Document Title	Specification of Memory Mapping
Document Category	Software specification (SWS)
Document UID	128
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Removal of DebugTraceProfile
2025-11-27	R25-11	AUTOSAR Release Management	• Mandatory MAKW building rules • Clarify usage of CoreScope • Removal of deprecated MemMap values
2024-11-27	R24-11	AUTOSAR Release Management	• Simplify MemMap header implementation • Add and rework examples • Add requirements for function level tracing
2023-11-23	R23-11	AUTOSAR Release Management	• Clarify usage of GLOBAL and LOCAL coreScope • Improve document readability
2022-11-24	R22-11	AUTOSAR Release Management	• Correction of inconsistent MAKW patterns and examples • Resolve incompatibility to [constr_4103] • Add 64bit alignment support • Deprecate compiler abstraction
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

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

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

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

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 8
2 Acronyms and Abbreviations 9
3 Related documentation 10
3.1 Input documents . . . 10
3.2 Related standards and norms . . . 11
3.3 Related specification . . . 11
4 Constraints and assumptions 12
4.1 Limitations . . . 12
4.2 Applicability to car domains . . . 12
5 Dependencies to other modules 13
5.1 File structure . . . 13
5.1.1 Code file structure . . . 13
5.1.2 Header file structure . . . 13
6 Requirements traceability 15
7 Functional specification 18
7.1 General issues . . . 18
7.2 Mapping of Variables and Code . . . 19
7.2.1 Splitting of Modules in allocatable Memory Parts . . . 27
7.2.2 Config Constants versus non-config Constants . . . 27
7.2.3 Variable Sections . . . 28
7.2.4 Constant and Calibration Sections . . . 31
7.2.5 Code Sections . . . 33
7.3 Requirements on Memory Mapping Header Files . . . 39
7.4 Usage Examples . . . 44
7.4.1 Code Section . . . 44
7.4.2 Fast Variable Section . . . 47
7.4.3 Code Section in ICC2 cluster . . . 52
7.4.4 Callout sections . . . 54
7.4.5 Allocatable Memory Parts . . . 56
8 API specification 59
9 Sequence diagrams 60
10 Configuration specification 61
10.1 How to read this chapter . . . 61
10.2 Containers and configuration parameters . . . 61
10.2.1 MemMap . . . 61
10.2.2 MemMapAddressingModeSet . . . 62
10.2.3 MemMapAddressingMode . . . 66

10.2.4 MemMapAllocation 68
10.2.5 MemMapGenericMapping 70
10.2.6 MemMapSectionSpecificMapping 72
10.2.7 MemMapMappingSelector 74
10.3 Published Information 74
A Appendix 75
A.1 Referenced Meta Classes 75
A.2 Source Code Example for ADC 107
A.3 Memory Mapping Header File Example for ADC 108
A.4 Specification Items 111
A.4.1 Added Specification Items in R26-11 111
A.4.2 Changed Specification Items in R26-11 111
A.4.3 Deleted Specification Items in R26-11 111
A.5 Trace Items not relevant for this document 111

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
Besides symbols with defined alignment (e.g. code) further symbols of different alignment (e.g. 8, 16 and 64 bit) and size have to be allocated. If unsorted, the linker will leave gaps in the memory in between those symbols. This is because the microcontroller platform requires a specific alignment of those symbols and the linkers usually do not offer an optimization of variable allocation. This wastage of memory can be circumvented if the symbol are mapped to specific memory sections depending on their alignment. So an according mean is provided where required.

Usage of specific RAM properties
Some variables (e.g. the RAM mirrors of the NVRAM Manager) must not be initialized after a non cold-power-on resets. It shall be possible to map them to a RAM section that is not initialized at any reset except cold-power-on-reset. For some variables (e.g. variables that are accessed via bit masks) it improves both performance and code size if these are located within a RAM section that allows bit manipulation instructions of the compiler.

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 different contexts (e.g. boot loader and application), it is necessary to allow the mapping of symbols 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 and Partitioning
The usage of hardware memory protection requires an assignment of symbols to partitions. Therefore an additional separation of symbols into different memory (partition) areas is needed. Such shall be realized by identifying the BSW module or SWC MSN or additional feature prefixes as well as related software addressing methods.

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:
MAKW	Memory Allocation Key Word

Table 2.1: Abbreviations and Acronyms

3 Related documentation

3.1 Input documents

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[4] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[5] Basic Software Module Description Template
AUTOSAR_CP_TPS_BSWModuleDescriptionTemplate

[6] Methodology for Classic Platform
AUTOSAR_CP_TR_Methodology

[7] Guide to BSW Distribution
AUTOSAR_CP_EXP_BSWDistributionGuide

[8] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

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

The user interface of the memory allocation mechanisms is assumed to be supported by any ANSI-C compiler. Instead the implementation of the abstraction inside the memory mapping header files is hardware, compiler and compiler version specific and results in specific #pragmas. So the mode sets made available to the mechanism need to reflect this limitation to be able to map to it accordingly.

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.

Originally during specification of abstraction and validation of concept the compilers listed in chapter 3.1 have been considered. The mechanism is limited to those and other compilers supporting the user interface and according #pragma abstraction.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

[SWS_MemMap_00020]

Upstream requirements: SRS_BSW_00384, SRS_BSW_00351

⌈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].⌋

5.1 File structure

5.1.1 Code file structure

Not applicable.

5.1.2 Header file structure

[SWS_MemMap_00028]

Upstream requirements: SRS_BSW_00465, SRS_BSW_00415, SRS_BSW_00351, SRS_BSW_00464

⌈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.⌋

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]

Upstream requirements: SRS_BSW_00465, SRS_BSW_00415, SRS_BSW_00351, SRS_BSW_00464

⌈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.⌋

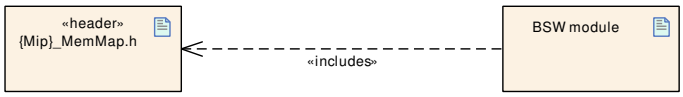


Figure 5.1: Basic Software Module specific memory mapping header file

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]

Upstream requirements: SRS_BSW_00465, SRS_BSW_00415, SRS_BSW_00351, SRS_BSW_00464

⌈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.⌋

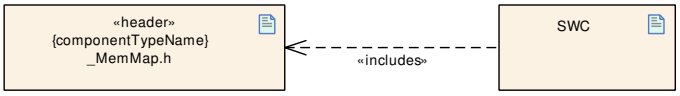


Figure 5.2: Software Component type specific memory mapping header file

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_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_00010] [SWS_MemMap_00036]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_MemMap_99999]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_MemMap_99999]
[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_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_MemMap_00001] [SWS_MemMap_00005]
[SRS_BSW_00345]	BSW Modules shall support pre-compile configuration	[SWS_MemMap_00003]
[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] [SWS_MemMap_00043] [SWS_MemMap_00044] [SWS_MemMap_00045] [SWS_MemMap_00046] [SWS_MemMap_00060] [SWS_MemMap_00061] [SWS_MemMap_00062] [SWS_MemMap_00063] [SWS_MemMap_00064] [SWS_MemMap_00070] [SWS_MemMap_00071] [SWS_MemMap_00072] [SWS_MemMap_00073] [SWS_MemMap_00080] [SWS_MemMap_00081] [SWS_MemMap_00082] [SWS_MemMap_00083]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_MemMap_99999]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_MemMap_99999]



△

Requirement	Description	Satisfied by
[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_99999]
[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_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_MemMap_99999]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_MemMap_99999]
[SRS_BSW_00389]	Containers shall have names	[SWS_MemMap_99999]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_MemMap_99999]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_MemMap_99999]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_MemMap_99999]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_MemMap_99999]
[SRS_BSW_00403]	The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets	[SWS_MemMap_99999]
[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_99999]
[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_99999]
[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_99999]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the Dem	[SWS_MemMap_99999]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_MemMap_99999]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_MemMap_99999]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_MemMap_00038] [SWS_MemMap_00043] [SWS_MemMap_00044] [SWS_MemMap_00060] [SWS_MemMap_00061] [SWS_MemMap_00062] [SWS_MemMap_00063] [SWS_MemMap_00064] [SWS_MemMap_00070] [SWS_MemMap_00071] [SWS_MemMap_00072] [SWS_MemMap_00073] [SWS_MemMap_00080] [SWS_MemMap_00081] [SWS_MemMap_00082] [SWS_MemMap_00083]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_MemMap_00022]
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module	[SWS_MemMap_99999]
[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_00469]	Fault detection and healing of production errors	[SWS_MemMap_99999]
[SRS_BSW_00471]	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	[SWS_MemMap_99999]
[SRS_BSW_00472]	Avoid detection of two production errors with the same root cause.	[SWS_MemMap_99999]
[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_99999]
[SRS_BSW_00490]	List possible security events	[SWS_MemMap_99999]
[SRS_BSW_00491]	Specification of trigger conditions and context data	[SWS_MemMap_99999]

Table 6.1: Requirements Tracing

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]

Upstream requirements: SRS_BSW_00328

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

[SWS_MemMap_00002]

Upstream requirements: SRS_BSW_00351

⌈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.⌋

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]

Upstream requirements: SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351

⌈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.⌋

[SWS_MemMap_00036]

Upstream requirements: SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351

⌈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.⌋

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 SwComponentTypes in the input configuration. To determine the number of MemorySections the applicable SwcImplementations 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 SwcImplementations provides as well the number of MemorySections 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

[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 7.2.3, 7.2.4 and subsection 7.2.5 defines the standardized names for those memory allocation keywords. If an implementation needs to extend the definition of a memory allocation keyword it shall use the refinement possibility for <NAME> as specified in [SWS_MemMap_00042]. Additionally, the <PREFIX> of the memory allocation keyword can be adapted conform to [SWS_MemMap_00040].

[SWS_MemMap_00038]

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Each AUTOSAR basic software module and software component shall support the configuration of the following different Section Types:

- `VAR` as described in [SWS_MemMap_00060].
- `VAR_FAST` as described in [SWS_MemMap_00061].
- `VAR_SLOW` as described in [SWS_MemMap_00062].
- `INTERNAL_VAR` as described in [SWS_MemMap_00063].
- `VAR_SAVED_ZONE` as described in [SWS_MemMap_00064].
- `CONST` as described in [SWS_MemMap_00070].
- `CONST_SAVED_RECOVERY_ZONE` as described in [SWS_MemMap_00071].
- `CONFIG_DATA` as described in [SWS_MemMap_00072].
- `CALIB` as described in [SWS_MemMap_00073].
- `CODE` as described in [SWS_MemMap_00080].
- `CODE_FAST` as described in [SWS_MemMap_00081].
- `CODE_SLOW` as described in [SWS_MemMap_00082].
- `CALLOUT_CODE` as described in [SWS_MemMap_00083].

The shortcut `{ALIGNMENT}` means the typical variable alignment which shall be selected according to the standard definition. 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.

64, used for variables and constants which typically have to be aligned to 64 bit. For instance used for variables and constants of size 64 bit or used for composite data types: arrays, structs and unions containing elements of maximum 64 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, 64 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 either a power on reset of the microcontroller or a power on reset of a battery backup memory itself after battery loss.

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

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.⌋

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]

Upstream requirements: SRS_BSW_00441, SRS_BSW_00351

⌈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 the <MIP> for BSW modules, if no SectionNamePrefix is defined for the MemorySection. <MIP> is the capitalized module implementation prefix built according to [SWS_BSW_00102].

  OR

- <PREFIX> is the symbol (case sensitive) of the SectionNamePrefix for BSW modules, if a SectionNamePrefix is defined for the MemorySection.

  OR

- <PREFIX> is the shortName (case sensitive) of the AtomicSwComponentType for software components.

  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.

⌋

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]

Upstream requirements: SRS_BSW_00351

⌈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).⌋

[SWS_MemMap_00039]

Upstream requirements: SRS_BSW_00351

⌈The part <NAME> from [SWS_MemMap_00022] shall contain the following {coreScope} keywords with the values GLOBAL as optional default without restrictions in memory access and LOCAL as mandatory alternative setting with restrictions in memory access to one desired core.

Consequently, the {coreScope} value GLOBAL shall not be written in the MAKW as well as SwAddrMethod name.

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 permitted 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 valid for non initialized memory sections.⌋

A detailed summary can be found in the following table. Further examples and usage hints are mentioned below.

Core Scope in MAKW or SwAddrMethod	**Valid for**	**Rationale**	**Useful for**
unset or GLOBAL	variables code constants config data calibration constants	A symbol can be accessed (read, write, execute) by any core in global address space. Any ModeSet with GLOBAL core scope can be used as allocation target. Thus, a symbol can be allocated close to a certain core using its GLOBAL ModeSets. GLOBAL scope shall be used for any user API which shall be available to other BSW modules, SWC or the RTE.	SWC BSW RTE CDD
LOCAL	variables code constants	A local symbol can be accessed (read, write, execute) by the core it is mapped to only. Only ModeSets with LOCAL core scope of the desired core can be used as allocation target.	BSW CDD

Table 7.2: Summary of Core Scope Behavior

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

Examples:

- ADC_START_SEC_CODE - is allocated to GLOBAL scope, as GLOBAL is default

- PWM_KERNEL_START_SEC_CODE_LOCAL - is allocated to LOCAL scope and can be mapped to a dedicated core using the unique prefix

Finally, it is an integrator decision to map memory section with the GLOBAL as well as LOCAL property to a core specific memory section. For GLOBAL the allocation target can be utilized to optimize the performance if the majority of memory accesses will occur from a specific core.

When using LOCAL, one shall be aware that the call tree accessing the symbol needs to be executed within at least the right core or at most the right partition on the right core. This is because otherwise memory protection errors or access violations might occur which usually lead to exceptional behaviour of the hardware.

More detailled recommendations on how to use the {coreScope} in an appropriate way can be found in the document [7].

[SWS_MemMap_00042]

Upstream requirements: SRS_BSW_00351

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

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 mandatory patterns for Memory Allocation Keywords are

```
{PREFIX}_START_SEC_CALIB[_{refinement}][_{safety}]_{ALIGNMENT}
{PREFIX}_STOP_SEC_CALIB[_{refinement}][_{safety}]_{ALIGNMENT}

{PREFIX}_START_SEC_CODE[_{refinement}][_{safety}][_{coreScope}]
{PREFIX}_STOP_SEC_CODE[_{refinement}][_{safety}][_{coreScope}]

{PREFIX}_START_SEC_CONFIG_DATA_{configClass}[_{refinement}][_{safety}]_{ALIGNMENT}
{PREFIX}_STOP_SEC_CONFIG_DATA_{configClass}[_{refinement}][_{safety}]_{ALIGNMENT}

{PREFIX}_START_SEC_CONST[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}
{PREFIX}_STOP_SEC_CONST[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}

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

Those are applied in the recommendations provided in subsection 7.2.3, 7.2.4 and subsection 7.2.5.

7.2.1 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]

Upstream requirements: SRS_BSW_00351

⌈When a BSW module is split into allocatable memory parts the `<PREFIX>` as described in [SWS_MemMap_00022] shall be build up according to [constr_4103] of [5].⌋

[SWS_MemMap_00041]

Upstream requirements: SRS_BSW_00351

⌈When a BSW module is split into allocatable memory parts all belonging `MemorySections.prefix` needs to reference a `SectionNamePrefix`.⌋

Please note the example given in 7.4.5.

<Msn>	**<vi>**	**<ai>**	**SectionNamePrefix.Symbol** (if SectionNamePrefix is defined)	**Resulting Prefix**
Fls	142	Ext	FLS_142_EXT_FEATURE	FLS_142_EXT_FEATURE
Fls	142	Ext	*undefined*	FLS_142_EXT
Adc	*don't care*	*undefined*	ADC_FEATURE	ADC_FEATURE
Adc	*don't care*	*undefined*	*undefined*	ADC

Table 7.3: Summary of Section Name Prefix for BSW Modules

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

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

See table in [SWS_MemMap_00072].

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[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}
{PREFIX}_STOP_SEC_CONST[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}
```

See table in [SWS_MemMap_00070].

7.2.3 Variable Sections

The following tables define keywords for variable sections:

[SWS_MemMap_00060] Section Type VAR

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}`
Description	To be used for all global or static variables. The name part _{refinement} shall be used to refine the allocation or initialization behavior. 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 safety QM 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	--

⌋

[SWS_MemMap_00061] Section Type VAR_FAST

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_FAST_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_FAST_{INIT_POLICY}[_{refinement}][_{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 _{refinement} shall be used to refine the allocation or initialization behavior. 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 safety QM 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	--

⌋

[SWS_MemMap_00062] Section Type VAR_SLOW

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

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

▽

△

Description	To be used for all infrequently accessed global or static variables. The name part _{refinement} shall be used to refine the allocation or initialization behavior. 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 safety QM 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	--

⌋

[SWS_MemMap_00063] Section Type INTERNAL_VAR

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_INTERNAL_VAR_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_INTERNAL_VAR_{INIT_POLICY}[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}`
Description	To be used for global or static variables those are accessible from a calibration tool. The name part _{refinement} shall be used to refine the allocation or initialization behavior. 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 safety QM 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	--

⌋

[SWS_MemMap_00064] Section Type VAR_SAVED_ZONE

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_VAR_SAVED_ZONE_{refinement}[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_VAR_SAVED_ZONE_{refinement}[_{safety}]_{ALIGNMENT}`
Description	To be used for RAM buffers of variables saved in non volatile memory. The name part _{refinement} shall denote at least 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 safety QM the attribute may be omitted. Note: As the SectionType is considered coreGlobal, the attribute {coreScope} does not need to be specified and is not part of the MAKW syntax.
Memory Section Type	VAR
Section Initialization Policy	`POWER-ON-CLEARED`
Status	--

⌋

7.2.4 Constant and Calibration Sections

The following tables define keywords for constant and calibration sections.

[SWS_MemMap_00070] Section Type CONST

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CONST[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CONST[_{refinement}][_{safety}][_{coreScope}]_{ALIGNMENT}`
Description	To be used for global or static constants. The name part _{refinement} is the typical period time value and unit of the ExecutableEntitys in this MemorySection. The name part _{refinement} 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 safety QM 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	CONST
Section Initialization Policy	--
Status	--

⌋

[SWS_MemMap_00071] Section Type CONST_SAVED_RECOVERY_ZONE

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CONST_SAVED_RECOVERY_ZONE_{refinement}[_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CONST_SAVED_RECOVERY_ZONE_{refinement}[_{safety}]_{ALIGNMENT}`
Description	To be used for ROM buffers of variables saved in non volatile memory. The name part _{refinement} shall denote at least 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 safety QM the attribute may be omitted. Note: As the SectionType is considered coreGlobal, the attribute {coreScope} does not need to be specified and is not part of the MAKW syntax.
Memory Section Type	CONST
Section Initialization Policy	--
Status	--

⌋

[SWS_MemMap_00072] Section Type CONFIG_DATA

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CONFIG_DATA_{configClass}[_{refinement}][_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CONFIG_DATA_{configClass}[_{refinement}][_{safety}]_{ALIGNMENT}`

▽

△

Description	Constants with attributes that show that they reside in one segment for module configuration. The name part _{configClass} shall contain the configClass with one of the strings PREBUILD or POSTBUILD. The name part _{refinement} shall be used to refine the memory allocation keyword to allow individual allocation. 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 safety QM the attribute may be omitted. In the related SwAddrMethod one option attribute shall describe the configClass with the possible values {configClassPreBuild, configClassPostBuild}. Note: As the SectionType is considered coreGlobal, the attribute {coreScope} does not need to be specified and is not part of the MAKW syntax.
Memory Section Type	CONFIG-DATA
Section Initialization Policy	`--`
Status	--

⌋

[SWS_MemMap_00073] Section Type CALIB

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CALIB[_{refinement}][_{safety}]_{ALIGNMENT}` `{PREFIX}_STOP_SEC_CALIB[_{refinement}][_{safety}]_{ALIGNMENT}`
Description	To be used for calibration constants. The name part _{refinement} shall be used to refine the memory allocation keyword to allow individual allocation. 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 safety QM the attribute may be omitted. Note: As the SectionType is considered coreGlobal, the attribute {coreScope} does not need to be specified and is not part of the MAKW syntax.
Memory Section Type	CALPRM
Section Initialization Policy	`--`
Status	--

⌋

7.2.5 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:

[SWS_MemMap_00080] Section Type CODE

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CODE[_{refinement}][_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CODE[_{refinement}][_{safety}][_{coreScope}]`
Description	To be used for mapping code to application block, boot block, external flash etc. The name part _{refinement} is the typical period time value and unit of the ExecutableEntitys in this MemorySection. The name part _{refinement} 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 safety QM 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	--

⌋

[SWS_MemMap_00081] Section Type CODE_FAST

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CODE_FAST[_{refinement}][_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CODE_FAST[_{refinement}][_{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 _{refinement} shall be used to refine the memory allocation keyword to allow individual allocation. 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 safety QM 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	--

⌋

[SWS_MemMap_00082] Section Type CODE_SLOW

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CODE_SLOW[_{refinement}][_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CODE_SLOW[_{refinement}][_{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 _{refinement} shall be used to refine the memory allocation keyword to allow individual allocation. 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 safety QM 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	--

⌋

[SWS_MemMap_00083] Section Type CALLOUT_CODE

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈

Syntax of Memory Allocation Keyword	`{PREFIX}_START_SEC_CALLOUT_CODE[_{refinement}][_{safety}][_{coreScope}]` `{PREFIX}_STOP_SEC_CALLOUT_CODE[_{refinement}][_{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 _{refinement} shall be used to refine the memory allocation keyword to allow individual allocation. 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 safety QM 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	--

⌋

[SWS_MemMap_00003]

Upstream requirements: SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00345, SRS_BSW_00351, SRS_BSW_00477

⌈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

Note: In between 1 to 5 there shall be no other preprocessor code added. This would prevent correct interpretation of source code and cause later preprocessor errors.

Note: For code which is invariably implemented as inline function the wrapping with Memory Allocation Keywords is not required.⌋

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.

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]

Upstream requirements: SRS_BSW_00306, SRS_BSW_00351, SRS_BSW_00477

⌈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).⌋

[SWS_MemMap_00023]

Upstream requirements: SRS_BSW_00306, SRS_BSW_00351, SRS_BSW_00477

⌈Memory mapping header files shall not be included inside the body of a function.⌋

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_MemMap_00083] 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 7.2.5 "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 7.4.4 Callout sections

7.3 Requirements on Memory Mapping Header Files

[SWS_MemMap_00005]

Upstream requirements: SRS_BSW_00328, SRS_BSW_00006, SRS_BSW_00306, SRS_BSW_00351

⌈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.⌋

[SWS_MemMap_00026]

Upstream requirements: SRS_BSW_00351

⌈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.⌋

[SWS_MemMap_00033]

Upstream requirements: SRS_BSW_00351

⌈All `MemorySection`s 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`⌋

Please note also [SWS_MemMap_00032].

[SWS_MemMap_00034]

Upstream requirements: SRS_BSW_00351

⌈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`⌋

Please note also [SWS_MemMap_00028].

[SWS_MemMap_00035]

Upstream requirements: SRS_BSW_00351

⌈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.⌋

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]

Upstream requirements: SRS_BSW_00351

⌈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.⌋

[SWS_MemMap_00015]

Upstream requirements: SRS_BSW_00306, SRS_BSW_00351

⌈The selected section shall be activated, if the section start macro is defined before including of the memory mapping header file.⌋

Assumption of use:
Before first usage of a memory mapping header file in a compilation unit it shall be ensured that all symbols are redirected to either default sections or special sections to collect those symbols if supported by the compiler / linker. This ensures that symbols with missing or wrong memory allocation can be detected.

[SWS_MemMap_00043]

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈If a section is selected, pragmas shall be set in a way to control the compiler / linker so that the intended symbol types are allocated properly.⌋

Please note that after selecting a section all symbols not covered by the selection are treated by the default settings (see Assumption of Use).

[SWS_MemMap_00006]

Upstream requirements: SRS_BSW_00306, SRS_BSW_00351

⌈The selected section shall be deactivated, if the section stop macro is defined before including of the memory mapping header file.⌋

[SWS_MemMap_00044]

Upstream requirements: SRS_BSW_00437, SRS_BSW_00351

⌈If a section is deselected the settings used before starting the section shall be restored if supported by the compiler / linker.⌋

[SWS_MemMap_00016]

Upstream requirements: SRS_BSW_00306, SRS_BSW_00351

⌈The selection of a section shall not be nested and only influence one of the three different symbol types of code, variables, or constants concurrently.⌋

Application hint:
The used pragmas behind a section shall be selected according to the manual of the used compiler / linker. In addition, the following hints might be considered:

- According to [SWS_MemMap_00043] the combination of code and constant pragmas below the same code section might be required to allow allocation of constants created by the compiler according to its optimization strategy.
- Setting combined pragmas for data as well as bss for allocation of variables under the same section might be useful to support initialized and uninitialized variables using the same initialization policy setting inside a section e.g., `INIT` can be used to initialize data to value and bss to zero similarly.
- Setting `#pragmas` for unused symbol types to undefined values shall be done to handle inaccurate non-handled symbols.

[SWS_MemMap_00047] ⌈To support the function level tracing it shall be possible to extend or replace the section name by the symbol and object file name. This allows a grouping of those symbols (functions, tasks, runnables) to one and the same memory group for tracing inside the linker invocation file (locator file).⌋

Rationale:
For the purpose of function level tracing it is required to group all relevant symbols into a contiguous memory area regardless of the previously used memory allocation keyword applied to it. But due to the fact, that usually several symbols share the same memory allocation keyword the section names need to be altered when generating the memory allocation header files to catch those in the locator file or by additional postprocessing tools tuning the memory allocation.

Usage hint:
Adding the symbol name to the section will cause significant build time impact depending on the used compiler. So it should be applied only when function level tracing is used.

[SWS_MemMap_00007]

Upstream requirements: SRS_BSW_00351

⌈The memory mapping header files shall check if they have been included with a valid Memory Allocation Keyword and in a valid - not nested - sequence (no START preceded by a START, no STOP without the corresponding START). This shall be done by a preprocessor check.⌋

[SWS_MemMap_00011]

Upstream requirements: SRS_BSW_00351

⌈The memory mapping header files shall undefine the module or software component specific Memory Allocation Key Words for starting or stopping a section.⌋

[SWS_MemMap_00013]

Upstream requirements: SRS_BSW_00351

⌈The memory mapping header files shall use if-else structures to reduce the compilation effort.⌋

[SWS_MemMap_00045]

Upstream requirements: SRS_BSW_00351

⌈The memory mapping header shall not contain sections of other BSW modules or software components.⌋

[SWS_MemMap_00046]

Upstream requirements: SRS_BSW_00351

⌈The memory mapping header files shall be used for memory allocation purpose only.⌋

Rationale:
As the memory mapping header files are usually generated or hand coded by the integration responsible party one can not assume that specific definitions will be provided.

According to previous requirements, the memory mapping header can be implemented as shown in the following example:

Example 7.4

```
/* Initialization of overall error handling */
#define MEMMAP_ERROR

/* Keyword evaluation */
#if defined {START_MAKW}
  #undef MEMMAP_ERROR
  #undef {START_MAKW}
  #ifndef MEMMAP_SEQUENCE_OPEN
    /* pragma start */
    {PRAGMAS}
    /* pragma end */
    #define MEMMAP_SEQUENCE_OPEN
    #define MEMMAP_SEQUENCE_OPEN_{SEQUENCE_MAKW}
  #else
    #error "{FileName}: {SEQUENCE_MAKW}: Please STOP the sequence
       before, START must not be followed by START!"
    #endif
#elif defined {STOP_MAKW}
  #undef MEMMAP_ERROR
  #undef {STOP_MAKW}
  #ifdef MEMMAP_SEQUENCE_OPEN
    #ifdef MEMMAP_SEQUENCE_OPEN_{SEQUENCE_MAKW}
      /* unhandled pragma start */
      {RESTORE_PRAGMAS}
      /* unhandled pragma end */
      #undef MEMMAP_SEQUENCE_OPEN
      #undef MEMMAP_SEQUENCE_OPEN_{SEQUENCE_MAKW}
    #else
```

```
28        #error "{FileName}: {SEQUENCE_MAKW}: START section is followed by
              wrong STOP section statement!"
29      #endif
30    #else
31      #error "{FileName}: {SEQUENCE_MAKW}: No START statement given,
           before STOP statement! STOP must not be followed by STOP!"
32    #endif
33  #endif
34
35  #if defined {START_MAKW} /* Next MAKW */
36    ...
37  #elif defined {STOP_MAKW}
38    ...
39  #endif
40
41  ...
42
43  /* Error evaluation */
44  #ifdef MEMMAP_ERROR
45    #undef MEMMAP_ERROR
46    #error "{FileName}: Undefined or missing START / STOP statement,
          please check your source code or re-generate the MemMap Header
          file!"
47  #endif
```

The used wildcards shall have the following meaning:

Wildcard	Explanation	Example
{START_MAKW}	Start MAKW	ADC_START_SEC_VAR_INIT_ASIL_B_32
{STOP_MAKW}	Stop MAKW	ADC_STOP_SEC_VAR_INIT_ASIL_B_32
{SEQUENCE_MAKW}	Keyword without START/STOP	ADC_SEC_VAR_INIT_ASIL_B_32
{FileName}	Name of the Memory Mapping Header File	Adc_MemMap.h
{PRAGMAS}	Pragmas used for allocation	/* Example Altium CTC */ #pragma section fardata "ram.partition_asil_b.32" #pragma section farbss "ram.partition_asil_b.32" #pragma clear #pragma section code "unhandled" #pragma section rodata "unhandled"
{RESTORE_PRAGMAS}	Pragmas for unhandled sections	/* Example Altium CTC */ #pragma section fardata "unhandled" #pragma section farbss "unhandled" #pragma section code "unhandled" #pragma section rodata "unhandled"

Table 7.4: MemMap Wildcards

Note:
Since its error prone to determine expected properties for memory which is not explicitly handled by Memory Allocation Key Words usually those symbols are treated in a way to cause linker errors. The unhandled or default sections might be used to catch those non-handled objects.

7.4 Usage 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.

7.4.1 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".

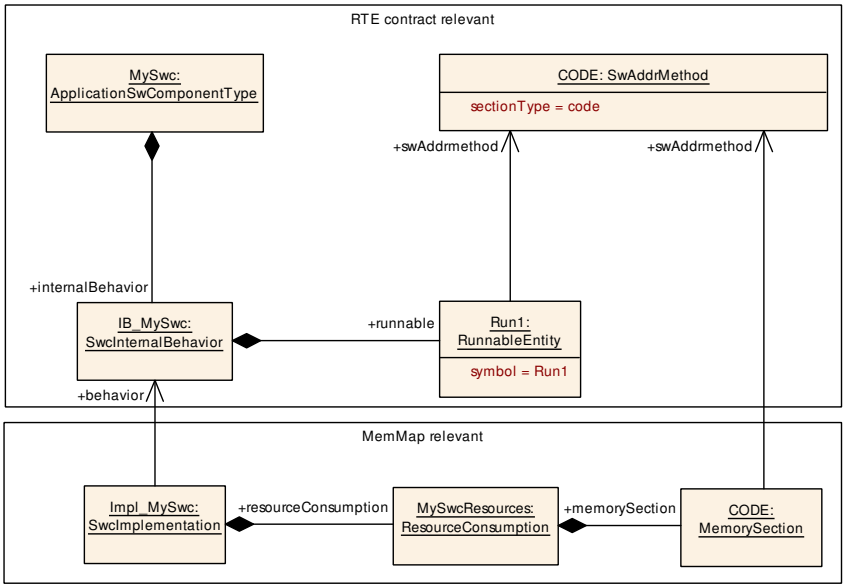


Figure 7.1: Example of `ApplicationSwComponentType` with code section

According the SWS RTE [8] the Runnable Entity prototype in the Application Header File of the software component is emitted as:

Example 7.5

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

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

void MySwc_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 "MySwc_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".

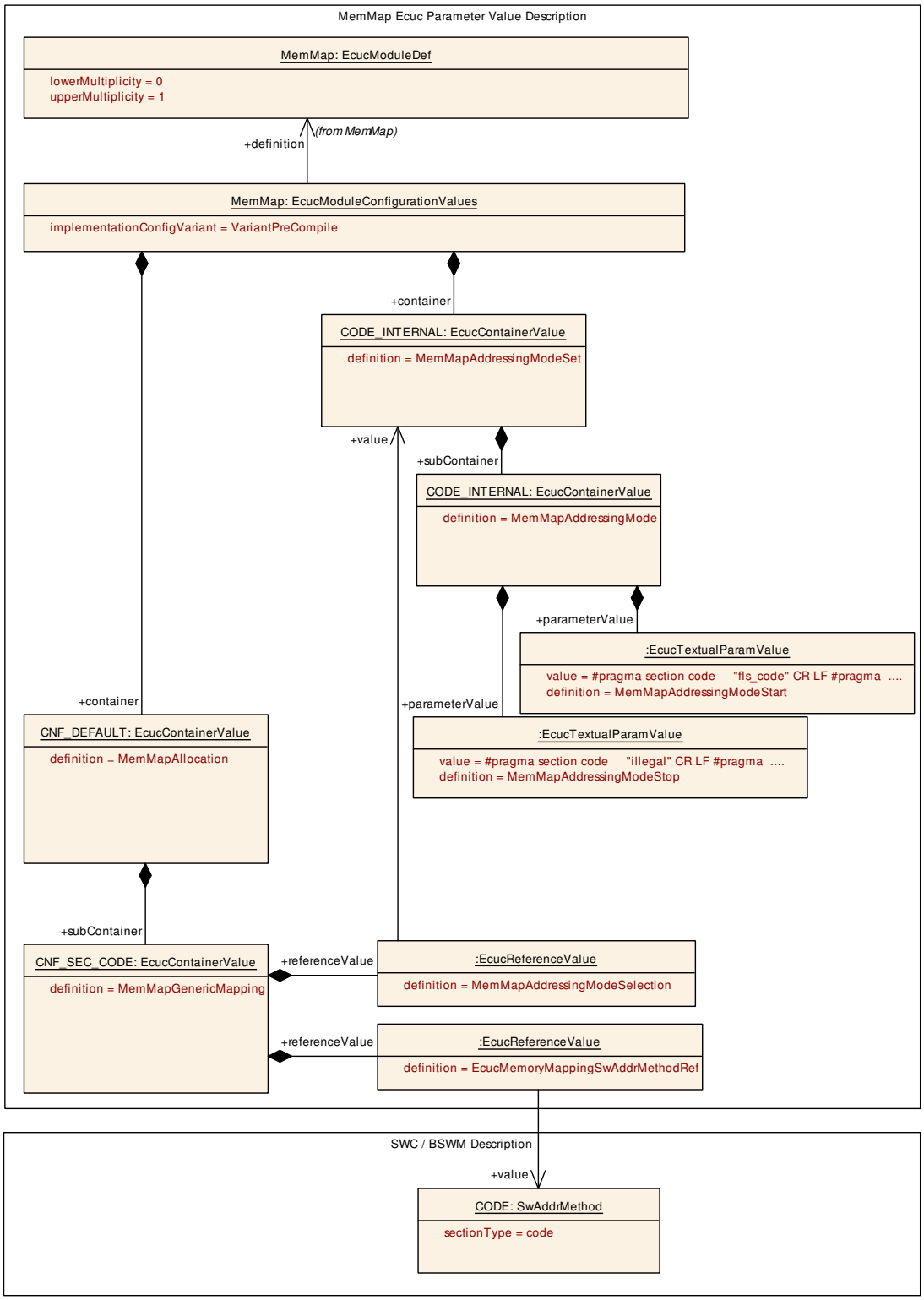


Figure 7.2: Example of `MemMap` configuration for a code section

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 the Memory Allocation Header for the Software Component shall implement the following MAKW:

- `MySwc_START_SEC_CODE`
- `MySwc_STOP_SEC_CODE`

7.4.2 Fast Variable Section

The following example shows `ApplicationSwComponentType` "MySwc" which contains in its `SwcInternalBehavior` two `VariableDataPrototype`s "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 `VariableDataPrototype`s 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 `VariableDataPrototype`s have to be sorted according their size into different `MemorySection`s.

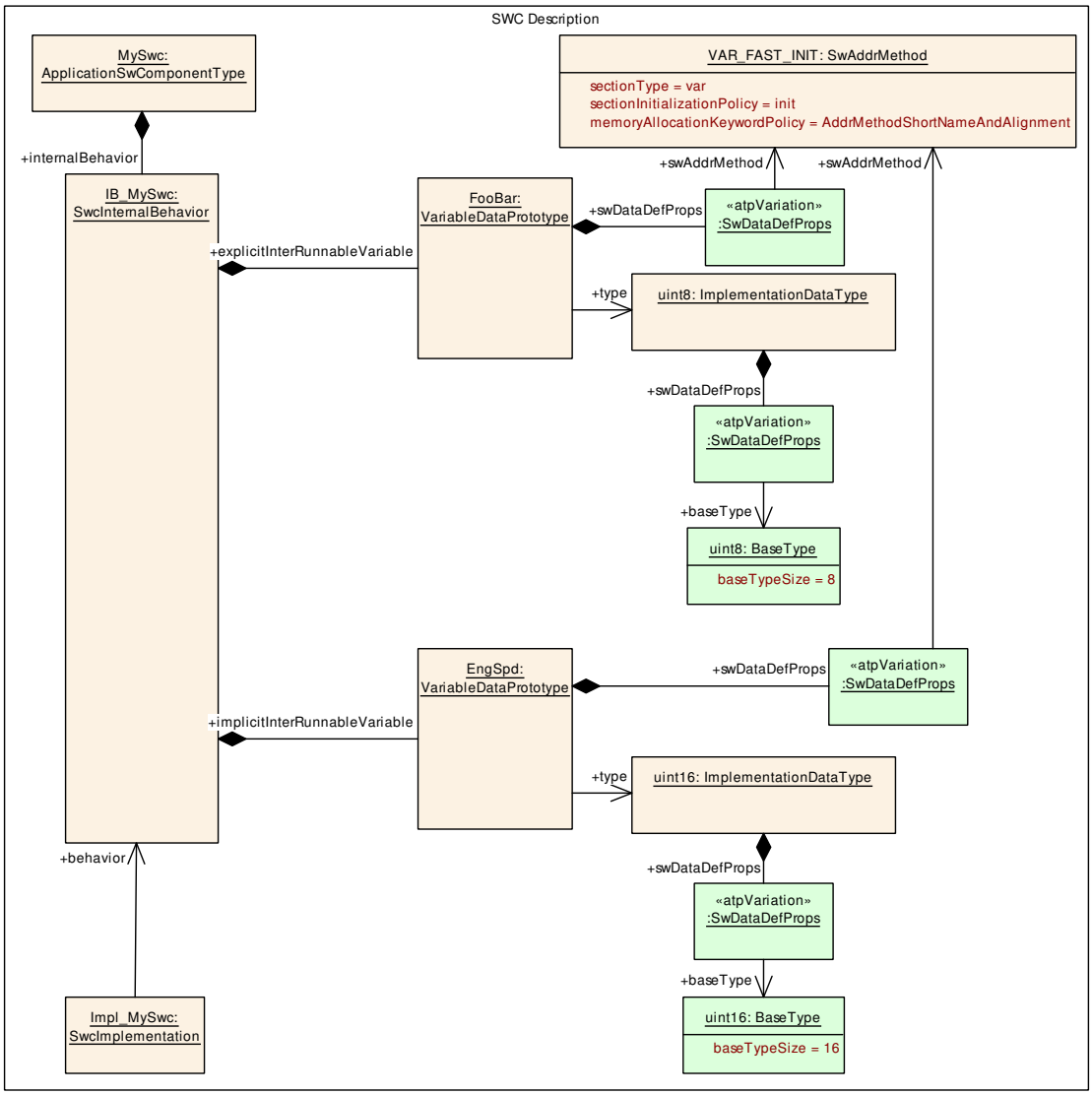


Figure 7.3: Example of `ApplicationSwComponentType` with `VariableDataPrototype`s

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.

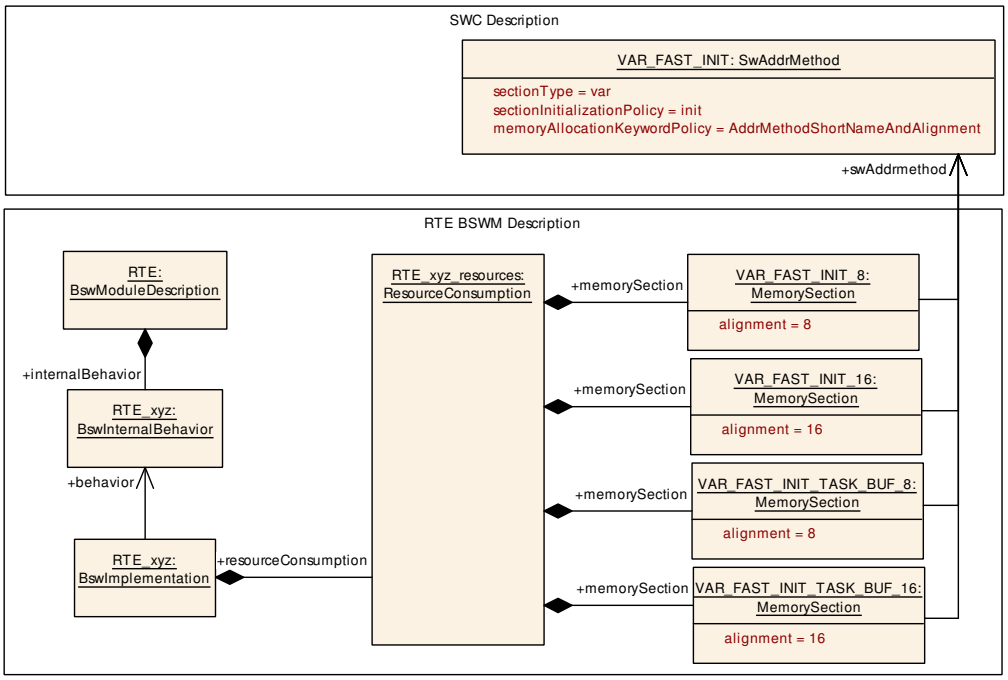


Figure 7.4: Example of Basic Software Module Description of RTE

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".

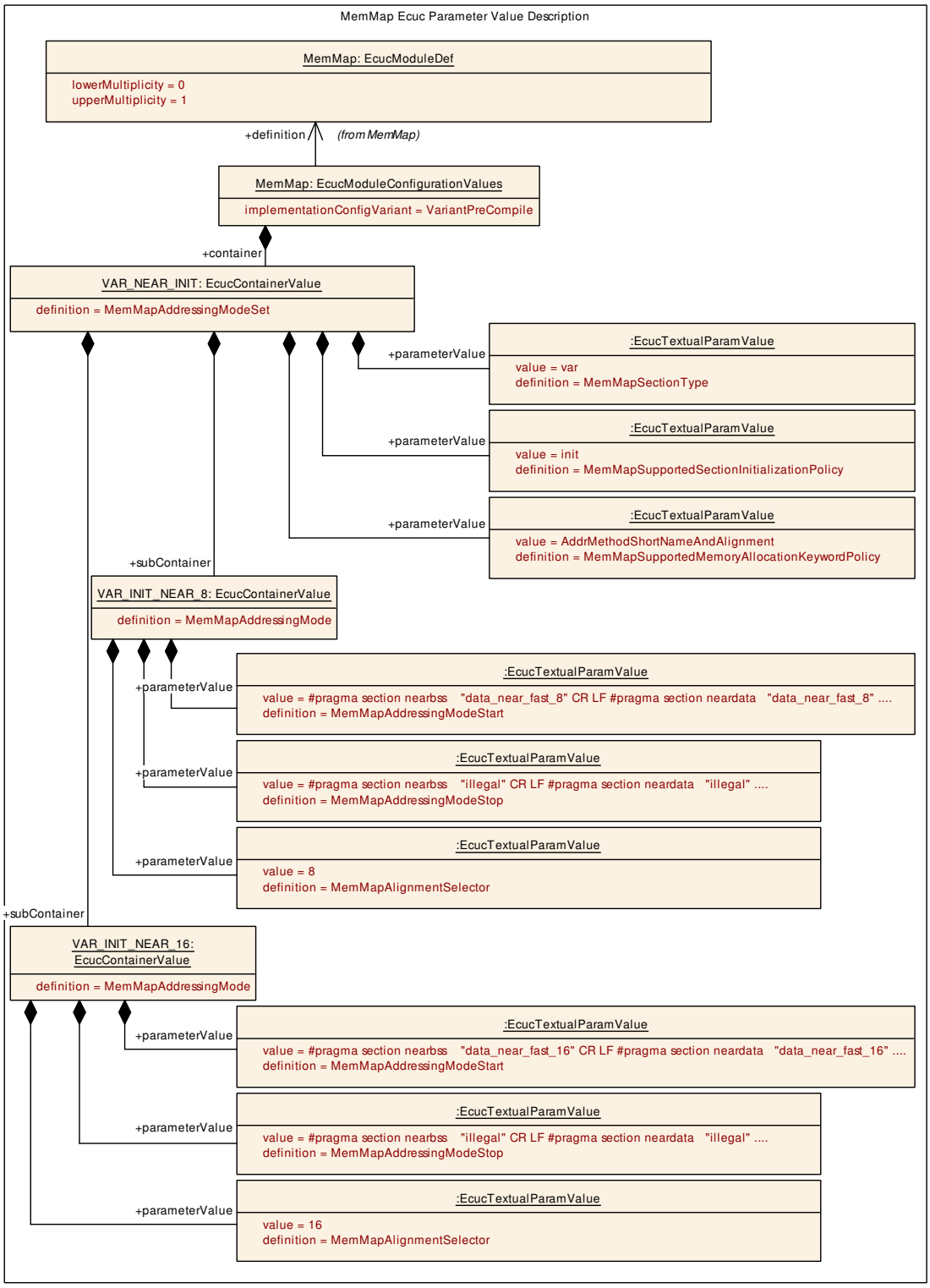


Figure 7.5: Example of MemMap configuration for a data section

The ECU Configuration of Memory Mapping defines a MemMapAddressing-ModeSet "VAR_NEAR_INIT" This supports the sectionType = var, section-InitializationPolicy = "INIT" and memoryAllocationKeywordPolicy = addrMethodShortNameAndAlignment. In this example MemMapAddressing-

Modes are shown for the alignment 8 and 16 (MemMapAlignmentSelector = 8 and MemMapAlignmentSelector = 16).

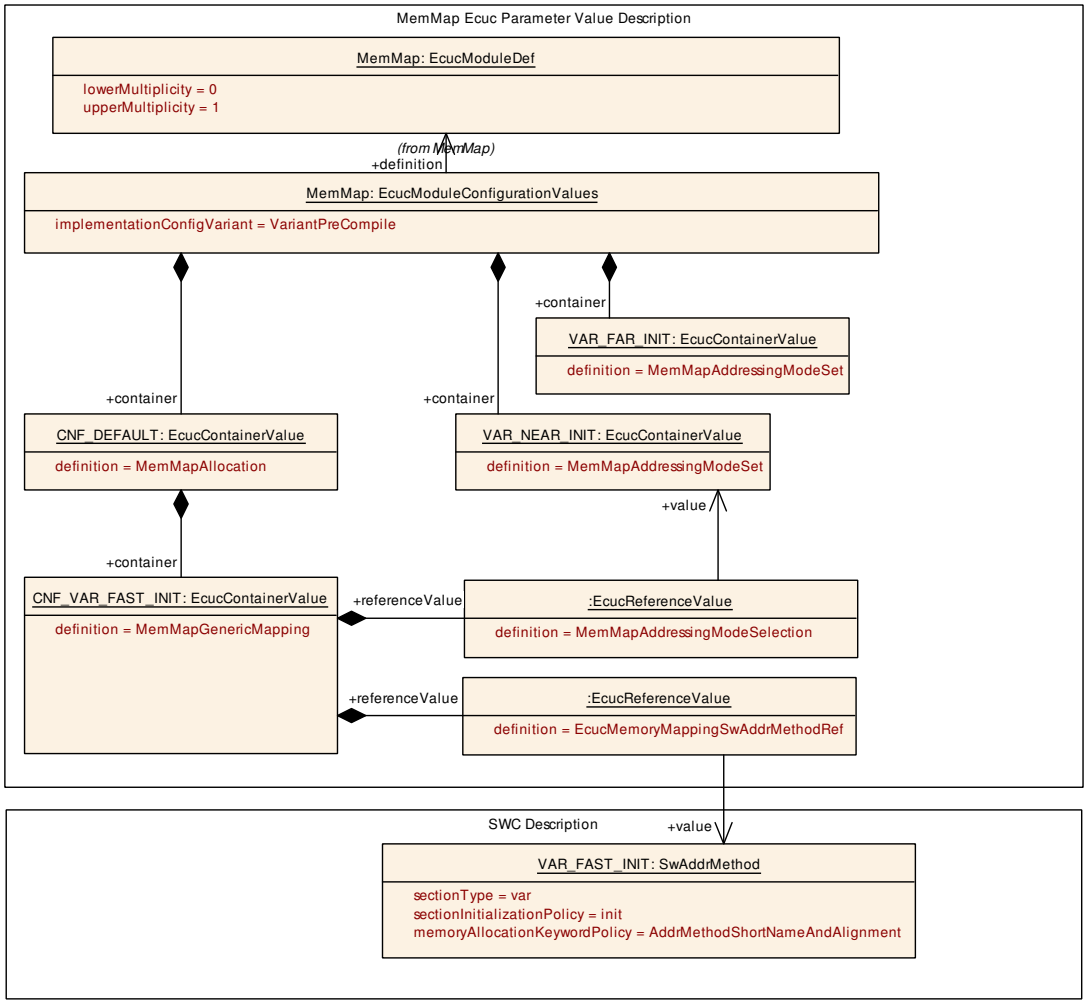


Figure 7.6: Example of MemMap configuration for a MemMapGenericMapping

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 the Memory Allocation Header for the BSW shall implement the following MAKW:

- RTE_START_SEC_VAR_FAST_INIT_8
- RTE_STOP_SEC_VAR_FAST_INIT_8
- RTE_START_SEC_VAR_FAST_INIT_16

- RTE_STOP_SEC_VAR_FAST_INIT_16
- RTE_START_SEC_VAR_FAST_INIT_TASK_BUF_8
- RTE_STOP_SEC_VAR_FAST_INIT_TASK_BUF_8
- RTE_START_SEC_VAR_FAST_INIT_TASK_BUF_16
- RTE_STOP_SEC_VAR_FAST_INIT_TASK_BUF_16

7.4.3 Code Section in ICC2 cluster

The following examples shows a Basic Software Module description of a Code Section in ICC2 cluster:

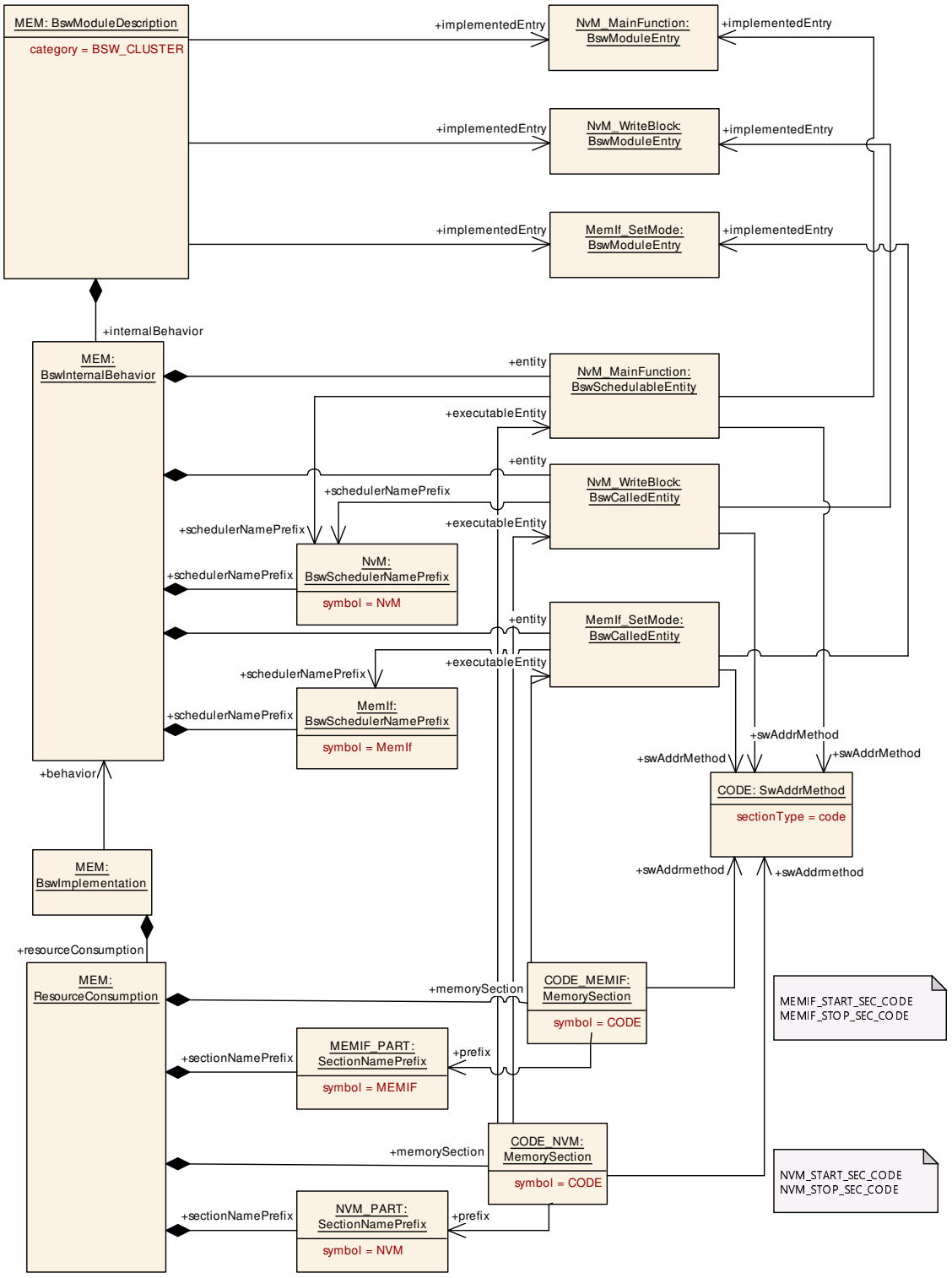


Figure 7.7: Example of BSW Module Description of an ICC2 cluster

With this information the Memory Allocation Header shall implement the following MAKW:

- MEMIF_START_SEC_CODE
- MEMIF_STOP_SEC_CODE

- NVM_START_SEC_CODE
- NVM_STOP_SEC_CODE

7.4.4 Callout sections

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

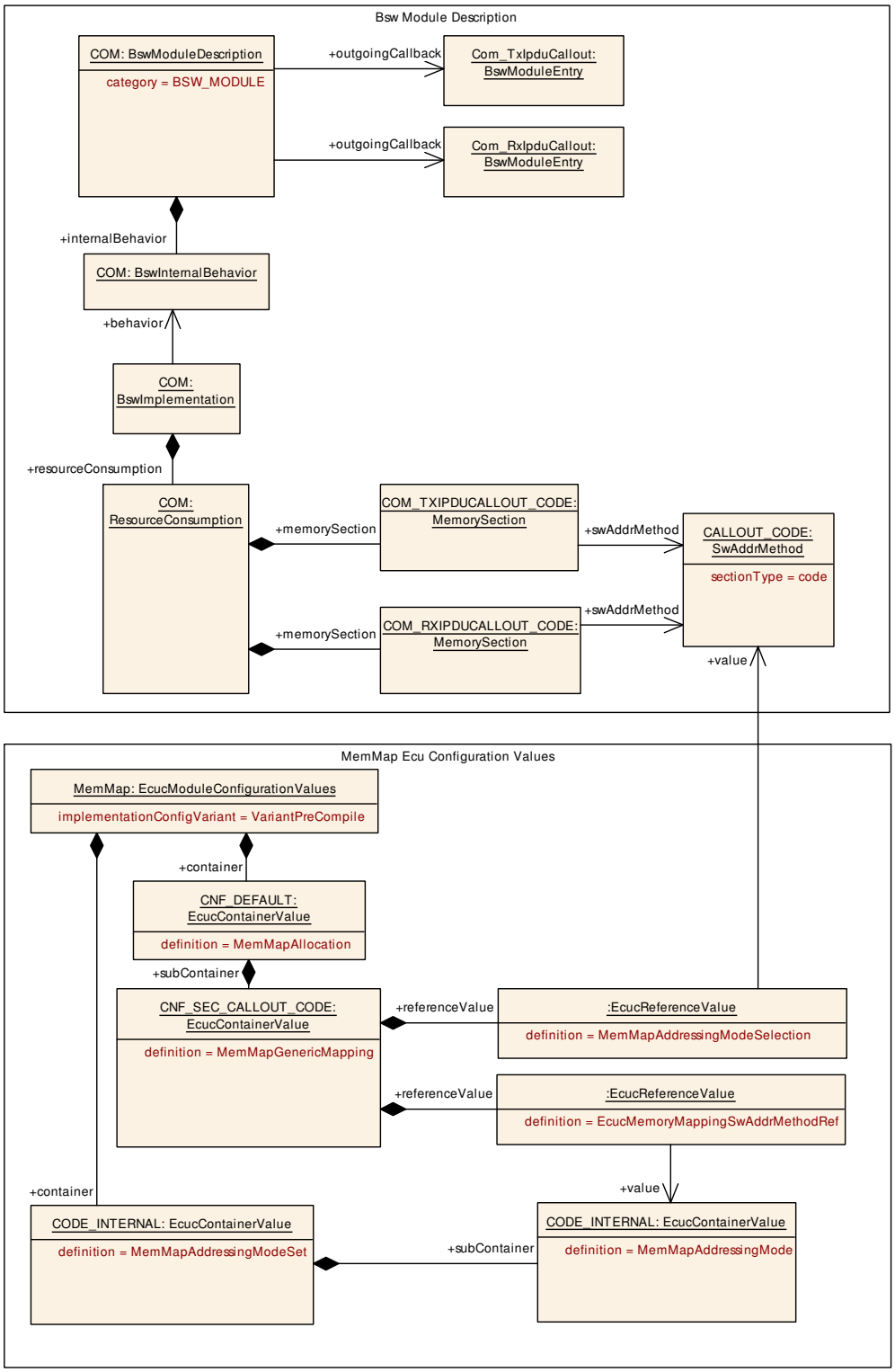


Figure 7.8: Example of description and configuration for callout code

With this information the Memory Allocation Header shall implement the following MAKW. These are build according to `SEC_CALLOUT_CODE_...` which is derived from BswModuleEntry.ShortName defined on Figure 7.8:

- `COM_START_SEC_CALLOUT_CODE_COM_RXIPDUCALLOUT`

- COM_STOP_SEC_CALLOUT_CODE_COM_RXIPDUCALLOUT
- COM_START_SEC_CALLOUT_CODE_COM_TXIPDUCALLOUT
- COM_STOP_SEC_CALLOUT_CODE_COM_TXIPDUCALLOUT

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

7.4.5 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.

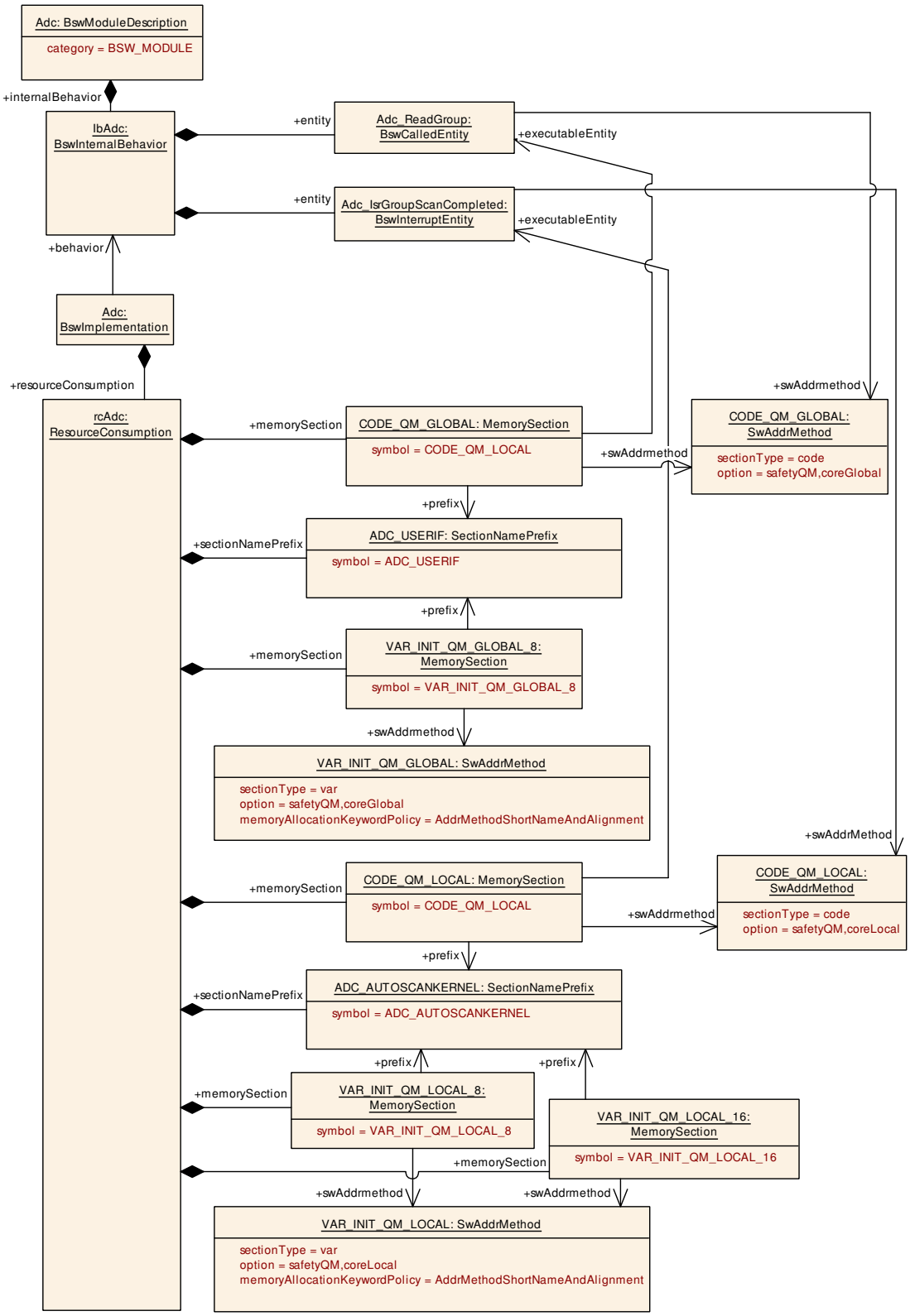


Figure 7.9: Example of description and configuration for allocatable memory parts

The shown configuration would result in the support of following Memory Allocation Keywords in the `Adc_MemMap.h` header file:

- ADC_AUTOSCANKERNEL_START_SEC_CODE_QM_LOCAL
- ADC_AUTOSCANKERNEL_STOP_SEC_CODE_QM_LOCAL
- ADC_AUTOSCANKERNEL_START_SEC_VAR_INIT_QM_LOCAL_8
- ADC_AUTOSCANKERNEL_STOP_SEC_VAR_INIT_QM_LOCAL_8
- ADC_AUTOSCANKERNEL_START_SEC_VAR_INIT_QM_LOCAL_16
- ADC_AUTOSCANKERNEL_STOP_SEC_VAR_INIT_QM_LOCAL_16
- ADC_USERIF_START_SEC_CODE_QM_GLOBAL
- ADC_USERIF_STOP_SEC_CODE_QM_GLOBAL
- ADC_USERIF_START_SEC_VAR_INIT_QM_GLOBAL_8
- ADC_USERIF_STOP_SEC_VAR_INIT_QM_GLOBAL_8

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

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

Chapter 10.3 specifies published information of the module MemMap.

10.1 How to read this chapter

For details refer to [2] Chapter 10.1 *"Introduction to configuration specification"*.

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

[ECUC_MemMap_00001] Definition of EcucModuleDef MemMap ⌈

Module Name	MemMap
Description	Configuration of the Memory Mapping module.
Post-Build Selectable Support	false
Post-Build Loadable Support	false

Included Containers		
Container Name	**Multiplicity**	**Description**
MemMapAddressingModeSet	0..*	Defines a set of addressing modes which might apply to a Sw AddrMethod.
MemMapAllocation	0..*	Defines which MemorySection of a BSW Module or a Software Component is implemented with which MemMapAddressing ModeSet. This can either be specified for a set of MemorySections which refer to an identical SwAddrMethod (MemMapGenericMapping) or for individual MemorySections (MemMapSectionSpecific Mapping). If both are defined for the same MemorySection the MemMapSectionSpecificMapping overrules the MemMap GenericMapping.

⌋

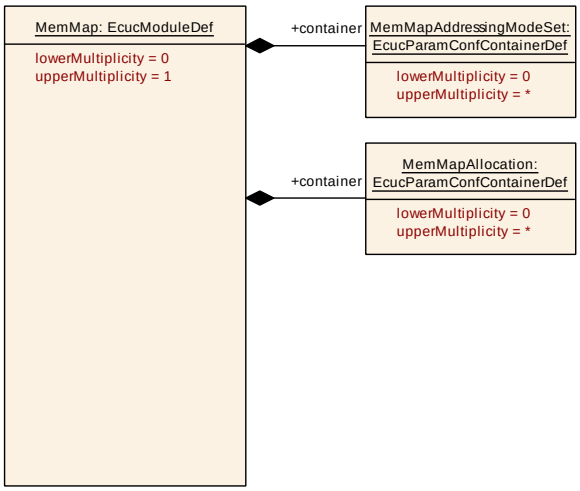


Figure 10.1: Overview about MemMap

10.2.2 MemMapAddressingModeSet

[ECUC_MemMap_00002] Definition of EcucParamConfContainerDef MemMapAddressingModeSet ⌈

Container Name	MemMapAddressingModeSet
Parent Container	MemMap
Upstream mapping defined	–
Description	Defines a set of addressing modes which might apply to a SwAddrMethod.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemMapSupportedAddressingMethodOption	0..*	[ECUC_MemMap_00009]
MemMapSupportedMemoryAllocationKeywordPolicy	0..*	[ECUC_MemMap_00017]
MemMapSupportedSectionInitializationPolicy	0..*	[ECUC_MemMap_00008]
MemMapSupportedSectionType	0..*	[ECUC_MemMap_00007]

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

⌋

[ECUC_MemMap_00009] Definition of EcucStringParamDef MemMapSupported AddressingMethodOption ⌈

Parameter Name	MemMapSupportedAddressingMethodOption
Parent Container	MemMapAddressingModeSet
Upstream mapping defined	–
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddr Methods. The attribute option of a swAddrMethod mapped via MemMapGenericMapping to this MemMapAddressingModeSet shall be equal to one of the configured MemMap SupportedAddressMethodOption's
Multiplicity	0..*
Type	EcucStringParamDef
Default value	–
Regular Expression	[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?
Pre-Compile only	true

⌋

[ECUC_MemMap_00017] Definition of EcucEnumerationParamDef MemMapSupportedMemoryAllocationKeywordPolicy ⌈

Parameter Name	MemMapSupportedMemoryAllocationKeywordPolicy	
Parent Container	MemMapAddressingModeSet	
Upstream mapping defined	–	
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddr Methods. The attribute MemoryAllocationKeywordPolicy of a swAddrMethod mapped via Mem MapGenericMapping 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 Bsw SchedulableEntitys.
Pre-Compile only	true	

⌋

[ECUC_MemMap_00008] Definition of EcucStringParamDef MemMapSupported SectionInitializationPolicy ⌈

Parameter Name	MemMapSupportedSectionInitializationPolicy
Parent Container	MemMapAddressingModeSet
Upstream mapping defined	–

▽

△

Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddr Methods. The sectionIntializationPolicy attribute value of a swAddrMethod mapped via MemMap GenericMapping 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 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	–
Pre-Compile only	true

⌋

[ECUC_MemMap_00007] Definition of EcucEnumerationParamDef MemMapSupportedSectionType ⌈

Parameter Name	MemMapSupportedSectionType	
Parent Container	MemMapAddressingModeSet	
Upstream mapping defined	–	
Description	This constrains the usage of this addressing mode set for Generic Mappings to swAddr Methods. 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.

▽

△

Pre-Compile only	true

⌋

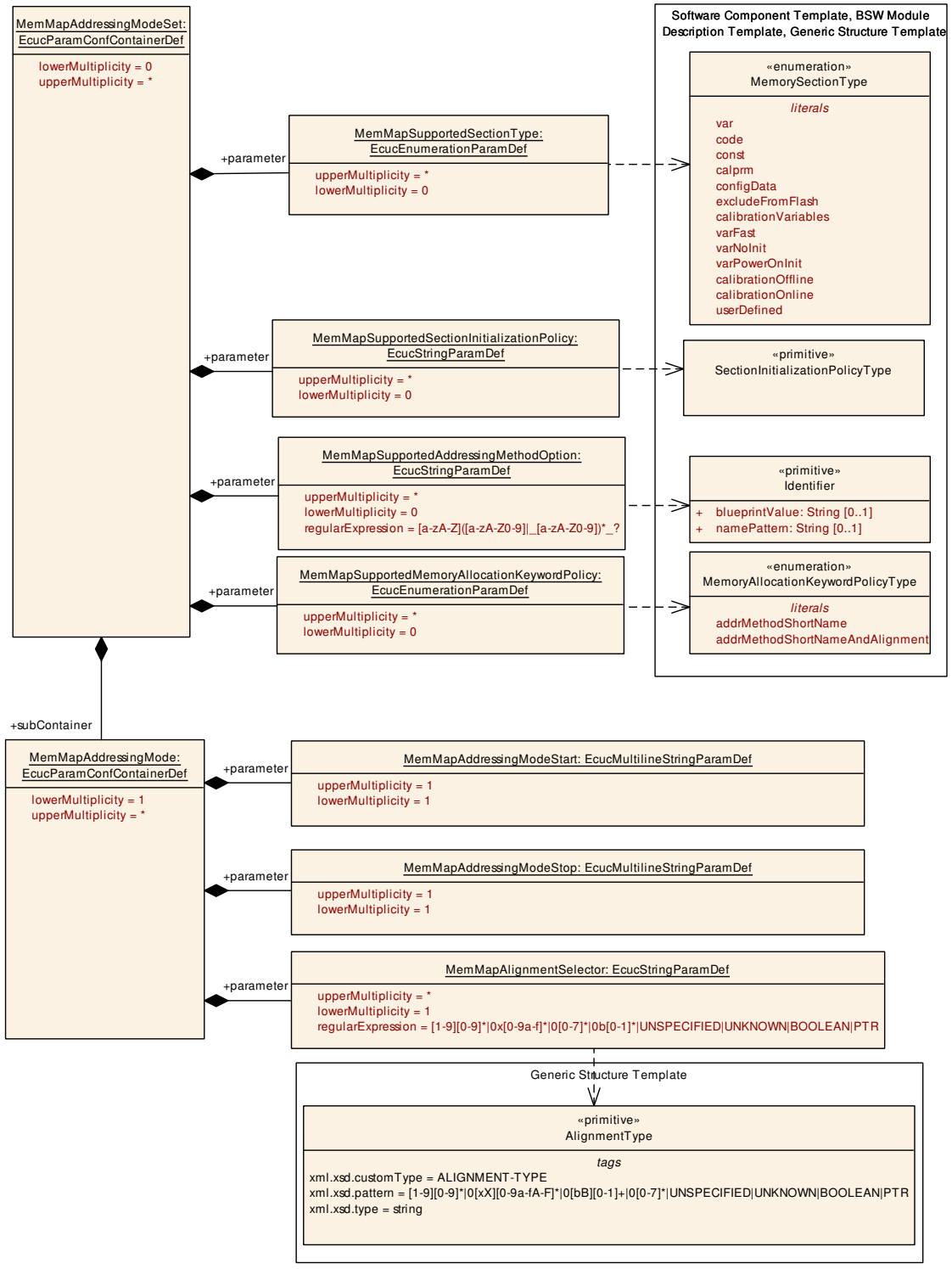


Figure 10.2: Overview about MemMapAddressingModeSet

10.2.3 MemMapAddressingMode

[ECUC_MemMap_00003] Definition of EcucParamConfContainerDef MemMapAddressingMode ⌈

Container Name	MemMapAddressingMode
Parent Container	MemMapAddressingModeSet
Upstream mapping defined	–
Description	Defines a addressing mode with a set of #pragma statements implementing the start and the stop of a section.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemMapAddressingModeStart	1	[ECUC_MemMap_00004]
MemMapAddressingModeStop	1	[ECUC_MemMap_00005]
MemMapAlignmentSelector	1..*	[ECUC_MemMap_00006]

No Included Containers

⌋

[ECUC_MemMap_00004] Definition of EcucMultilineStringParamDef MemMapAddressingModeStart ⌈

Parameter Name	MemMapAddressingModeStart
Parent Container	MemMapAddressingMode
Upstream mapping defined	–
Description	Defines a set of #pragma statements implementing the start of a section.
Multiplicity	1
Type	EcucMultilineStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_MemMap_00005] Definition of EcucMultilineStringParamDef MemMapAddressingModeStop ⌈

Parameter Name	MemMapAddressingModeStop
Parent Container	MemMapAddressingMode
Upstream mapping defined	–
Description	Defines a set of #pragma statements implementing the start of a section.
Multiplicity	1
Type	EcucMultilineStringParamDef

▽

△

Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_MemMap_00006] Definition of EcucStringParamDef MemMapAlignment Selector ⌈

Parameter Name	MemMapAlignmentSelector
Parent Container	MemMapAddressingMode
Upstream mapping defined	–
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 Mem MapAlignmentSelector 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
Pre-Compile only	true

⌋

10.2.4 MemMapAllocation

[ECUC_MemMap_00010] Definition of EcucParamConfContainerDef MemMapAllocation ⌈

Container Name	MemMapAllocation
Parent Container	MemMap
Upstream mapping defined	–
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 Sw AddrMethod (MemMapGenericMapping) or for individual MemorySections (MemMap SectionSpecificMapping). If both are defined for the same MemorySection the Mem MapSectionSpecificMapping overrules the MemMapGenericMapping.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
MemMapGenericMapping	0..*	Defines which SwAddrMethod is implemented with which Mem MapAddressingModeSet. The pragmas for the implementation of the MemorySelector Keywords are taken from the MemMapAddressingModeStart and MemMapAddressingModeStop parameters of the MemMap AddressingModeSet for the individual alignments. That this mapping becomes valid requires matching MemMap SupportedSectionType's, MemMapSupportedSection InitializationPolicy's and MemMapSupportedAddressingMethod Option's. The MemMapGenericMapping applies only if it is not overruled by an MemMapSectionSpecificMapping
MemMapMappingSelector	0..*	The container holds a section criteria reusable for MemMap GenericMappings.
MemMapSectionSpecificMapping	0..*	Defines which MemorySection of a BSW Module or a Software Component is implemented with which MemMapAddressing ModeSet. The pragmas for the implementation of the MemorySelector Keywords are taken from the MemMapAddressingModeStart and MemMapAddressingModeStop parameters of the MemMap AddressingModeSet for the specific alignment of the Memory Section. The MemMapSectionSpecificMapping precedes a mapping defined by MemMapGenericMapping.

⌋

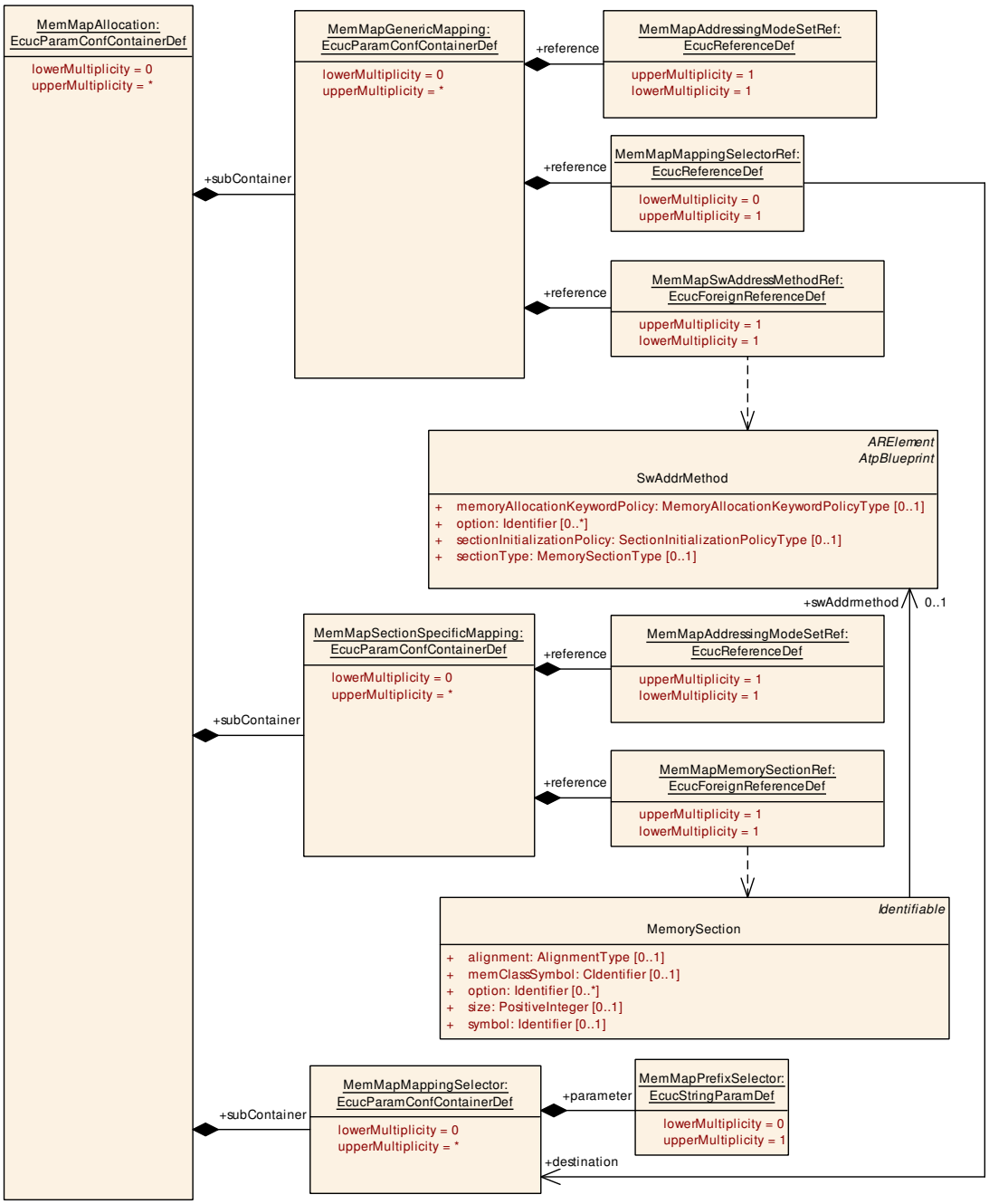


Figure 10.3: Overview about MemMapAllocation

10.2.5 MemMapGenericMapping

[ECUC_MemMap_00011] Definition of EcucParamConfContainerDef MemMap GenericMapping ⌈

Container Name	MemMapGenericMapping
Parent Container	MemMapAllocation
Upstream mapping defined	–
Description	Defines which SwAddrMethod is implemented with which MemMapAddressingMode Set. 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 MemMapSupportedAddressing MethodOption's. The MemMapGenericMapping applies only if it is not overruled by an MemMapSection SpecificMapping
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemMapAddressingModeSetRef	1	[ECUC_MemMap_00012]
MemMapMappingSelectorRef	0..1	[ECUC_MemMap_00023]
MemMapSwAddressMethodRef	1	[ECUC_MemMap_00013]

No Included Containers

⌋

[ECUC_MemMap_00012] Definition of EcucReferenceDef MemMapAddressing ModeSetRef ⌈

Parameter Name	MemMapAddressingModeSetRef
Parent Container	MemMapGenericMapping
Upstream mapping defined	–
Description	Reference to the MemMapAddressingModeSet which applies to the MemMapGeneric Mapping.
Multiplicity	1
Type	Reference to MemMapAddressingModeSet
Pre-Compile only	true

⌋

[ECUC_MemMap_00023] Definition of EcucReferenceDef MemMapMappingSelectorRef ⌈

Parameter Name	MemMapMappingSelectorRef
Parent Container	MemMapGenericMapping
Upstream mapping defined	–
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
Pre-Compile only	true

⌋

[ECUC_MemMap_00013] Definition of EcucForeignReferenceDef MemMapSwAddressMethodRef ⌈

Parameter Name	MemMapSwAddressMethodRef
Parent Container	MemMapGenericMapping
Upstream mapping defined	[4, CP_TPS_SoftwareComponentTemplate]
Description	Reference to the SwAddrMethod which applies to the MemMapGenericMapping.
Multiplicity	1
Type	Foreign reference to SW-ADDR-METHOD
Pre-Compile only	true

⌋

10.2.6 MemMapSectionSpecificMapping

[ECUC_MemMap_00014] Definition of EcucParamConfContainerDef MemMapSectionSpecificMapping ⌈

Container Name	MemMapSectionSpecificMapping
Parent Container	MemMapAllocation
Upstream mapping defined	–
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 MemMap GenericMapping.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemMapAddressingModeSetRef	1	[ECUC_MemMap_00015]
MemMapMemorySectionRef	1	[ECUC_MemMap_00016]

No Included Containers

⌋

[ECUC_MemMap_00015] Definition of EcucReferenceDef MemMapAddressingModeSetRef ⌈

Parameter Name	MemMapAddressingModeSetRef
Parent Container	MemMapSectionSpecificMapping
Upstream mapping defined	–
Description	Reference to the MemMapAddressingModeSet which applies to the MemMapModule SectionSpecificMapping.
Multiplicity	1
Type	Reference to MemMapAddressingModeSet
Pre-Compile only	true

⌋

[ECUC_MemMap_00016] Definition of EcucForeignReferenceDef MemMapMemorySectionRef ⌈

Parameter Name	MemMapMemorySectionRef
Parent Container	MemMapSectionSpecificMapping
Upstream mapping defined	[4, CP_TPS_SoftwareComponentTemplate]
Description	Reference to the MemorySection which applies to the MemMapSectionSpecific Mapping.

▽

△

Multiplicity	1
Type	Foreign reference to MEMORY-SECTION
Pre-Compile only	true

⌋

10.2.7 MemMapMappingSelector

[ECUC_MemMap_00021] Definition of EcucParamConfContainerDef MemMapMappingSelector ⌈

Container Name	MemMapMappingSelector
Parent Container	MemMapAllocation
Upstream mapping defined	–
Description	The container holds a section criteria reusable for MemMapGenericMappings.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
MemMapPrefixSelector	0..1	[ECUC_MemMap_00022]

No Included Containers

⌋

[ECUC_MemMap_00022] Definition of EcucStringParamDef MemMapPrefixSelector ⌈

Parameter Name	MemMapPrefixSelector
Parent Container	MemMapMappingSelector
Upstream mapping defined	–
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 MemMapAddressing ModeSet 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	–
Pre-Compile only	true

⌋

10.3 Published Information

For details refer to [2] Chapter 10.3 *"Published Information"*.

A Appendix

A.1 Referenced Meta Classes

Class	**ApplicationSwComponentType**			
Note	The `ApplicationSwComponentType` is used to represent the application software. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Collectable Element*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: ApplicationSwComponentType

Class	***AtomicSwComponentType*** (abstract)			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Subclasses	ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponent Type, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponent Type, ServiceSwComponentType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
internalBehavior	SwcInternalBehavior	0..1	aggr	The `SwcInternalBehavior`s owned by an `AtomicSwComponentType` can be located in a different physical file. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=internalBehavior, internalBehavior.short Name, internalBehavior.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the `SymbolProps` for the `AtomicSwComponentType`. **Tags:** atp.Splitkey=symbolProps, symbolProps.short Name

Table A.2: AtomicSwComponentType

Class	**BaseTypeDirectDefinition**			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	*ARObject*, *BaseTypeDefinition*			
Aggregated by	*BaseType*.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

▽

△

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

Table A.3: BaseTypeDirectDefinition

Class	**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 This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arRelease Version	RevisionLabelString	0..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.

▽

△

Class	**BswImplementation**			
behavior	BswInternalBehavior	0..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 split, but we want supply the implementation later, the BswImplementation is not aggregated in BswBehavior **Tags:** atp.Splitkey=behavior
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:** atp.Splitkey=preconfiguredConfiguration xml.roleWrapperElement=true
recommended Configuration	EcucModule ConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster. **Tags:** atp.Splitkey=recommendedConfiguration
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 Bsw Implementation 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:** atp.Splitkey=vendorSpecificModuleDef xml.roleWrapperElement=true

Table A.4: BswImplementation

Class	**BswModuleDescription**			
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 This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bswModule Dependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=bswModuleDependency, bswModule Dependency.shortName, bswModule Dependency.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
bswModule Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=bswModuleDocumentation, bswModule Documentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=6
expectedEntry	BswModuleEntry	*	ref	Indicates an entry which is required by this module. Replacement of outgoingCallback / requiredEntry. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=expectedEntry, expectedEntry.bswModule Entry, expectedEntry.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:** atpVariation **Tags:** atp.Splitkey=implementedEntry, implementedEntry.bsw ModuleEntry, 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>>. **Tags:** atp.Splitkey=internalBehavior, internalBehavior.short Name 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



△

Class	**BswModuleDescription**			
providedClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module provides a client server entry which can be called from another partition 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:** atpVariation **Tags:** atp.Splitkey=providedClientServerEntry, providedClient ServerEntry.shortName, providedClientServer Entry.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:** atpVariation **Tags:** atp.Splitkey=providedData, providedData.shortName, providedData.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
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:** atpVariation **Tags:** atp.Splitkey=providedModeGroup, providedMode Group.shortName, providedModeGroup.variation Point.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:** atpVariation **Tags:** atp.Splitkey=releasedTrigger, releasedTrigger.shortName, releasedTrigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=35

▽

△

Class	**BswModuleDescription**			
requiredClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module requires a client server entry which can be implemented on another partition 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:** atpVariation **Tags:** atp.Splitkey=requiredClientServerEntry, requiredClient ServerEntry.shortName, requiredClientServer Entry.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:** atpVariation **Tags:** atp.Splitkey=requiredData, requiredData.shortName, requiredData.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
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:** atpVariation **Tags:** atp.Splitkey=requiredModeGroup, requiredMode Group.shortName, requiredModeGroup.variation Point.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:** atpVariation **Tags:** atp.Splitkey=requiredTrigger, requiredTrigger.shortName, requiredTrigger.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Table A.5: BswModuleDescription

Class	**DependencyOnArtifact**			
Note	Dependency on the existence of another artifact, e.g. a library.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*Implementation*.generatedArtifact, *Implementation*.requiredArtifact, *Implementation*.requiredGenerator Tool			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**DependencyOnArtifact**			
artifact Descriptor	AutosarEngineering Object	0..1	aggr	The specified artifact needs to exist. **Tags:** atp.Splitkey=artifactDescriptor
usage	DependencyUsage Enum	*	attr	Specification for which process step(s) this dependency is required.

Table A.6: DependencyOnArtifact

Class	**EcucModuleConfigurationValues**			
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 This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
container	EcucContainerValue	*	aggr	Aggregates all containers that belong to this module configuration. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=container, container.shortName, 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. **Tags:** atp.Splitkey=definition 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. **Tags:** atp.Status=obsolete

▽

△

Class	**EcucModuleConfigurationValues**			
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. **Tags:** atp.Splitkey=moduleDescription
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.7: EcucModuleConfigurationValues

Class	**EcucValueCollection**			
Note	This represents the anchor point of the ECU configuration description. **Tags:** atp.recommendedPackage=EcucValueCollections This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
ecucValue	EcucModule ConfigurationValues	*	ref	References to the configuration of individual software modules that are present on this ECU. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=ecucValue, ecucValue.ecucModule ConfigurationValues, ecucValue.variationPoint.shortLabel 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. **Tags:** atp.Splitkey=ecuExtract

Table A.8: EcucValueCollection

Class	***EngineeringObject*** (abstract)			
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***

▽

△

Class	***EngineeringObject*** (abstract)			
category	NameToken	0..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	0..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.9: EngineeringObject

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, AppliedStandard, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpClassifier*, *Atp Feature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifestAddressable Object*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResourceDefinition, Block State, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanSecSecureChannel, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClientIdDefinition, Client ServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, *CouplingElementAbstractDetails*, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPort StructuralElement*, *CpSoftwareClusterResource*, CpSoftwareClusterResourceToApplicationPartition Mapping, CpSoftwareClusterToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareClusterToResourceMapping, *CryptoServiceMapping*, CyclicHandlingComDataToOsTask ProxyMapping, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, DataTransformation, *Dds AbstractServiceInstanceElementCp*, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, Diagnostic ExtendedDataRecordElement, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltArgumentProps, DltLogChannel, Dlt Message, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, ECUMapping, *EOCExecutableEntity RefAbstract*, EcuPartition, EcuPartitionToCoreMapping, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, Ethernet WakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeature Relation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayTpConnection ▽

▽

△

Class	**Identifiable** (abstract)			
	△ Control, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, J1939Node, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, ModeSwitchSenderComSpecProps, NetworkEndpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, QueuedReceiverComSpecProps, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RteEventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventInSystemSeparation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobRequirement, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServerCallPoint*, ServerComSpecProps, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *StackUsage*, StaticSocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SwitchAsynchronousTrafficShaperGroupEntry, SwitchAtsInstanceEntry, SwitchFlowMeteringEntry, SwitchStreamFilterActionDestPortModification, SwitchStreamFilterEntry, SwitchStreamFilterRule, SwitchStreamGateEntry, SwitchStreamIdentification, SystemMapping, SystemSignalGroupToCommunicationResourceMapping, SystemSignalToCommunicationResourceMapping, TDCpSoftwareClusterMapping, TDCpSoftwareClusterResourceMapping, TcpOptionFilterList, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **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:** atp.Splitkey=annotation xml.sequenceOffset=-25
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	MultiLanguageOverviewParagraph	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:** atp.Splitkey=desc xml.sequenceOffset=-60

▽

△

Class	***Identifiable*** (abstract)			
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:** atp.Splitkey=introduction 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.10: Identifiable

Class	***Implementation*** (abstract)			
Note	Description of an implementation a single software component or module. This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	BswImplementation, SwcImplementation			
Aggregated by	ARPackage.element			
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:** atp.Splitkey=buildActionManifest, buildAction Manifest.buildActionManifest, buildAction Manifest.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	*	aggr	Specifies the provided implementation code. **Tags:** atp.Splitkey=codeDescriptor, codeDescriptor.short Name
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released **Tags:** atp.Splitkey=compiler, compiler.shortName

▽

△

Class	***Implementation*** (abstract)			
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:** atp.Splitkey=generatedArtifact, generatedArtifact.shortName, generatedArtifact.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation. **Tags:** atp.Splitkey=hwElement
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released. **Tags:** atp.Splitkey=linker, linker.shortName
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The measurement & calibration support data belonging to this implementation. **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 existence 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:** atp.Splitkey=requiredArtifact, requiredArtifact.shortName, requiredArtifact.variationPoint.shortLabel 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:** atp.Splitkey=requiredGeneratorTool, requiredGeneratorTool.shortName, requiredGeneratorTool.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
resource Consumption	ResourceConsumption	0..1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class. **Tags:** atp.Splitkey=resourceConsumption, 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. **Tags:** atp.Splitkey=swcBswMapping

▽

△

Class	*Implementation* (abstract)			
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.11: Implementation

Class	**ImplementationDataType**			
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*, *AtpClassifier*, *AtpType*, *Autosar DataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
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 `ImplementationDataTypeElement` is subject to variability with the purpose to support the conditional existence of elements inside a `ImplementationDataType` representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Tags:** atp.Splitkey=symbolProps, symbolProps.short Name
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.12: ImplementationDataType

Enumeration	**MemoryAllocationKeywordPolicyType**
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Aggregated by	SwAddrMethod.memoryAllocationKeywordPolicy
Literal	***Description***

▽

△

Enumeration	**MemoryAllocationKeywordPolicyType**
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.13: MemoryAllocationKeywordPolicyType

Class	**MemorySection**			
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 memory AllocationKeywordPolicy value of the referenced SwAddrMethod is set to addrMethodShortNameAnd Alignment 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*			
Aggregated by	ResourceConsumption.memorySection			
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. **Tags:** atp.Splitkey=executableEntity

▽

△

Class	**MemorySection**			
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. **Tags:** atp.Splitkey=prefix
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. **Tags:** atp.Splitkey=swAddrmethod
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.14: MemorySection

Enumeration	**MemorySectionType**
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.
Aggregated by	SwAddrMethod.sectionType
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
const	To be used for global or static constants. **Tags:** atp.EnumerationLiteralIndex=6

▽

△

Enumeration	**MemorySectionType**
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.15: MemorySectionType

Class	***Referrable*** (abstract)			
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, 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
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** atp.Splitkey=shortNameFragment xml.sequenceOffset=-90

Table A.16: Referrable

Class	**RunnableEntity**			
Note	A `RunnableEntity` represents the smallest code-fragment that is provided by an `AtomicSwComponentType` and are executed under control of the RTE. `RunnableEntity`s 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*			
Aggregated by	*AtpClassifier*.atpFeature, SwcInternalBehavior.runnable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	RunnableEntityArgument	*	aggr	This represents the formal definition of 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:** atpVariation **Tags:** atp.Splitkey=asynchronousServerCallResultPoint, asynchronousServerCallResultPoint.shortName, asynchronousServerCallResultPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
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 `AtomicSwComponentType`). 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:** atpVariation **Tags:** atp.Splitkey=dataReadAccess, 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:** atpVariation **Tags:** atp.Splitkey=dataReceivePointByArgument, 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:** atpVariation **Tags:** atp.Splitkey=dataReceivePointByValue, dataReceivePoint ByValue.shortName, dataReceivePointByValue.variation Point.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:** atpVariation **Tags:** atp.Splitkey=dataSendPoint, dataSendPoint.shortName, dataSendPoint.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:** atpVariation **Tags:** atp.Splitkey=dataWriteAccess, dataWriteAccess.short Name, dataWriteAccess.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:** atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint, externalTriggering Point.ident.shortName, externalTriggeringPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
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:** atpVariation **Tags:** atp.Splitkey=internalTriggeringPoint, internalTriggering Point.shortName, internalTriggeringPoint.variation Point.shortLabel vh.latestBindingTime=preCompileTime
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:** atpVariation **Tags:** atp.Splitkey=modeAccessPoint, modeAccess Point.ident.shortName, modeAccessPoint.variation Point.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:** atpVariation **Tags:** atp.Splitkey=modeSwitchPoint, modeSwitchPoint.short Name, modeSwitchPoint.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:** atpVariation **Tags:** atp.Splitkey=parameterAccess, parameterAccess.short Name, parameterAccess.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**RunnableEntity**			
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:** atpVariation **Tags:** atp.Splitkey=readLocalVariable, readLocalVariable.short Name, readLocalVariable.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:** atpVariation **Tags:** atp.Splitkey=serverCallPoint, serverCallPoint.shortName, serverCallPoint.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
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`. **Tags:** atp.Splitkey=waitPoint, waitPoint.shortName
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:** atpVariation **Tags:** atp.Splitkey=writtenLocalVariable, writtenLocal Variable.shortName, writtenLocalVariable.variation Point.shortLabel vh.latestBindingTime=preCompileTime

Table A.17: RunnableEntity

Class	**SectionNamePrefix**			
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*			
Aggregated by	ResourceConsumption.sectionNamePrefix			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	SectionNamePrefix			
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). **Tags:** atp.Splitkey=implementedIn

Table A.18: SectionNamePrefix

Class	SwAddrMethod			
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*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
memory Allocation KeywordPolicy	MemoryAllocation KeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.
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 addressing method.

Table A.19: SwAddrMethod

Class	SwBaseType			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *BaseType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.20: SwBaseType

Class	**SwComponentType** (abstract)			
Note	Base class for AUTOSAR software components.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AtomicSwComponentType*, CompositionSwComponentType, ParameterSwComponentType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
consistency Needs	ConsistencyNeeds	*	aggr	This represents the collection of `ConsistencyNeeds` owned by the enclosing `SwComponentType`. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=consistencyNeeds, consistencyNeeds.shortName, consistencyNeeds.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
port	PortPrototype	*	aggr	The `PortPrototype`s through which this `SwComponentType` can communicate. The aggregation of `PortPrototype` is subject to variability with the purpose to support the conditional existence of `PortPrototype`s. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=port, port.shortName, port.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=portGroup, portGroup.shortName, portGroup.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
swcMapping Constraint	SwComponentMapping Constraints	*	ref	Reference to constraints that are valid for this SwComponentType. **Tags:** atp.Splitkey=swcMappingConstraint This Attribute is only used by the AUTOSAR Classic Platform.
swComponent Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the `SwComponentType`. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swComponentDocumentation, swComponentDocumentation.variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=-10
unitGroup	UnitGroup	*	ref	This allows for the specification of which `UnitGroup`s are relevant in the context of referencing `SwComponentType`. **Tags:** atp.Splitkey=unitGroup This Attribute is only used by the AUTOSAR Classic Platform.

Table A.21: SwComponentType

Class	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 This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
behavior	SwcInternalBehavior	0..1	ref	The internal behavior implemented by this Implementation. **Tags:** atp.Splitkey=behavior
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:** atp.Splitkey=perInstanceMemorySize, perInstance MemorySize.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
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.22: SwcImplementation

Class	SwcInternalBehavior			
Note	The `SwcInternalBehavior` of an `AtomicSwComponentType` describes the relevant aspects of the software-component with respect to the RTE, i.e. the `RunnableEntity`s and the `RTEEvent`s they respond to.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtomicSwComponentType*.internalBehavior, *AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**SwcInternalBehavior**			
arTypedPer Instance Memory	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:** atpVariation **Tags:** atp.Splitkey=arTypedPerInstanceMemory, arTypedPer InstanceMemory.shortName, arTypedPerInstance Memory.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 `RTEEvent`s. Note: the number of `RTEEvent`s might vary due to the conditional existence of `PortPrototype`s using `DataReceivedEvent`s or due to different scheduling needs of algorithms. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=event, event.shortName, event.variation Point.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea Policy	SwcExclusiveArea Policy	*	aggr	Options how to generate the ExclusiveArea related APIs. When no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, exclusiveArea Policy.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
explicitInter RunnableVariable	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:** atpVariation **Tags:** atp.Splitkey=explicitInterRunnableVariable, explicitInter RunnableVariable.shortName, explicitInterRunnable Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
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:** atpVariation **Tags:** atp.Splitkey=implicitInterRunnableVariable, implicitInter RunnableVariable.shortName, implicitInterRunnable Variable.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The `includedDataTypeSet` is used by a software component for its implementation. **Tags:** atp.Splitkey=includedDataTypeSet
includedMode Declaration GroupSet	IncludedMode DeclarationGroupSet	*	aggr	This aggregation represents the included Mode DeclarationGroups **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:** atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, instantiationData DefProps.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
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:** atpVariation **Tags:** atp.Splitkey=perInstanceMemory, perInstance Memory.shortName, perInstanceMemory.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
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:** atpVariation **Tags:** atp.Splitkey=perInstanceParameter, perInstance Parameter.shortName, perInstanceParameter.variation Point.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:** atpVariation **Tags:** atp.Splitkey=portAPIOption, portAPIOption.port, port APIOption.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
runnable	RunnableEntity	*	aggr	This is a `RunnableEntity` specified for the particular `SwcInternalBehavior`. The aggregation of `RunnableEntity` is subject to variability with the purpose to support the conditional existence of `RunnableEntity`s. Note: the number of `RunnableEntity`s might vary due to the conditional existence of `PortPrototype`s using `DataReceivedEvent`s or due to different scheduling needs of algorithms. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=runnable, runnable.shortName, runnable.variationPoint.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 expert domain (e.g OBD expert for Obd related Service Needs) tools. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=serviceDependency, service Dependency.shortName, serviceDependency.variation Point.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
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:** atpVariation **Tags:** atp.Splitkey=sharedParameter, sharedParameter.short Name, sharedParameter.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. **Tags:** atp.Splitkey=variationPointProxy, variationPoint Proxy.shortName

Table A.23: SwcInternalBehavior

Class	**SwcToImplMapping**			
Note	Map instances of an AtomicSwComponentType to a specific Implementation. This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	SystemMapping.swImplMapping			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
component	SwComponent Prototype	*	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. **Tags:** atp.Splitkey=component, component.context Composition, component.contextComponent, component.targetComponent **InstanceRef implemented by:** ComponentInSystem InstanceRef
component Implementation	SwcImplementation	0..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. **Tags:** atp.Splitkey=componentImplementation

Table A.24: SwcToImplMapping

Class	**SystemMapping**
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*

▽

△

Class	**SystemMapping**			
Aggregated by	System.mapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
application PartitionToEcu Partition Mapping	ApplicationPartitionTo EcuPartitionMapping	*	aggr	Mapping of ApplicationPartitions to EcuPartitions **Stereotypes:** atpVariation **Tags:** atp.Splitkey=applicationPartitionToEcuPartitionMapping, applicationPartitionToEcuPartitionMapping.shortName, applicationPartitionToEcuPartitionMapping.variation Point.shortLabel vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.
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. **Tags:** atp.Splitkey=appOsTaskProxyToEcuTaskProxy Mapping, appOsTaskProxyToEcuTaskProxy Mapping.shortName This Attribute is only used by the AUTOSAR Classic Platform.
com Management Mapping	ComManagement Mapping	*	aggr	Mappings between Mode Management PortGroups and communication channels. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=comManagementMapping, comManagement Mapping.shortName, comManagementMapping.variation Point.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
cryptoService Mapping	CryptoServiceMapping	*	aggr	This aggregation represents the collection of crypto service mappings in the context of the enclosing System Mapping. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=cryptoServiceMapping, cryptoService Mapping.shortName, cryptoServiceMapping.variation Point.shortLabel vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.
cyclicHandling ComDataToOs TaskProxy Mapping	CyclicHandlingCom DataToOsTaskProxy Mapping	*	aggr	Mapping of VariableDataPrototypes to an OsTaskProxy for the Cyclic Handling of Communication Data in the RTE. **Tags:** atp.Splitkey=cyclicHandlingComDataToOsTask ProxyMapping, cyclicHandlingComDataToOsTaskProxy Mapping.shortName This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**SystemMapping**			
dataMapping	DataMapping	*	aggr	The data mappings defined. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataMapping, dataMapping.variation Point.shortLabel vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.
ddsISignalTo TopicMapping	DdsCpISignalToDds TopicMapping	*	aggr	Collection of DdsISignalToDdsTopicMappings. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=ddsISignalToTopicMapping, ddsISignalTo TopicMapping.variationPoint.shortLabel atp.Status=candidate vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.
ecuPartitionTo CoreMapping	EcuPartitionToCore Mapping	*	aggr	Mapping of EcuPartions to a Cores. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=ecuPartitionToCoreMapping, ecuPartitionTo CoreMapping.shortName, ecuPartitionToCore Mapping.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
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:** atp.Splitkey=ecuResourceMapping, ecuResource Mapping.shortName, ecuResourceMapping.variation Point.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
j1939Controller ApplicationTo J1939Node Mapping	J1939Controller ApplicationToJ1939 NodeMapping	*	aggr	Mapping of a J1939ControllerApplication to a J1939Node. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=j1939ControllerApplicationToJ1939Node Mapping, j1939ControllerApplicationToJ1939Node Mapping.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
mapping Constraint	MappingConstraint	*	aggr	Constraints that limit the mapping freedom for the mapping of SW components to ECUs. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=mappingConstraint, mapping Constraint.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**SystemMapping**			
pncMapping	PncMapping	*	aggr	Mappings between Virtual Function Clusters and Partial Network Clusters. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=pncMapping, pncMapping.ident.shortName, pncMapping.pncIdentifier, pncMapping.shortLabel, pnc Mapping.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
portElementTo ComResource Mapping	PortElementTo Communication ResourceMapping	*	aggr	maps a communication resource to CP Software Clusters **Stereotypes:** atpVariation **Tags:** atp.Splitkey=portElementToComResourceMapping, port ElementToComResourceMapping.shortName, port ElementToComResourceMapping.variationPoint.short Label vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.
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:** atp.Splitkey=resourceEstimation, resource Estimation.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
resourceTo Application Partition Mapping	CpSoftwareCluster ResourceToApplication PartitionMapping	*	aggr	Maps a Software Cluster resource to an Application Partition to restrict the usage. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=resourceToApplicationPartitionMapping, resourceToApplicationPartitionMapping.shortName, resourceToApplicationPartitionMapping.variation Point.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
rteEvent Separation	RteEventInSystem Separation	*	aggr	Separation constraint that limits the mapping freedom for the mapping of RteEvents to OsTasks in the System context. **Tags:** atp.Splitkey=rteEventSeparation, rteEvent Separation.shortName This Attribute is only used by the AUTOSAR Classic Platform.
rteEventToOs TaskProxy Mapping	RteEventInSystemToOs TaskProxyMapping	*	aggr	Constraint that enforces a mapping of RteEvent to a particular OsTask in the System context. **Tags:** atp.Splitkey=rteEventToOsTaskProxyMapping, rte EventToOsTaskProxyMapping.shortName This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**SystemMapping**			
signalPath Constraint	SignalPathConstraint	*	aggr	Constraints that limit the mapping freedom for the mapping of data elements to signals. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=signalPathConstraint, signalPath Constraint.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
softwareCluster ToApplication Partition Mapping	CpSoftwareClusterTo ApplicationPartition Mapping	*	aggr	The mapping of ApplicationPartitions to a CpSoftware Cluster. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=softwareClusterToApplicationPartition Mapping, softwareClusterToApplicationPartition Mapping.shortName, softwareClusterToApplication PartitionMapping.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
softwareCluster ToResource Mapping	CpSoftwareClusterTo ResourceMapping	*	aggr	maps a service resource to CP Software Clusters **Stereotypes:** atpVariation **Tags:** atp.Splitkey=softwareClusterToResourceMapping, softwareClusterToResourceMapping.shortName, softwareClusterToResourceMapping.variationPoint.short Label vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
swCluster Mapping	CpSoftwareClusterTo EcuInstanceMapping	*	aggr	The mappings of SW cluster to ECUs. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swClusterMapping, swClusterMapping.short Name, swClusterMapping.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
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:** atpVariation **Tags:** atp.Splitkey=swcToApplicationPartitionMapping, swcTo ApplicationPartitionMapping.shortName, swcTo ApplicationPartitionMapping.variationPoint.shortLabel vh.latestBindingTime=postBuild This Attribute is only used by the AUTOSAR Classic Platform.

▽

△

Class	**SystemMapping**			
swImplMapping	SwcToImplMapping	*	aggr	The mappings of AtomicSoftwareComponent Instances to Implementations. atpVariation: Derived, because SwcToEcuMapping is variable. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swImplMapping, swImplMapping.shortName, swImplMapping.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
swMapping	SwcToEcuMapping	*	aggr	The mappings of SW components to ECUs. atpVariation: SWC shall be mapped to other ECUs. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=swMapping, swMapping.shortName, swMapping.variationPoint.shortLabel vh.latestBindingTime=preCompileTime This Attribute is only used by the AUTOSAR Classic Platform.
systemSignalGroupToComResourceMapping	SystemSignalGroupToCommunicationResourceMapping	*	aggr	Mapping of a communication resource to a SystemSignal Group. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=systemSignalGroupToComResourceMapping, systemSignalGroupToComResourceMapping.shortName, systemSignalGroupToComResourceMapping.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.
systemSignalToComResourceMapping	SystemSignalToCommunicationResourceMapping	*	aggr	Mapping of a communication resource to a SystemSignal. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=systemSignalToComResourceMapping, systemSignalToComResourceMapping.shortName, systemSignalToComResourceMapping.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime This Attribute is only used by the AUTOSAR Classic Platform.

Table A.25: SystemMapping

Class	**VariableDataPrototype**
Note	A `VariableDataPrototype` represents a formalized generic piece of information that is typically mutable by the application software layer. `VariableDataPrototype` is used in various contexts and the specific context gives the otherwise generic `VariableDataPrototype` a dedicated semantics.
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	**VariableDataPrototype**			
Aggregated by	ApplicationInterface.indication, *AtpClassifier*.atpFeature, BswInternalBehavior.arTypedPerInstance Memory, BswModuleDescription.providedData, BswModuleDescription.requiredData, BulkNvData Descriptor.bulkNvBlock, DiagnosticSovdAccessArgument.contentObject, *InternalBehavior*.staticMemory, NvBlockDescriptor.ramBlock, NvDataInterface.nvData, SenderReceiverInterface.dataElement, Service Interface.event, SwcInternalBehavior.arTypedPerInstanceMemory, SwcInternalBehavior.explicitInter RunnableVariable, SwcInternalBehavior.implicitInterRunnableVariable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype **Tags:** atp.Splitkey=initValue

Table A.26: VariableDataPrototype

A.2 Source Code Example for ADC

The chapter shall show an example of MemMap usage in source code for an ADC implementation:

```
#define ADC_START_SEC_VAR_INIT_ASIL_B_32
#include <Adc_MemMap.h>

uint32 Adc_ResultBuffer[128];

#define ADC_STOP_SEC_VAR_INIT_ASIL_B_32
#include <Adc_MemMap.h>

#define ADC_CFG_START_SEC_CONST_ASIL_B_32
#include <Adc_MemMap.h>

const Adc_ConfigType AdcCfg[2] = INIT_VALUES;

#define ADC_CFG_STOP_SEC_CONST_ASIL_B_32
#include <Adc_MemMap.h>

#define ADC_START_SEC_CODE_SLOW_ASIL_B
#include <Adc_MemMap.h>

void Adc_Init(const Adc_ConfigType* ConfigPtr) { ; }

#define ADC_STOP_SEC_CODE_SLOW_ASIL_B
#include <Adc_MemMap.h>

#define ADC_START_SEC_CODE_SLOW_ASIL_B
#include <Adc_MemMap.h>

void Adc_DeInit(void) { ; }

#define ADC_STOP_SEC_CODE_SLOW_ASIL_B
#include <Adc_MemMap.h>

#define ADC_START_SEC_CODE_FAST_ASIL_B
#include <Adc_MemMap.h>

```

```
36 void Adc_StartGroupConversion (Adc_GroupType Group) { ; }
37
38 #define ADC_STOP_SEC_CODE_FAST_ASIL_B
39 #include <Adc_MemMap.h>
```

A.3 Memory Mapping Header File Example for ADC

The Memory Allocation Header file `Adc_MemMap.h` related to the usage in chapter A.2 is shown below. The included file `MemMap_RestoreUnhandledDefaults.h` is assumed to be vendor specific and used to set the unhandled default sections for robustness handling. The detailed content has to be defined according to the used compiler/linker.

```
1  /* Initialization of overall error handling */
2  #define MEMMAP_ERROR
3
4  /* Keyword evaluation */
5  #if defined ADC_START_SEC_VAR_INIT_ASIL_B_32
6    #undef MEMMAP_ERROR
7    #undef ADC_START_SEC_VAR_INIT_ASIL_B_32
8    #ifndef MEMMAP_SEQUENCE_OPEN
9      /* pragma start */
10     #include "MemMap_RestoreUnhandledDefaults.h"
11     #pragma section fardata "ram.partition_asil_b.32"
12     #pragma section farbss "ram.partition_asil_b.32"
13     #pragma clear
14     /* pragma end */
15     #define MEMMAP_SEQUENCE_OPEN
16     #define MEMMAP_SEQUENCE_OPEN_ADC_SEC_VAR_INIT_ASIL_B_32
17   #else
18     #error "Adc_MemMap.h: ADC_SEC_VAR_INIT_ASIL_B_32: Please STOP the
           sequence before, START must not be followed by START!"
19   #endif
20 #elif defined ADC_STOP_SEC_VAR_INIT_ASIL_B_32
21   #undef MEMMAP_ERROR
22   #undef ADC_STOP_SEC_VAR_INIT_ASIL_B_32
23   #ifdef MEMMAP_SEQUENCE_OPEN
24     #ifdef MEMMAP_SEQUENCE_OPEN_ADC_SEC_VAR_INIT_ASIL_B_32
25       /* pragma start */
26       #include "MemMap_RestoreUnhandledDefaults.h"
27       /* pragma end */
28       #undef MEMMAP_SEQUENCE_OPEN
29       #undef MEMMAP_SEQUENCE_OPEN_ADC_SEC_VAR_INIT_ASIL_B_32
30     #else
31       #error "Adc_MemMap.h: ADC_SEC_VAR_INIT_ASIL_B_32: START section
             is followed by wrong STOP section statement!"
32     #endif
33   #else
34       #error "Adc_MemMap.h: ADC_SEC_VAR_INIT_ASIL_B_32: No START
             statement given before STOP statement! STOP must not be
             followed by STOP!"
35   #endif
36 #endif
```

```
37
38 #if defined ADC_START_SEC_CODE_FAST_ASIL_B
39   #undef MEMMAP_ERROR
40   #undef ADC_START_SEC_CODE_FAST_ASIL_B
41   #ifndef MEMMAP_SEQUENCE_OPEN
42     /* pragma start */
43     #include "MemMap_RestoreUnhandledDefaults.h"
44     #pragma section text "rom.fast.partition_asil_b"
45     /* pragma end */
46     #define MEMMAP_SEQUENCE_OPEN
47     #define MEMMAP_SEQUENCE_OPEN_ADC_SEC_CODE_FAST_ASIL_B
48   #else
49     #error "Adc_MemMap.h: ADC_SEC_CODE_FAST_ASIL_B: Please STOP the
         sequence before, START must not be followed by START!"
50   #endif
51 #elif defined ADC_STOP_SEC_CODE_FAST_ASIL_B
52   #undef MEMMAP_ERROR
53   #undef ADC_STOP_SEC_CODE_FAST_ASIL_B
54   #ifdef MEMMAP_SEQUENCE_OPEN
55     #ifdef MEMMAP_SEQUENCE_OPEN_ADC_SEC_CODE_FAST_ASIL_B
56       /* pragma start */
57       #include "MemMap_RestoreUnhandledDefaults.h"
58       /* pragma end */
59       #undef MEMMAP_SEQUENCE_OPEN
60       #undef MEMMAP_SEQUENCE_OPEN_ADC_SEC_CODE_FAST_ASIL_B
61     #else
62       #error "Adc_MemMap.h: ADC_SEC_CODE_FAST_ASIL_B: START section is
           followed by wrong STOP section statement!"
63     #endif
64   #else
65     #error "Adc_MemMap.h: ADC_SEC_CODE_FAST_ASIL_B: No START statement
         given before STOP statement! STOP must not be followed by STOP!
         "
66   #endif
67 #endif
68
69 #if defined ADC_START_SEC_CODE_SLOW_ASIL_B
70   #undef MEMMAP_ERROR
71   #undef ADC_START_SEC_CODE_SLOW_ASIL_B
72   #ifndef MEMMAP_SEQUENCE_OPEN
73     /* pragma start */
74     #include "MemMap_RestoreUnhandledDefaults.h"
75     #pragma section text "rom.slow.partition_asil_b"
76     /* pragma end */
77     #define MEMMAP_SEQUENCE_OPEN
78     #define MEMMAP_SEQUENCE_OPEN_ADC_SEC_CODE_SLOW_ASIL_B
79   #else
80     #error "Adc_MemMap.h: ADC_SEC_CODE_SLOW_ASIL_B: Please STOP the
         sequence before, START must not be followed by START!"
81   #endif
82
83 #elif defined ADC_STOP_SEC_CODE_SLOW_ASIL_B
84   #undef MEMMAP_ERROR
85   #undef ADC_STOP_SEC_CODE_SLOW_ASIL_B
86   #ifdef MEMMAP_SEQUENCE_OPEN
87     #ifdef MEMMAP_SEQUENCE_OPEN_ADC_SEC_CODE_SLOW_ASIL_B
```

```
88        /* pragma start */
89        #include "MemMap_RestoreUnhandledDefaults.h"
90        /* pragma end */
91        #undef MEMMAP_SEQUENCE_OPEN
92        #undef MEMMAP_SEQUENCE_OPEN_ADC_SEC_CODE_SLOW_ASIL_B
93      #else
94        #error "Adc_MemMap.h: ADC_SEC_CODE_SLOW_ASIL_B: START section is
              followed by wrong STOP section statement!"
95      #endif
96    #else
97        #error "Adc_MemMap.h: ADC_SEC_CODE_SLOW_ASIL_B: No START
              statement given before STOP statement! STOP must not be
              followed by STOP!"
98    #endif
99  #endif
100
101 #if defined ADC_CFG_START_SEC_CONST_ASIL_B_32
102   #undef MEMMAP_ERROR
103   #undef ADC_CFG_START_SEC_CONST_ASIL_B_32
104   #ifndef MEMMAP_SEQUENCE_OPEN
105     /* pragma start */
106     #include "MemMap_RestoreUnhandledDefaults.h"
107     #pragma section rodata "rom.partition_asil_b.32"
108     /* pragma end */
109     #define MEMMAP_SEQUENCE_OPEN
110     #define MEMMAP_SEQUENCE_OPEN_ADC_CFG_SEC_CONST_ASIL_B_32
111   #else
112     #error "Adc_MemMap.h: ADC_CFG_SEC_CONST_ASIL_B_32: Please STOP the
            sequence before, START must not be followed by START!"
113   #endif
114 #elif defined ADC_CFG_STOP_SEC_CONST_ASIL_B_32
115   #undef MEMMAP_ERROR
116   #undef ADC_CFG_STOP_SEC_CONST_ASIL_B_32
117   #ifdef MEMMAP_SEQUENCE_OPEN
118     #ifdef MEMMAP_SEQUENCE_OPEN_ADC_CFG_SEC_CONST_ASIL_B_32
119       /* pragma start */
120       #include "MemMap_RestoreUnhandledDefaults.h"
121       /* pragma end */
122       #undef MEMMAP_SEQUENCE_OPEN
123       #undef MEMMAP_SEQUENCE_OPEN_ADC_CFG_SEC_CONST_ASIL_B_32
124     #else
125       #error "Adc_MemMap.h: ADC_CFG_SEC_CONST_ASIL_B_32: START section
              is followed by wrong STOP section statement!"
126     #endif
127   #else
128     #error "Adc_MemMap.h: ADC_CFG_SEC_CONST_ASIL_B_32: No START
          statement given before STOP statement! STOP must not be
          followed by STOP!"
129   #endif
130 #endif
131
132 /* Error evaluation */
133 #ifdef MEMMAP_ERROR
134   #undef MEMMAP_ERROR
```

```
135   #error "Adc_MemMap.h: Undefined or missing START / STOP statement,
          please check your source code or re-generate the MemMap Header
          file!"
136 #endif
```

A.4 Specification Items

A.4.1 Added Specification Items in R26-11

none

A.4.2 Changed Specification Items in R26-11

none

A.4.3 Deleted Specification Items in R26-11

none

A.5 Trace Items not relevant for this document

[SWS_MemMap_99999] Unreferenced Reqirements

Upstream requirements: SRS_BSW_00170, SRS_BSW_00168, SRS_BSW_00369, SRS_BSW_00375, SRS_BSW_00383, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00403, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00425, SRS_BSW_00432, SRS_BSW_00461, SRS_BSW_00469, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00478, SRS_BSW_00490, SRS_BSW_00491

⌈Incoming requirements are checked, but not relevant for this document.⌋