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

Document Version	1.2.2
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	1.2.2	AUTOSAR Release Management	Editorial changes
07.04.2011	1.2.1	AUTOSAR Administration	Legal disclaimer revised
10.09.2010	1.2.0	AUTOSAR Administration	• MEMMAP003 changed: Application hint added for the handling of INLINE code implementation. • Legal disclaimer revised
23.06.2008	1.1.1	AUTOSAR Administration	Legal disclaimer revised
12.12.2007	1.1.0	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 MEMMAP021,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
13.02.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

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

Avoidance of waste of RAM
If different variables (8, 16 and 32 bit) are used within different modules on a 32 bit platform, the linker will leave gaps in RAM when allocating the variables in the RAM. This is because the microcontroller platform requires a specific alignment of variables and some linkers do not allow an optimization of variable allocation.

This waste of memory can be circumvented if the variables are mapped to specific memory sections depending on their size. This minimizes unused space in RAM.

Usage of specific RAM properties
Some variables (e.g. the RAM mirrors of the NVRAM Manager) must not be initialized after a power-on reset. It shall be possible to map them to a RAM section that is not initialized after a reset.

For some variables (e.g. variables that are accessed via bit masks) it improves both performance and code size if they are located within a RAM section that allows for bit manipulation instructions of the compiler. Those RAM sections are usually known as ‘Near Page’ or ‘Zero Page’.

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

Usage of the same source code of a module for boot loader and application
If a module shall be used both in boot loader and application, it is necessary to allow the mapping of code and data to different memory sections.

A mechanism for mapping of code and data to memory sections that is supported by all compilers listed in chapter 3.1 is the usage of pragmas. As pragmas are very compiler specific, a mechanism that makes use of those pragmas in a standardized way has to be specified.

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

2 Acronyms and abbreviations

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

3 Related documentation

3.1 Input documents

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

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

[3] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

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

[5] ARM ADS compiler manual

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

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

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

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

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

3.2 Related standards and norms

Not applicable.

4 Constraints and assumptions

4.1 Limitations

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

The concepts described in this document do only apply to C compilers. C++ is not in scope of this version.

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

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

4.2 Applicability to car domains

No restrictions.

4.3 Applicability to safety related environments

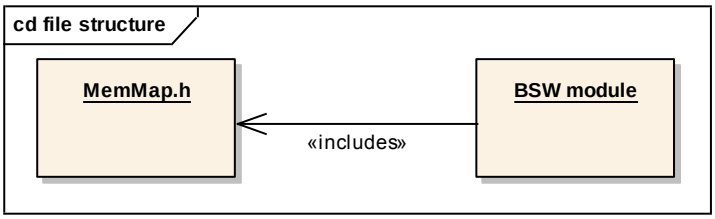
No restrictions. The memory mapping file does not implement any functionality, only symbols and macros.

5 Dependencies to other modules

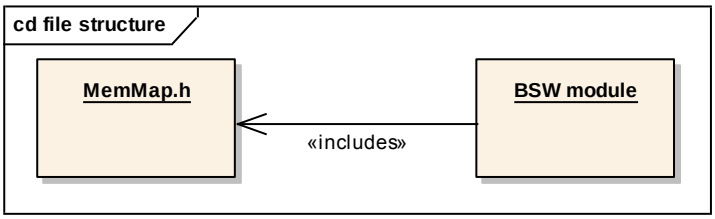
MEMMAP020: The SWS Memory Mapping is applicable for each AUTOSAR software module. Therefore the implementation of memory mapping file shall fulfil the implementation and configuration specific needs of each software module in a specific build scenario. See also MEMMAP004, MEMMAP003, MEMMAP018 and MEMMAP001MEMMAP008.

5.1 File structure

5.1.1 Code file structure



5.1.2 Header file structure



6 Requirements traceability

Document: AUTOSAR General Requirements on Basic Software Modules

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Not applicable (Memory Mapping is specific per build scenario)
[BSW00404] Reference to post build time configuration	Not applicable (Memory Mapping is specific per build scenario)
[BSW00405] Reference to multiple configuration sets	Not applicable (Memory Mapping is specific per build scenario)
[BSW00345] Pre-compile-time configuration	Not applicable (Memory Mapping is specific per build scenario)
[BSW159] Tool-based configuration	Not applicable (Memory Mapping is specific per build scenario)
[BSW167] Static configuration checking	Not applicable (Memory Mapping is specific per build scenario)
[BSW171] Configurability of optional functionality	Not applicable (Memory Mapping is specific per build scenario)
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (Memory Mapping is specific per build scenario)
[BSW00380] Separate C-Files for configuration parameters [approved]	Not applicable (Memory Mapping is specific per build scenario)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00381] Separate configuration header file for pre-compile time parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00412] Separate H-File for configuration parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00383] List dependencies of configuration files	Not applicable (Memory Mapping is specific per build scenario)
[BSW00384] List dependencies to other modules	MEMMAP020
[BSW00387] Specify the configuration class of callback function	Not applicable (Memory Mapping is specific per build scenario)
[BSW00388] Introduce containers	Not applicable (Memory Mapping is specific per build scenario)
[BSW00389] Containers shall have names	Not applicable (Memory Mapping is specific per build scenario)
[BSW00390] Parameter content shall be unique within the module	Not applicable (Memory Mapping is specific per build

Requirement	Satisfied by
	scenario)
[BSW00391] Parameter shall have unique names	Not applicable (Memory Mapping is specific per build scenario)
[BSW00392] Parameters shall have a type	Not applicable (Memory Mapping is specific per build scenario)
[BSW00393] Parameters shall have a range	Not applicable (Memory Mapping is specific per build scenario)
[BSW00394] Specify the scope of the parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00395] List the required parameters (per parameter)	Not applicable (Memory Mapping is specific per build scenario)
[BSW00396] Configuration classes	Not applicable (Memory Mapping is specific per build scenario)
[BSW00397] Pre-compile-time parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00398] Link-time parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00399] Loadable Post-build time parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00400] Selectable Post-build time parameters	Not applicable (Memory Mapping is specific per build scenario)
[BSW00402] Published information	MEMMAP019
[BSW00375] Notification of wake-up reason	Not applicable (Memory Mapping is not a BSW module)
[BSW101] Initialization interface	Not applicable (Memory Mapping is not a BSW module)
[BSW00416] Sequence of Initialization	Not applicable (Memory Mapping is not a BSW module)
[BSW00406] Check module initialization	Not applicable (Memory Mapping is not a BSW module)
[BSW168] Diagnostic Interface of SW components	Not applicable (Memory Mapping is not a BSW module)
[BSW00407] Function to read out published parameters	Not applicable (Memory Mapping is not a BSW module)
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (Memory Mapping is not a BSW module)
[BSW00424] BSW main processing function task allocation	Not applicable (Memory Mapping is not a BSW module)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (Memory Mapping is not a BSW module)
[BSW00426] Exclusive areas in BSW modules	Not applicable (Memory Mapping is not a BSW module)
[BSW00427] ISR description for BSW modules	Not applicable (Memory Mapping is not a BSW module)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (Memory Mapping is not a BSW module)
[BSW00429] Restricted BSW OS functionality access	Not applicable (Memory Mapping is not a BSW module)

Requirement	*Satisfied by*
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (Memory Mapping is not a BSW module)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (Memory Mapping is not a BSW module)
[BSW00433] Calling of main processing functions	Not applicable (Memory Mapping is not a BSW module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (Memory Mapping is not a BSW module)
[BSW00336] Shutdown interface	Not applicable (Memory Mapping is not a BSW module)
[BSW00337] Classification of errors	Not applicable (Memory Mapping is not a BSW module)
[BSW00338] Detection and Reporting of development errors	Not applicable (Memory Mapping is not a BSW module)
[BSW00369] Do not return development error codes via API	Not applicable (Memory Mapping is not a BSW module)
[BSW00339] Reporting of production relevant error status	Not applicable (Memory Mapping is not a BSW module)
[BSW00421] Reporting of production relevant error events	Not applicable (Memory Mapping is not a BSW module)
[BSW00422] Debouncing of production relevant error status	Not applicable (Memory Mapping is not a BSW module)
[BSW00420] Production relevant error event rate detection	Not applicable (Memory Mapping is not a BSW module)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable, (Memory Mapping does not report errors)
[BSW00323] API parameter checking	Not applicable (Memory Mapping is not a BSW module)
[BSW004] Version check	Not applicable (Memory Mapping is not a BSW module)
[BSW00409] Header files for production code error IDs	Not applicable (Memory Mapping is not a BSW module)
[BSW00385] List possible error notifications	Not applicable (Memory Mapping is not a BSW module)
[BSW00386] Configuration for detecting an error	Not applicable (Memory Mapping is not a BSW module)
[BSW161] Microcontroller abstraction	Not applicable (non-functional requirement)
[BSW162] ECU layout abstraction	Not applicable (non-functional requirement)
[BSW00324] Do not use HIS I/O Library	Not applicable (non-functional requirement)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (non-functional requirement)
[BSW00415] User dependent include files	Not applicable (non-functional requirement)
[BSW164] Implementation of interrupt service routines	Not applicable (non-functional requirement)
[BSW00325] Runtime of interrupt service routines	Not applicable (Memory Mapping is not a BSW module)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (Memory Mapping is not a BSW module)
[BSW00342] Usage of source code and object code	Not applicable (non-functional requirement)

Requirement	*Satisfied by*
[BSW00343] Specification and configuration of time	Not applicable (Memory Mapping is not a BSW module)
[BSW160] Human-readable configuration data	Not applicable (Memory Mapping is not a BSW module)
[BSW007] HIS MISRA C	Not applicable, (Memory Mapping is the C-language extension header)
[BSW00300] Module naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00413] Accessing instances of BSW modules	Not applicable (Memory Mapping is not a BSW module)
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (Memory Mapping is not a BSW module)
[BSW00305] Self-defined data types naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00307] Global variables naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00310] API naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00373] Main processing function naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00327] Error values naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00335] Status values naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00350] Development error detection keyword	Not applicable (Memory Mapping is not a BSW module)
[BSW00408] Configuration parameter naming convention	Not applicable (Memory Mapping is not a BSW module)
[BSW00410] Compiler switches shall have defined values	Not applicable (Memory Mapping is not a BSW module)
[BSW00411] Get version info keyword	Not applicable (Memory Mapping is not a BSW module)
[BSW00346] Basic set of module files	Not applicable (Memory Mapping is not a BSW module)
[BSW158] Separation of configuration from implementation	Not applicable (Memory Mapping is not a BSW module)
[BSW00314] Separation of interrupt frames and service routines	Not applicable (Memory Mapping is not a BSW module)
[BSW00370] Separation of callback interface from API	Not applicable (Memory Mapping is not a BSW module)
BSW00348] Standard type header	Not applicable (Memory Mapping is not a BSW module)
[BSW00353] Platform specific type header	Not applicable (Memory Mapping is a C-language extension header)
[BSW00361] Compiler specific language extension header	MEMMAP002
[BSW00301] Limit imported information	Not applicable (Memory Mapping is not a BSW module)
[BSW00302] Limit exported information	Not applicable (Memory Mapping is not a BSW module)
[BSW00328] Avoid duplication of code	supported by: MEMMAP001, MEMMAP005
[BSW00312] Shared code shall be reentrant	Not applicable (Memory Mapping is not a BSW module)
[BSW006] Platform independency	supported by:

Requirement	*Satisfied by*
	MEMMAP010, MEMMAP004, MEMMAP003, MEMMAP005, MEMMAP006, MEMMAP007, MEMMAP011, MEMMAP013
[BSW00357] Standard API return type	Not applicable (Memory Mapping is not a BSW module)
[BSW00377] Module specific API return types	Not applicable (Memory Mapping is not a BSW module)
[BSW00304] AUTOSAR integer data types	Not applicable (Memory Mapping is not a BSW module)
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (Memory Mapping is not a BSW module)
[BSW00378] AUTOSAR boolean type	Not applicable (Memory Mapping is not a BSW module)
[BSW00306] Avoid direct use of compiler and platform specific keywords	supported by: MEMMAP010, MEMMAP004, MEMMAP003, MEMMAP005, MEMMAP006, MEMMAP007, MEMMAP011, MEMMAP013
[BSW00308] Definition of global data	Not applicable (Memory Mapping is not a BSW module)
[BSW00309] Global data with read-only constraint	Not applicable (Memory Mapping is not a BSW module)
[BSW00371] Do not pass function pointers via API	Not applicable (Memory Mapping is not a BSW module)
[BSW00358] Return type of `init()` functions	Not applicable (Memory Mapping is not a BSW module)
[BSW00414] Parameter of init function	Not applicable (Memory Mapping is not a BSW module)
[BSW00414] Parameter of init function	Not applicable (Memory Mapping is not a BSW module)
[BSW00359] Return type of callback functions	Not applicable (Memory Mapping is not a BSW module)
[BSW00360] Parameters of callback functions	Not applicable (Memory Mapping is not a BSW module)
[BSW00329] Avoidance of generic interfaces	Not applicable (Memory Mapping is not a BSW module)
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (Memory Mapping is not a BSW module)
[BSW00331] Separation of error and status values	Not applicable (Memory Mapping is not a BSW module)
[BSW009] Module User Documentation	Not applicable (Memory Mapping is not a BSW module)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (Memory Mapping is not a BSW module)
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (Memory Mapping is not a BSW module)
[BSW010] Memory resource documentation	Not applicable (Memory Mapping is not a BSW module)
[BSW00333] Documentation of callback function context	Not applicable (Memory Mapping is not a BSW module)
[BSW00374] Module vendor identification	MEMMAP019
[BSW00379] Module identification	MEMMAP019
[BSW003] Version identification	MEMMAP019
[BSW00318] Format of module version numbers	MEMMAP019

Requirement	***Satisfied by***
[BSW00321] Enumeration of module version numbers	MEMMAP019
[BSW00341] Microcontroller compatibility documentation	Not applicable (Memory Mapping is not a BSW module)
[BSW00334] Provision of XML file	Not applicable (Memory Mapping is not a BSW module)

7 Analysis

This chapter does not contain requirements. It just gives an overview to used keywords and their syntax within different compilers. This analysis is required for a correct and complete specification of methods and keywords.

7.1 Memory allocation of variables

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

Compiler	***Required syntax***

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

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

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

Tasking, TC1796	`#pragma pack 0 / 2` packing of structs. Shall be visible at type declaration

Compiler	Required syntax

	`#pragma section type "string"`
	`#pragma noclear`
	
	`#pragma clear`
	
	`#pragma for_extern_data_use_memory`
	`#pragma for_initialized_data_use_memory`
	`#pragma for_uninitialized_data_use_memory`

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

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

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

7.2 Memory allocation of constant variables

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

Compiler	Required syntax
Cosmic, S12X	`#pragma section const {name}` `#pragma section const {}`

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

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

Compiler	*Required syntax*
	G Global S Sysstack U Usrstack A address Absolute section AT constant address (decimal, octal or hexadecimal number)

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

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

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

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

7.3 Memory allocation of code

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

Compiler	*Required syntax*
Cosmic, S12X	`#pragma section ( name )` `#pragma section ()`

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

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

Compiler	***Required syntax***

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

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

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

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

8 Functional specification

8.1 General issues

The memory mapping file includes 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.

MEMMAP001: For each build scenario (e.g. Boot loader, ECU Application) an own memory mapping file has to be provided.

MEMMAP002: The memory mapping file name shall be ‘MemMap.h’.

MEMMAP010: If a compiler/linker does not require or support requisite functionality of SWS Memory Mapping, the memory allocation keyword defines shall be undefined without further effect.

For instance:
```
#ifdef EEP_START_SEC_VAR_16BIT
    #undef EEP_START_SEC_VAR_16BIT
#endif
```

8.2 Mapping of variables and code

8.2.1 Requirements on implementations using MemMap.h

MEMMAP004: Each AUTOSAR software module shall support the configuration of at least the following different memory types. It is allowed to add module specific sections as they are mapped and thus are configurable within the module’s configuration file. The shortcut ‘MSN’ means ‘module short name of BSW module list’, e.g. ‘EEP’ or ‘CAN’.
The shortcut ‘SIZE’ means the variable size. Possible SIZE postfixes are

BOOLEAN, used for variables and constants of size 1 bit
8BIT, used for variables and constants of size 8 bit
16BIT, used for variables and constants of size 16 bit
32BIT, used for variables and constants of size 32 bit
UNSPECIFIED, used for variables and constants of unknown size

START__START
START__STOP

Memory type	*Syntax of memory allocation keyword*	*Comments*
Code	<MSN>_START_SEC_CODE	To be used for mapping code to application block, boot block, external flash etc.
	<MSN>_STOP_SEC_CODE	
Variables	<MSN>_START_SEC_VAR_NOINIT_<SIZE>	To be used for all global or static variables that are never initialized
	<MSN>_STOP_SEC_VAR_NOINIT_<SIZE>	
Variables	<MSN>_START_SEC_VAR_POWER_ON_INIT_<SIZE>	To be used for all global or static variables that are initialized only after power on reset
	<MSN>_STOP_SEC_VAR_POWER_ON_INIT_<SIZE>	
Variables	<MSN>_START_SEC_VAR_FAST_<SIZE>	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.
	<MSN>_STOP_SEC_VAR_FAST_<SIZE>	
Variables	<MSN>_START_SEC_INTERNAL_VAR_<SIZE>	To be used for global or static variables accessible from a calibration tool.
	<MSN>_STOP_SEC_INTERNAL_VAR_<SIZE>	
Variables	<MSN>_START_SEC_VAR_SAVED_ZONE<X>_<SIZE>	To be used for RAM buffers of variables saved in non volatile memory.
	<MSN>_STOP_SEC_VAR_SAVED_ZONE<X>_<SIZE>	
Variables	<MSN>_START_SEC_VAR_SAVED_RECOVERY_ZONE<X>	To be used for ROM buffers of variables saved in non volatile memory.
	<MSN>_STOP_SEC_VAR_SAVED_RECOVERY_ZONE<X>	
Variables	<MSN>_START_SEC_VAR_<SIZE>	To be used for global or static variables that are initialized after every reset (the normal case).
	<MSN>_STOP_SEC_VAR_<SIZE>	
Constants	<MSN>_START_SEC_CONST_<SIZE>	To be used for global or static constants.
	<MSN>_STOP_SEC_CONST_<SIZE>	
Constants	<MSN>_START_SEC_CALIB_<SIZE>	To be used for calibration constants.
	<MSN>_STOP_SEC_CALIB_<SIZE>	
Constants	<MSN>_START_SEC_CARTO_<SIZE>	To be used for cartography constants.
	<MSN>_STOP_SEC_CARTO_<SIZE>	
Configuration data	<MSN>_START_CONFIG_DATA_<SIZE> <MSN>_STOP_CONFIG_DATA_<SIZE>	Constants with attributes that show that they reside in one segment for module configuration.

MEMMAP021: 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 the keyword to be used for each code section:

Memory type	*Syntax of memory allocation keyword*	*Comments*
Fast code	<MSN>_START_SEC_CODE_FAST_<NUM> <MSN>_STOP_SEC_CODE_FAST_<NUM>	To be used for code that shall go into fast code memory segments.
Slow code	<MSN>_START_SEC_CODE_SLOW <MSN>_STOP_SEC_CODE_SLOW	To be used for code that shall go into slow code memory segments.
Library code	<MSN>_START_SEC_CODE_LIB <MSN>_STOP_SEC_CODE_LIB	To be used for code that shall go into library segments for <MSN> module.

MEMMAP003: Each AUTOSAR software module 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 MemMap.h
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 MemMap.h

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

Application hint:
For code which his implemented with the `INLINE` macro of the “Compiler.h” the wrapping with Memory Allocation Keywords is required at least for the code which is remaining if `INLINE` is set to empty.
In the case that the `INLINE` is set to the inline keyword of the compiler the related Memory Allocation Keywords shall not define any linker section assignments or change the addressing behavior because this is already set by the environment of the calling function where the code is inlined. In the case that the `INLINE` is set to empty the related Memory Allocation Keywords shall be configured like for regular code.
Please note as well that in the Basic Software Module Description the `MemorySection` related to the used Memory Allocation Keywords has to document the usage of `INLINE` in the option attribute. For further information see [3]

The inclusion of MemMap.h 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 “MemMap.h”. For details on configuring sections see “Configuration specification”

For instance:

```
#define EEP_START_SEC_VAR_16BIT
#include “MemMap.h”
static uint16 EepTimer;
static uint16 EepRemainingBytes;
#define EEP_STOP_SEC_VAR_16BIT
#include “MemMap.h”
```

MEMMAP018: Each AUTOSAR software module shall support the configuration of all C-objects assignable to one of the memory types code, variables and constants.

Application hint:
An implicit assignment of object to default sections is not allowed because properties of default sections are platform and tool depended and therefore these implementations are not platform independed.

8.2.2 Requirements on MemMap.h

MEMMAP005: The file MemMap.h 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 MEMMAP004). Code, variables or constants declared after this selection shall be mapped to this section.

MEMMAP015: The selected section shall be activated, if the section macro is defined before include of the file “MemMap.h”.

MEMMAP016: The selection of a section shall only influence the linkers behaviour for one of the three different object types code, variables or constants concurrently.

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

For instance:

```
#ifdef EEP_START_SEC_VAR_16BIT
```

```
    #undef EEP_START_SEC_VAR_16BIT
    #define START_SECTION_DATA_16BIT
#elif
/*
   additional mappings of modules sections into project
   sections
*/
...
#endif


#ifdef START_SECTION_DATA_16BIT
    #pragma section data "sect_data16"
    #undef START_SECTION_DATA_16BIT
    #undef MEMMAP_ERROR
#elif
/*
   additional statements for switching the project sections
*/
...
#endif
```

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

MEMMAP006: The file MemMap.h shall provide a mechanism to deselect different code and variable sections by checking the definition of the module specific memory allocation key words for stopping a section (see MEMMAP004). Code or variables declared after this selection shall be mapped to default section. The selected section shall be deactivated, if the section macro is defined before include of the file “MemMap.h”.

For instance:

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


/* additional module specific mappings */
...

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

MEMMAP007: The file MemMap.h shall check if it has been included with a valid memory mapping symbol. This shall be done by a preprocessor check.

For instance:

```
#define MEMMAP_ERROR

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

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

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

MEMMAP011: The file MemMap.h shall undefine the module specific memory allocation key words for starting or stopping a section.

For instance:
```
#ifdef EEP_STOP_SEC_CODE
    #undef EEP_STOP_SEC_CODE
```

MEMMAP013: The file MemMap.h shall use if-else structures reducing the compilation effort.

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

#endif
```

9 API specification

Not applicable.

10 Sequence diagrams

Not applicable.

11 Configuration specification

The file MemMap.h is specific for each build scenario. Therefore there is no standardized configuration interface specified.

11.1 Published Information

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

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [3] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.