to the configuration of individual software modules that are present on this ECU. atpVariation: [RS_ECUC_00079] **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
ecuExtract	System	0..1	ref	Represents the extract of the System Configuration that is relevant for the ECU configured with that ECU Configuration Description.

Table A.7: EcucValueCollection

Class	***EngineeringObject*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This class specifies an engineering object. Usually such an object is represented by a file artifact. The properties of engineering object are such that the artifact can be found by querying an ASAM catalog file. The engineering object is uniquely identified by domain+category+shortLabel+revisionLabel.			
Base	*ARObject*			
Subclasses	AutosarEngineeringObject, BuildEngineeringObject, Graphic			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
category	NameToken	1	attr	This denotes the role of the engineering object in the development cycle. Categories are such as • SWSRC for source code • SWOBJ for object code • SWHDR for a C-header file Further roles need to be defined via Methodology. **Tags:**xml.sequenceOffset=20
domain	NameToken	0..1	attr	This denotes the domain in which the engineering object is stored. This allows to indicate various segments in the repository keeping the engineering objects. The domain may segregate companies, as well as automotive domains. Details need to be defined by the Methodology. Attribute is optional to support a default domain. **Tags:**xml.sequenceOffset=40
revisionLabel	RevisionLabelString	*	attr	This is a revision label denoting a particular version of the engineering object. **Tags:**xml.sequenceOffset=30
shortLabel	NameToken	1	attr	This is the short name of the engineering object. Note that it is modeled as NameToken and not as Identifier since in ASAM-CC it is also a NameToken. **Tags:**xml.sequenceOffset=10

Table A.8: EngineeringObject

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable

▽

△

Class	***Identifiable*** (abstract)			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, AppOsTask ProxyToEcuTaskProxyMapping, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartition Mapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *Atp Feature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifestAddressable Object*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResourceDefinition, Block State, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, CouplingPort, *CouplingPortStructuralElement*, *CpSoftwareClusterResource*, CpSoftwareCluster ResourceToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareCluster ToResourceMapping, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, Dependency OnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, DoIp RoutingActivation, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidation Condition, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, F MFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlatInstance Descriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *Frame Triggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignal ToIPduMapping, ISignalTriggering, *IdentCaption*, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, Mc DataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, Network Endpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *Physical Channel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, Possible ErrorReaction, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, Rte EventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventInSystem Separation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEvent ContextProps, *ServerCallPoint*, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalService TranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *Spec ElementReference*, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcTo ImplMapping, SystemMapping, TDCpSoftwareClusterMapping, TDCpSoftwareClusterResourceMapping, TcpOptionFilterList, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, Traceable Table, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, Variable Access, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=-25

▽

△

Class	***Identifiable*** (abstract)			
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table A.9: Identifiable

Class	***Implementation*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	BswImplementation, SwcImplementation			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
buildAction Manifest	BuildActionManifest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	*	aggr	Specifies the provided implementation code.

▽

△

Class	***Implementation*** (abstract)			
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released
generated Artifact	DependencyOnArtifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is <<atpSplitable>> because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=mcSupport
programming Language	Programminglanguage Enum	0..1	attr	Programming language the implementation was created in.
requiredArtifact	DependencyOnArtifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
required GeneratorTool	DependencyOnArtifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
resource Consumption	ResourceConsumption	0..1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=resourceConsumption.shortName
swcBsw Mapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or Bsw Implementtion or for both.
swVersion	RevisionLabelString	0..1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.
usedCode Generator	String	0..1	attr	Optional: code generator used.
vendorId	PositiveInteger	0..1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table A.10: Implementation

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:**atp.recommendedPackage=ImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to True, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=symbolProps.shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table A.11: ImplementationDataType

Enumeration	**MemoryAllocationKeywordPolicyType**
Package	M2::MSR::DataDictionary::AuxillaryObjects
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Literal	***Description***
addrMethodShort Name	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod. This is the default value if the attribute does not exist. **Tags:**atp.EnumerationLiteralIndex=0
addrMethodShort NameAndAlignment	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod and a variable alignment postfix. Thereby the alignment postfix needs to be consistent with the alignment attribute of the related MemorySection. **Tags:**atp.EnumerationLiteralIndex=1

Table A.12: MemoryAllocationKeywordPolicyType

Class	**MemorySection**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage			
Note	Provides a description of an abstract memory section used in the Implementation for code or data. It shall be declared by the Implementation Description of the module or component, which actually allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the generated Implementation Description of the RTE shall contain the corresponding MemorySections. The attribute "symbol" (if symbol is missing: "shortName") defines the module or component specific section name used in the code. For details see the document "Specification of Memory Mapping". Typically the section name is build according the pattern: <SwAddrMethod shortName>[_<further specialization nominator>][_<alignment>] where • **[<SwAddrMethod shortName>]** is the shortName of the referenced SwAddrMethod • **[_<further specialization nominator>]** is an optional infix to indicate the specialization in the case that several MemorySections for different purpose of the same Implementation Description referring to the same or equally named SwAddrMethods. • **[_<alignment>]** is the alignment attributes value and is only applicable in the case that the memoryAllocationKeywordPolicy value of the referenced SwAddrMethod is set to addrMethod ShortNameAndAlignment MemorySection used to Implement the code of RunnableEntitys and BswSchedulableEntitys shall have a symbol (if missing: shortName) identical to the referred SwAddrMethod to conform to the generated RTE header files. In addition to the section name described above, a prefix is used in the corresponding macro code in order to define a name space. This prefix is by default given by the shortName of the BswModule Description resp. the SwComponentType. It can be superseded by the prefix attribute.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
alignment	AlignmentType	0..1	attr	The attribute describes the typical alignment of objects within this memory section.
executableEntity	ExecutableEntity	*	ref	Reference to the ExecutableEntitites located in this section. This allows to locate different Executable Entitities in different sections even if the associated Sw Addrmethod is the same. This is applicable to code sections only.
memClass Symbol	CIdentifier	0..1	attr	Defines a specific symbol in order to generate the compiler abstraction "memclass" code for this Memory Section. The existence of this attribute supersedes the usage of swAddrmethod.shortName for this purpose. The complete name of the "memclass" preprocessor symbol is constructed as <prefix>_<memClassSymbol> where prefix is defined in the same way as for the enclosing MemorySection. See also AUTOSAR_SWS_CompilerAbstraction SWS_COMPILER_00040. **Tags:**atp.Status=obsolete
option	Identifier	*	attr	This attribute introduces the ability to specify further intended properties of this MemorySection. The following two values are standardized (to be used for code sections only and exclusively to each other): • INLINE - The code section is declared with the compiler abstraction macro INLINE. • LOCAL_INLINE - The code section is declared with the compiler abstraction macro LOCAL_INLINE

△

Class	**MemorySection**			
				△ In both cases (INLINE and LOCAL_INLINE) the inline expansion depends on the compiler specific implementation of these macros. Depending on this, the code section either corresponds to an actual section in memory or is put into the section of the caller. See AUTOSAR_SWS_CompilerAbstraction for more details.
prefix	SectionNamePrefix	0..1	ref	The prefix used to set the memory section's namespace in the code. The existence of a prefix element supersedes rules for a default prefix (such as the Bsw ModuleDescription's shortName). This allows the user to define several name spaces for memory sections within the scope of one module, cluster or SWC.
size	PositiveInteger	0..1	attr	The size in bytes of the section.
swAddrmethod	SwAddrMethod	0..1	ref	This association indicates that this module specific (abstract) memory section is part of an overall SwAddr Method, referred by the upstream declarations (e.g. calibration parameters, data element prototypes, code entities) which share a common addressing strategy. This can be evaluated for the ECU configuration of the build support. This association shall always be declared by the Implementation description of the module or component, which allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the software components only declare the grouping of its data prototypes to SwAddrMethods, and the generated Implementation Description of the RTE actually sets up this association.
symbol	Identifier	0..1	attr	Defines the section name as explained in the main description. By using this attribute for code generation (instead of the shortName) it is possible to define several different MemorySections having the same name - e.g. symbol = CODE - but using different sectionName Prefixes.

Table A.13: MemorySection

Enumeration	**MemorySectionType**
Package	M2::MSR::DataDictionary::AuxillaryObjects
Note	Enumeration to specify the essential nature of the data which can be allocated in a common memory class by the means of the AUTOSAR Memory Mapping.
Literal	***Description***
calibrationVariables	This memory section is reserved for "virtual variables" that are computed by an MCD system during a measurement session but do not exist in the ECU memory. **Tags:**atp.EnumerationLiteralIndex=2
calprm	To be used for calibratable constants of ECU-functions. **Tags:**atp.EnumerationLiteralIndex=3
code	To be used for mapping code to application block, boot block, external flash etc. **Tags:**atp.EnumerationLiteralIndex=4
configData	Constants with attributes that show that they reside in one segment for module configuration. **Tags:**atp.EnumerationLiteralIndex=5

▽

△

Enumeration	**MemorySectionType**
const	To be used for global or static constants. **Tags:**atp.EnumerationLiteralIndex=6
excludeFromFlash	This memory section is reserved for "virtual parameters" that are taken for computing the values of so-called dependent parameter of an MCD system. Dependent Parameters that are not at the same time "virtual parameters" are allocated in the ECU memory. Virtual parameters, on the other hand, are not allocated in the ECU memory. Virtual parameters exist in the ECU Hex file for the purpose of being considered (for computing the values of dependent parameters) during an offline-calibration session. **Tags:**atp.EnumerationLiteralIndex=7
var	To be used for global or static variables. The expected initialization is specified with the attribute sectionInitializationPolicy. **Tags:**atp.EnumerationLiteralIndex=9

Table A.14: MemorySectionType

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:**xml.sequenceOffset=-90

Table A.15: Referrable

Class	**RunnableEntity**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	A RunnableEntity represents the smallest code-fragment that is provided by an AtomicSwComponentType and are executed under control of the RTE. RunnableEntities are for instance set up to respond to data reception or operation invocation on a server.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	RunnableEntityArgument	*	aggr	This represents the formal definition of a an argument to a RunnableEntity.

▽

△

Class	**RunnableEntity**			
asynchronous ServerCall ResultPoint	AsynchronousServer CallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
canBeInvoked Concurrently	Boolean	0..1	attr	If the value of this attribute is set to "true" the enclosing RunnableEntity can be invoked concurrently (even for one instance of the corresponding AtomicSwComponent Type). This implies that it is the responsibility of the implementation of the RunnableEntity to take care of this form of concurrency.
dataRead Access	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReadAccess.shortName, dataReadAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceive PointBy Argument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReceivePointByArgument.shortName, dataReceivePointByArgument.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
dataReceive PointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointBy Value is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataReceivePointByValue.shortName, data ReceivePointByValue.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataSendPoint.shortName, dataSend Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataWrite Access	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataWriteAccess.shortName, dataWrite Access.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
external TriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint.ident.shortName, externalTriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internal TriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint.shortName, internal TriggeringPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
modeAccess Point	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeAccessPoint.ident.shortName, mode AccessPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeSwitch Point	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeSwitchPoint.shortName, modeSwitch Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
parameter Access	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a Parameter DataPrototype which may either be local or within a Port Prototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of Parameter Access (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=parameterAccess.shortName, parameter Access.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of read LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=readLocalVariable.shortName, readLocal Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
serverCallPoint	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serverCallPoint.shortName, serverCall Point.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
symbol	CIdentifier	0..1	attr	The symbol describing this RunnableEntity's entry point. This is considered the API of the RunnableEntity and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The WaitPoint associated with the RunnableEntity.
writtenLocal Variable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of written LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=writtenLocalVariable.shortName, written LocalVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.16: RunnableEntity

Class	**SectionNamePrefix**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage			
Note	A prefix to be used for generated code artifacts defining a memory section name in the source code of the using module or SWC.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
implementedIn	DependencyOnArtifact	0..1	ref	Optional reference that allows to Indicate the code artifact (header file) containing the preprocessor implementation of memory sections with this prefix. The usage of this link supersedes the usage of a memory mapping header with the default name (derived from the BswModuleDescription's shortName).

Table A.17: SectionNamePrefix

Class	**SwAddrMethod**			
Package	M2::MSR::DataDictionary::AuxillaryObjects			
Note	Used to assign a common addressing method, e.g. common memory section, to data or code objects. These objects could actually live in different modules or components. **Tags:**atp.recommendedPackage=SwAddrMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
memory Allocation KeywordPolicy	MemoryAllocation KeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.

▽

△

Class	**SwAddrMethod**			
option	Identifier	*	attr	This attribute introduces the ability to specify further intended properties of the MemorySection in with the related objects shall be placed. These properties are handled as to be selected. The intended options are mentioned in the list. In the Memory Mapping configuration, this option list is used to determine an appropriate MemMapAddressing ModeSet.
section Initialization Policy	SectionInitialization PolicyType	0..1	attr	Specifies the expected initialization of the variables (inclusive those which are implementing VariableData Prototypes). Therefore this is an implementation constraint for initialization code of BSW modules (especially RTE) as well as the start-up code which initializes the memory segment to which the AutosarData Prototypes referring to the SwAddrMethod's are later on mapped. If the attribute is not defined it has the identical semantic as the attribute value "INIT"
sectionType	MemorySectionType	0..1	attr	Defines the type of memory sections which can be associated with this addresssing method.

Table A.18: SwAddrMethod

Class	**SwBaseType**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:**atp.recommendedPackage=BaseTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *BaseType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.19: SwBaseType

Class	***SwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
consistency Needs	ConsistencyNeeds	*	aggr	This represents the collection of ConsistencyNeeds owned by the enclosing SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=consistencyNeeds.shortName, consistency Needs.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	***SwComponentType*** (abstract)			
port	PortPrototype	*	aggr	The PortPrototypes through which this SwComponent Type can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=port.shortName, port.variationPoint.short Label vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swcMapping Constraint	SwComponentMapping Constraints	*	ref	Reference to constraints that are valid for this Sw ComponentType.
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, sw ComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which UnitGroups are relevant in the context of referencing SwComponentType.

Table A.20: SwComponentType

Class	**SwcImplementation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcImplementation			
Note	This meta-class represents a specialization of the general Implementation meta-class with respect to the usage in application software. **Tags:**atp.recommendedPackage=SwcImplementations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
behavior	SwcInternalBehavior	0..1	ref	The internal behavior implemented by this Implementation.
perInstance MemorySize	PerInstanceMemory Size	*	aggr	Allows a definition of the size of the per-instance memory for this implementation. The aggregation of PerInstance MemorySize is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects, in this case PerInstanceMemory. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcImplementation**			
required RTEVendor	String	0..1	attr	Identify a specific RTE vendor. This information is potentially important at the time of integrating (in particular: linking) the application code with the RTE. The semantics is that (if the association exists) the corresponding code has been created to fit to the vendor-mode RTE provided by this specific vendor. Attempting to integrate the code with another RTE generated in vendor mode is in general not possible.

Table A.21: SwcImplementation

Class	**SwcInternalBehavior**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	The SwcInternalBehavior of an AtomicSwComponentType describes the relevant aspects of the software-component with respect to the RTE, i.e. the RunnableEntities and the RTEEvents they respond to.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arTypedPerInstanceMemory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is set to "true" or if the component defines NVRAM access via permanent blocks. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=arTypedPerInstanceMemory.shortName, arTypedPerInstanceMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
event	RTEEvent	*	aggr	This is a RTEEvent specified for the particular SwcInternalBehavior. The aggregation of RTEEvent is subject to variability with the purpose to support the conditional existence of RTE events. Note: the number of RTE events might vary due to the conditional existence of PortPrototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveAreaPolicy	SwcExclusiveAreaPolicy	*	aggr	Options how to generate the ExclusiveArea related APIs. When no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, exclusiveAreaPolicy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
explicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=explicitInterRunnableVariable.shortName, explicitInterRunnableVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
handle TerminationAnd Restart	HandleTerminationAnd RestartEnum	0..1	attr	This attribute controls the behavior with respect to stopping and restarting. The corresponding AtomicSw ComponentType may either not support stop and restart, or support only stop, or support both stop and restart.
implicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnable Variable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implicitInterRunnableVariable.shortName, implicitInterRunnableVariable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a software component for its implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedDataTypeSet
includedMode Declaration GroupSet	IncludedMode DeclarationGroupSet	*	aggr	This aggregation represents the included Mode DeclarationGroups **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=includedModeDeclarationGroupSet
instantiation DataDefProps	InstantiationDataDef Props	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of Port Prototypes and component local memories like "per InstanceParameter" or "arTypedPerInstanceMemory". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationData DefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
perInstance Memory	PerInstanceMemory	*	aggr	Defines a per-instance memory object needed by this software component. The aggregation of PerInstance Memory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceMemory.shortName, perInstance Memory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstance Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) that needs to be available for each instance of the software-component. This is typically only useful if supportsMultipleInstantiation is set to "true". The aggregation of perInstanceParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=perInstanceParameter.shortName, per InstanceParameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portAPIOption	PortAPIOption	*	aggr	Options for generating the signature of port-related calls from a runnable to the RTE and vice versa. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portAPIOption, portAPIOption.variation Point.shortLabel vh.latestBindingTime=preCompileTime
runnable	RunnableEntity	*	aggr	This is a RunnableEntity specified for the particular Swc InternalBehavior. The aggregation of RunnableEntity is subject to variability with the purpose to support the conditional existence of RunnableEntities. Note: the number of RunnableEntities might vary due to the conditional existence of Port Prototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=runnable.shortName, runnable.variation Point.shortLabel vh.latestBindingTime=preCompileTime
service Dependency	SwcService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation of SwcServiceDependency is subject to variability with the purpose to support the conditional existence of ports as well as the conditional existence of ServiceNeeds. The SwcServiceDependency owned by an SwcInternal Behavior can be located in a different physical file in order to support that SwcServiceDependency might be provided in later development steps or even by different ▽

▽

△

Class	SwcInternalBehavior			
				△ expert domain (e.g OBD expert for Obd related Service Needs) tools. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceDependency.shortName, service Dependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
shared Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same Sw ComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sharedParameter.shortName, shared Parameter.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
supports Multiple Instantiation	Boolean	0..1	attr	Indicate whether the corresponding software-component can be multiply instantiated on one ECU. In this case the attribute will result in an appropriate component API on programming language level (with or without instance handle).
variationPoint Proxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Stereotypes:** atpSplitable **Tags:**atp.Splitkey=variationPointProxy.shortName

Table A.22: SwcInternalBehavior

Class	SwcToImplMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::SWmapping			
Note	Map instances of an AtomicSwComponentType to a specific Implementation.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
component	SwComponent Prototype	1..*	iref	Reference to the software component instances that are being mapped to the specified Implementation. The targeted SwComponentPrototype needs be of the Atomic SwComponentType being implemented by the referenced Implementation. **InstanceRef implemented by:**ComponentInSystem InstanceRef
component Implementation	SwcImplementation	1	ref	Reference to a specific Implementation description. Implementation to be used by the specified SW component instance. This allows to achieve more precise estimates for the resource consumption that results from mapping the instance of an atomic SW component onto an ECU.

Table A.23: SwcToImplMapping

Class	**SystemMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The system mapping aggregates all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints).			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
application PartitionToEcu Partition Mapping	ApplicationPartitionTo EcuPartitionMapping	*	aggr	Mapping of ApplicationPartitions to EcuPartitions **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=applicationPartitionToEcuPartition Mapping.shortName, applicationPartitionToEcuPartition Mapping.variationPoint.shortLabel vh.latestBindingTime=postBuild
appOsTask ProxyToEcu TaskProxy Mapping	AppOsTaskProxyToEcu TaskProxyMapping	*	aggr	Mapping of an OsTaskProxy that was created in the context of a SwComponent to an OsTaskProxy that was created in the context of an Ecu.
com Management Mapping	ComManagement Mapping	*	aggr	Mappings between Mode Management PortGroups and communication channels. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=systemDesignTime
cryptoService Mapping	CryptoServiceMapping	*	aggr	This aggregation represents the collection of crypto service mappings in the context of the enclosing System Mapping. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=cryptoServiceMapping.shortName, crypto ServiceMapping.variationPoint.shortLabel vh.latestBindingTime=postBuild
dataMapping	DataMapping	*	aggr	The data mappings defined. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
ecuResource Mapping	ECUMapping	*	aggr	Mapping of hardware related topology elements onto their counterpart definitions in the ECU Resource Template. atpVariation: The ECU Resource type might be variable. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=systemDesignTime
j1939Controller ApplicationTo J1939NmNode Mapping	J1939Controller ApplicationToJ1939Nm NodeMapping	*	aggr	Mapping of a J1939ControllerApplication to a J1939Nm Node.
mapping Constraint	MappingConstraint	*	aggr	Constraints that limit the mapping freedom for the mapping of SW components to ECUs. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=systemDesignTime
pncMapping	PncMapping	*	aggr	Mappings between Virtual Function Clusters and Partial Network Clusters. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=systemDesignTime

△

Class	**SystemMapping**			
portElementTo ComResource Mapping	PortElementTo Communication ResourceMapping	*	aggr	maps a communication resource to CP Software Clusters **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portElementToComResourceMapping.short Name, portElementToComResourceMapping.variation Point.shortLabel atp.Status=draft vh.latestBindingTime=postBuild
resource Estimation	EcuResourceEstimation	*	aggr	Resource estimations for this set of mappings, zero or one per ECU instance. atpVariation: Used ECUs are variable. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=systemDesignTime
resourceTo Application Partition Mapping	CpSoftwareCluster ResourceToApplication PartitionMapping	*	aggr	Maps a Software Cluster resource to an Application Partition to restrict the usage. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=resourceToApplicationPartition Mapping.shortName, resourceToApplicationPartition Mapping.variationPoint.shortLabel atp.Status=draft vh.latestBindingTime=systemDesignTime
rteEvent Separation	RteEventInSystem Separation	*	aggr	Separation constraint that limits the mapping freedom for the mapping of RteEvents to OsTasks in the System context.
rteEventToOs TaskProxy Mapping	RteEventInSystemToOs TaskProxyMapping	*	aggr	Constraint that enforces a mapping of RteEvent to a particular OsTask in the System context.
signalPath Constraint	SignalPathConstraint	*	aggr	Constraints that limit the mapping freedom for the mapping of data elements to signals. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=systemDesignTime
softwareCluster ToResource Mapping	CpSoftwareClusterTo ResourceMapping	*	aggr	maps a service resource to CP Software Clusters **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=softwareClusterToResourceMapping.short Name, softwareClusterToResourceMapping.variation Point.shortLabel atp.Status=draft vh.latestBindingTime=preCompileTime
swCluster Mapping	CpSoftwareClusterTo EcuInstanceMapping	*	aggr	The mappings of SW cluster to ECUs. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swClusterMapping.shortName, swCluster Mapping.variationPoint.shortLabel atp.Status=draft vh.latestBindingTime=systemDesignTime
swcTo Application Partition Mapping	SwcToApplication PartitionMapping	*	aggr	Allows to map a given SwComponentPrototype to a formally defined partition at a point in time when the corresponding EcuInstance is not yet known or defined. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=swcToApplicationPartitionMapping.short Name, swcToApplicationPartitionMapping.variation Point.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	**SystemMapping**			
swImplMapping	SwcToImplMapping	*	aggr	The mappings of AtomicSoftwareComponent Instances to Implementations. atpVariation: Derived, because SwcToEcuMapping is variable. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
swMapping	SwcToEcuMapping	*	aggr	The mappings of SW components to ECUs. atpVariation: SWC shall be mapped to other ECUs. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table A.24: SystemMapping

Class	**VariableDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided. In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table A.25: VariableDataPrototype

B Not applicable requirements

[SWS_MemMap_00999] ⌈These requirements are not applicable to this specification.⌋ *(SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00344, SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00171, SRS_BSW_00170, SRS_BSW_00419, SRS_BSW_00383, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00101, SRS_BSW_00416, SRS_BSW_00406, SRS_BSW_00168, SRS_BSW_00407, 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_00336, SRS_BSW_00337, SRS_BSW_00369, SRS_BSW_00339, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00323, SRS_BSW_00004, SRS_BSW_00409, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00164, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00300, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00310, SRS_BSW_00373, SRS_BSW_00327, SRS_BSW_00335, SRS_*

BSW_00350, *SRS_BSW_00408*, *SRS_BSW_00410*, *SRS_BSW_00411*, *SRS_BSW_00346*, *SRS_BSW_00314*, *SRS_BSW_00348*, *SRS_BSW_00353*, *SRS_BSW_00301*, *SRS_BSW_00302*, *SRS_BSW_00312*, *SRS_BSW_00357*, *SRS_BSW_00377*, *SRS_BSW_00378*, *SRS_BSW_00308*, *SRS_BSW_00309*, *SRS_BSW_00358*, *SRS_BSW_00414*, *SRS_BSW_00359*, *SRS_BSW_00360*, *SRS_BSW_00330*, *SRS_BSW_00331*, *SRS_BSW_00009*, *SRS_BSW_00401*, *SRS_BSW_00172*, *SRS_BSW_00010*, *SRS_BSW_00333*, *SRS_BSW_00341*, *SRS_BSW_00334*, *SRS_BSW_00305*, *SRS_BSW_00380*, *SRS_BSW_00438*, *SRS_BSW_00439*, *SRS_BSW_00440*, *SRS_BSW_00447*, *SRS_BSW_00448*, *SRS_BSW_00449*, *SRS_BSW_00450*, *SRS_BSW_00451*, *SRS_BSW_00452*, *SRS_BSW_00453*, *SRS_BSW_00454*, *SRS_BSW_00456*, *SRS_BSW_00457*, *SRS_BSW_00458*, *SRS_BSW_00459*, *SRS_BSW_00460*, *SRS_BSW_00461*, *SRS_BSW_00462*, *SRS_BSW_00003*, *SRS_BSW_00304*, *SRS_BSW_00318*, *SRS_BSW_00321*, *SRS_BSW_00374*, *SRS_BSW_00379*, *SRS_BSW_00402*, *SRS_BSW_00463*, *SRS_BSW_00466*, *SRS_BSW_00467*, *SRS_BSW_00469*, *SRS_BSW_00470*, *SRS_BSW_00471*, *SRS_BSW_00472*, *SRS_BSW_00473*, *SRS_BSW_00478*, *SRS_BSW_00479*, *SRS_BSW_00480*, *SRS_BSW_00481*, *SRS_BSW_00482*)