Document Title	Specification of Module Flash Driver
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.1.0
Document Status	Draft
Part of Release	2.1
Revision	0015

Document Change History

Date	Version	Changed by	Change Description
14.02.2007	2.1.0	AUTOSAR Administration	• File include structure updated • Type usage corrected • Compare Job results adapted • API towards DEM corrected • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
10.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template • new functionality: Read, Compare and SetMode functions • scalability: functionality can be configured (on/off) • adapted to new MemHwA architecture
10.07.2004	1.0.0	AUTOSAR Administration	Initial release

Release Notes

Compatibility considerations with respect to current release

No known incompatibilities

Errata and known deficiencies

- The inclusion of the header file of the Memory Mapping Module is depicted in Figure 5-1 but not properly traced by to the related general BSW Requirements

- Missing entries in traceability matrix

Changes planned for next release

Traceability matrix may be updated.

Disclaimer

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

This document specifies the functionality, API and the configuration of the AUTOSAR Basic Software module Flash Driver.

This specification is applicable to drivers for both internal and external flash memory.

The flash driver provides services for reading, writing and erasing flash memory and a configuration interface for setting / resetting the write / erase protection if supported by the underlying hardware.

In application mode of the ECU, the flash driver is only to be used by the Flash EEPROM emulation module for writing data. It is not intended to write program code to flash memory in application mode. This shall be done in boot mode which is out of scope of AUTOSAR.

A driver for an internal flash memory accesses the microcontroller hardware directly and is located in the Microcontroller Abstraction Layer. An external flash memory is usually connected via the microcontroller's data / address busses (memory mapped access), the flash driver then uses the handlers / drivers for those busses to access the external flash memory device. The driver for an external flash memory device is located in the ECU Abstraction Layer.

FLS088: The functional requirements and the functional scope are the same for both types of drivers. Hence the API is semantically identical.

2 Acronyms and abbreviations

Abbreviation / Acronym:	*Description:*
DET	Development Error Tracer – module to which development errors are reported.
DEM	Diagnostic Event Manager – module to which production relevant errors are reported.
AC	(Flash) access code – abbreviation introduced to keep the names of the configuration parameters reasonably short.

Further definitions of terms used throughout this document

Term:	*Definition*
Flash sector	A flash sector is the smallest amount of flash memory that can be erased in one pass. The size of the flash sector depends upon the flash technology and is therefore hardware dependent.
Flash page	A flash page is the smallest amount of flash memory that can be programmed in one pass. The size of the flash page depends upon the flash technology and is therefore hardware dependent.
Flash access code	Internal flash driver routines called by the main function (job processing function) to erase or write the flash hardware.

3 Related documentation

3.1 AUTOSAR deliverables

[1] List of Basic Software Modules,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SoftwareModuleList.pdf

[2] Layered Software Architecture,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[4] General Requirements on SPAL,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_SPAL_General.pdf

[5] Requirements on Flash Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_Flash_Driver.pdf

[6] Requirements on Memory Hardware Abstraction Layer,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_MemHW_AbstractionLayer.pdf

3.2 Related standards and norms

[7] HIS Flash Driver Specification
HIS flash driver v130.pdf on
http://www.automotive-his.de/download/

4 Constraints and assumptions

4.1 Limitations

- The flash driver only erases or programs complete flash sectors respectively flash pages, i.e. it does not offer any kind of re-write strategy since it does not use any internal buffers.
- The flash driver does not provide mechanisms for providing data integrity (e.g. checksums, redundant storage, etc.).

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

5.1.1 Code file structure

FLS159: The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- Fls_Lcfg.c – for link time configurable parameters and
- Fls_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

FLS179: Pre- and post-compile configuration parameters shall be located outside the source code of the module to allow for automatic (tool based) configuration.

5.1.2 Header file structure

FLS107: The file include structure shall be as follows:

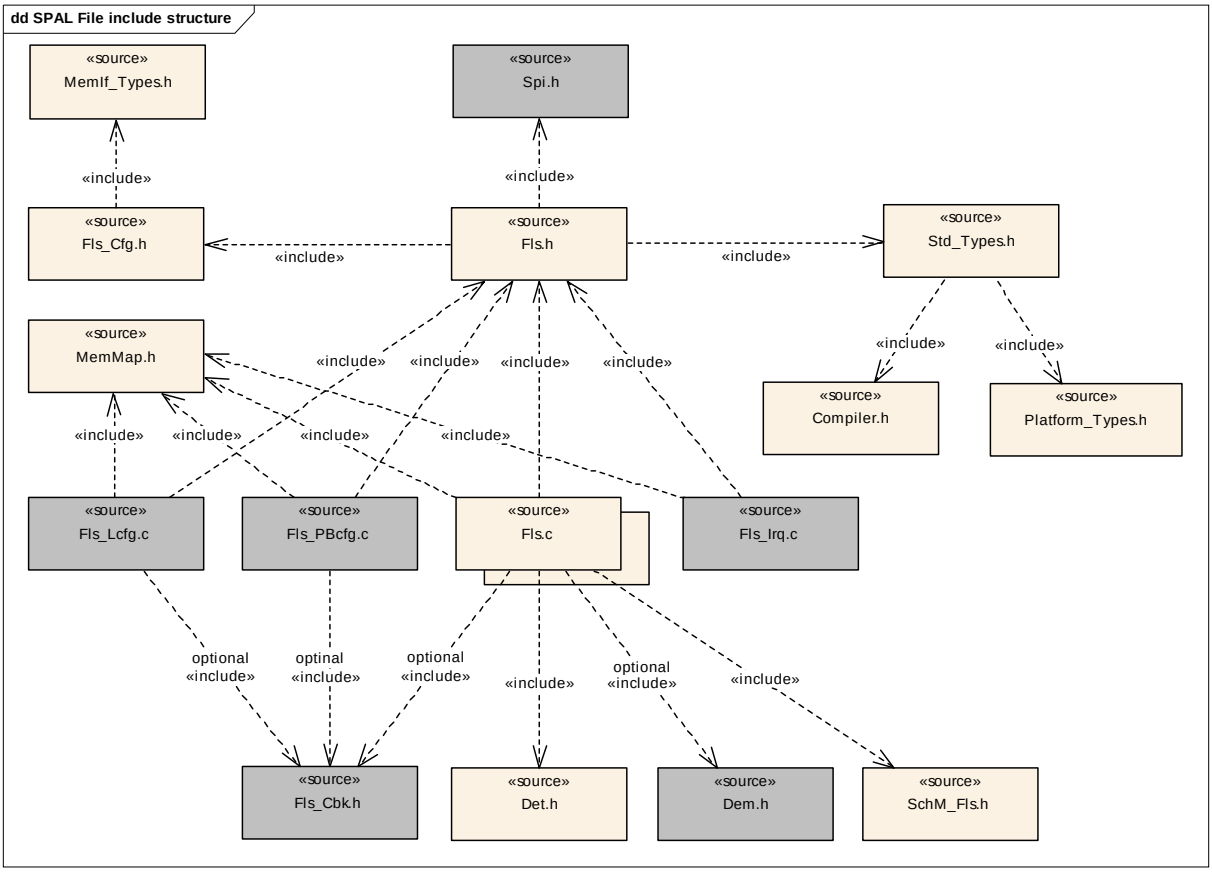

Figure 1: File include structure

- `Fls.h` shall include `MemIf_Types.h`, `Dem.h` and `Det.h`
- `Fls.h` shall include `Fls_Cfg.h`
- `Fls.c` shall include `Fls.h`
- `Fls_PBcfg.c` shall include `Fls.h`
- In case of a driver for an external SPI flash memory, `Fls.h` shall include `Spi.h`

FLS073: Types and definitions common to several flash driver instances shall be given in the header file “MemIf_Types.h”, Types and definitions specific for one flash driver shall be given in the header file “Fls.h”. This file shall be included in the flash driver’s implementation module “Fls.c”.

5.2 System clock

If the hardware of the internal flash memory depends on the system clock, changes to the system clock (e.g. PLL on → PLL off) may also affect the clock settings of the flash memory hardware.

5.3 Communication or I/O drivers

If the flash memory is located in an external device, the access to this device shall be enacted via the corresponding communication respectively I/O driver.

6 Requirements traceability

Document: General Requirements on Basic Software Modules

Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Not applicable (this module does not provide any link-time parameters)
[BSW00404] Reference to post build time configuration	FLS014, FLS173, FLS174
[BSW00405] Reference to multiple configuration sets	FLS014, FLS173, FLS174
[BSW00345] Pre-compile-time configuration	FLS171, FLS172
[BSW159] Tool-based configuration	FLS179
[BSW167] Static configuration checking	FLS053
[BSW171] Configurability of optional functionality	FLS172, FLS183, FLS184, FLS185, FLS186, FLS187, FLS188
[BSW170] Data for reconfiguration of AUTOSAR SW-components	Not applicable (this module does not depend on faults, signal qualities, …)
[BSW00380] Separate C-File for configuration parameters	FLS159, FLS179
[BSW00419] Separate C-Files for pre-compile time configuration parameters	FLS179
[BSW00381] Separate configuration header file for pre-compile time parameters	FLS107
[BSW00412] Separate H-File for configuration parameters	FLS107
BSW00383] List dependencies of configuration files	FLS189
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Not applicable (this module does not provide any callback routines)
[BSW00388] Introduce containers	Chapter 10.2
[BSW00389] Containers shall have names	Chapter 10.2
[BSW00390] Parameter content shall be unique within the module	Chapter 10.2
[BSW00391] Parameter shall have unique names	Chapter 10.2
[BSW00392] Parameters shall have a type	Chapter 10.2
[BSW00393] Parameters shall have a range	Chapter 10.2
[BSW00394] Specify the scope of the parameters	Chapter 10.2
BSW00395] List the required parameters (per parameter)	Chapter 10.2
[BSW00396] Configuration classes	Chapter 10.2
[BSW00397] Pre-compile-time parameters	Chapter 10.2.2, Chapter 10.2.3
[BSW00398] Link-time parameters	Not applicable (this module does not provide any link-time parameters)
[BSW00399] Loadable Post-build time parameters	Chapter 10.2.3
[BSW00400] Selectable Post-build time parameters	Chapter 10.2.3
[BSW00402] Published information	Chapter 10.3
[BSW00375] Notification of wake-up reason	Not applicable (this module does not wake up the ECU)
[BSW101] Initialization interface	FLS014

Requirement	*Satisfied by*
[BSW00416] Sequence of Initialization	Not applicable (requirement on system architecture, not on a single module)
[BSW00406] Check module initialization	FLS017
[BSW168] Diagnostic Interface of SW components	Not applicable (no use case)
[BSW00407] Function to read out published parameters	Chapter 8.3.11
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (this module does not provide an AUTOSAR interface)
[BSW00424] BSW main processing function task allocation	Not applicable (requirement on system design, not on a single module)
[BSW00425] Trigger conditions for schedulable objects	Chapter 8.5
[BSW00426] Exclusive areas in BSW modules	Not applicable (this module does not provide any exclusive areas)
[BSW00427] ISR description for BSW modules	Not applicable (no ISR's defined for this module, usage of interrupts is implementation specific)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (this module does provide only one main processing function)
[BSW00429] Restricted BSW OS functionality access	Not applicable (requirement on the implementation, not for the specification)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (requirement on the BSW scheduler module)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	FLS190
[BSW00433] Calling of main processing functions	Not applicable (requirement on system design, not on a single module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (this module does not provide any exclusive areas)
[BSW00336] Shutdown interface	Not applicable (no use case).
[BSW00337] Classification of errors	FLS004, FLS007
[BSW00338] Detection and Reporting of development errors	FLS077
[BSW00369] Do not return development error codes via API	FLS164, FLS006
[BSW00339] Reporting of production relevant error status	Not applicable (this module only provides production relevant error events, no error status)
[BSW00421] Reporting of production relevant error events	FLS006, FLS104 , FLS105 , FLS106, FLS154
[BSW00422] Debouncing of production relevant error status	Not applicable (requirement on the DEM)
[BSW00420] Production relevant error event rate detection	Not applicable (requirement on the DEM)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (this is a BSW mdoule)

Requirement	*Satisfied by*
[BSW00323] API parameter checking	FLS015, FLS020, FLS021, FLS026, FLS027, FLS097, FLS098
[BSW004] Version check	FLS053
[BSW00409] Header files for production code error IDs	FLS160, FLS107
[BSW00385] List possible error notificatons	FLS004, FLS007
[BSW00386] Configuration for detecting an error	FLS077, FLS162, FLS163, FLS172
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00324] Do not use HIS I/O Library	Not applicable (architecture decision)
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00415] User dependent include files	Not applicable (only one user for this module)
[BSW164] Implementation of interrupt service routines	FLS193
[BSW00325] Runtime of interrupt service routines	FLS193
[BSW00326] Transition from ISRs to OS tasks	Not applicable (requirement on implementatio, not on specification)
[BSW00342] Usage of source code and object code	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00343] Specification and configuration of time	FLS178
[BSW160] Human-readable configuration data	Not applicable (requirement on documentation, not on specification)
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW00300] Module naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00413] Accessing instances of BSW modules	Conflict: This is currently not reflected in the driver's specification. This requirement will have impact on almost all BSW modules, therefore it can not be implemented within the Release 2.0 timeframe.
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (requirement on the implementation, not on the specification)
[BSW00305] Self-defined data types naming convention	Chapter 8.2
[BSW00307] Global variables naming convention	Not applicable (requirement on the implementation, not on the specification)
[BSW00310] API naming convention	Chapter 8.3
[BSW00373] Main processing function naming convention	Chapter 8.3.7
[BSW00327] Error values naming convention	FLS004, FLS007

Requirement	*Satisfied by*
[BSW00335] Status values naming convention	Chapter 8.1.2
[BSW00350] Development error detection keyword	FLS077, FLS162, FLS172
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00410] Compiler switches shall have defined values	Chapter 10.2
[BSW00411] Get version info keyword	Chapter 10.2.2
[BSW00346] Basic set of module files	FLS107
[BSW158] Separation of configuration from implementation	FLS107
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module does not implement any ISRs)
[BSW00370] Separation of callback interface from API	Not applicable (this module does not provide any callback routines)
[BSW00348] Standard type header	Not applicable (standard header files included via interface header file)
[BSW00353] Platform specific type header	Not applicable (standard header files included via interface header file)
[BSW00361] Compiler specific language extension header	Not applicable (standard header files included via interface header file)
[BSW00301] Limit imported information	FLS107
[BSW00302] Limit exported information	Not applicable (requirement on the implementation, not on the specification)
[BSW00328] Avoid duplication of code	Not applicable (requirement on the implementation, not on the specification)
[BSW00312] Shared code shall be reentrant	Not applicable (requirement on the implementation, not on the specification)
[BSW006] Platform independency	Not applicable (this is a module of the microcontroller abstraction layer)
[BSW00357] Standard API return type	Chapter 8.3.2, Chapter 8.3.3. Chapter 8.3.8, Chapter 8.3.9
[BSW00377] Module specific API return types	Chapter 8.3.5, Chapter 8.3.6
[BSW00304] AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (requirement on implementation, not for specification)
[BSW00378] AUTOSAR boolean type	Not applicable (requirement on implementation, not for specification)
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not for specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not for specification)

Requirement	*Satisfied by*
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not for specification)
[BSW00371] Do not pass function pointers via API	Not applicable (no function pointers in this specification)
[BSW00358] Return type of init() functions	Chapter 8.3.1
[BSW00414] Parameter of init function	Chapter 8.3.1, FLS194
[BSW00376] Return type and parameters of main processing functions	Chapter 8.3.7
[BSW00359] Return type of callback functions	Not applicable (this module does not provide any callback routines)
[BSW00360] Parameters of callback functions	Not applicable (this module does not provide any callback routines)
[BSW00329] Avoidance of generic interfaces	Chapter 8.3 (explicit interfaces defined)
[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not for specification)
[BSW00331] Separation of error and status values	FLS004, FLS164, FLS006
[BSW009] Module User Documentation	Not applicable (requirement on documentation, not on specification)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (all configuration parameters are single instance only)
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (no internal scheduling policy)
[BSW010] Memory resource documentation	Not applicable (requirement on documentation, not on specification)
[BSW00333] Documentation of callback function context	Not applicable (requirement on documentation, not for specifciation)
[BSW00374] Module vendor identification	FLS178
[BSW00379] Module identification	FLS178
[BSW003] Version identification	FLS178
[BSW00318] Format of module version numbers	FLS178
[BSW00321] Enumeration of module version numbers	Not applicable (requirement on implementation, not for specification)
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on documentation, not on specification)
[BSW00334] Provision of XML file	Not applicable (requirement on documentation, not on specification)

Document: General Requirements on SPAL

Requirement	***Satisfied by***
[BSW12263] Object code compatible configuration concept	FLS173, FLS174
[BSW12056] Configuration of notification mechanisms	FLS173, FLS174
[BSW12267] Configuration of wakeup sources	Not applicable (this module does not wake up the ECU / MCU)
[BSW12057] Driver module initialization	FLS014
[BSW12163] Driver module de-initialization	Not applicable (no use case)
[BSW12125] Initialization of hardware resources	FLS086
[BSW12461] Responsibility for register initialization	FLS086
[BSW12462] Provide settings for register initialization	Not applicable (requirement on documentation not on specification)
BSW12463] Combine and forward settings for register initialization	Not applicable (requirement on configuration, not on specification)
[BSW12068] MCAL initialization sequence	Not applicable (not a requirement for this driver but for system integration)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (the flash driver does not wake the ECU / MCU)
[BSW157] Notification mechanisms of drivers and handlers	Chapter 8.3.5, Chapter 8.6.3, FLS164, FLS006
[BSW12169] Control of operation mode	FLS155
[BSW12063] Raw value mode	Not applicable (the flash driver does not interpret the flash data)
[BSW12075] Use of application buffers	FLS002, FLS003
[BSW12129] Resetting of interrupt flags	FLS072
[BSW12064] Change of operation mode during running operation	Not applicable (the flash driver does not support different modes)
[BSW12448] Behavior after development error detection	FLS015, FLS020, FLS021, FLS026, FLS027, FLS097, FLS098
[BSW12067] Setting of wake-up conditions	Not applicable (the flash driver does not wake the ECU / MCU)
[BSW12077] Non-blocking implementation	Chapter 8.3.7
[BSW12078] Runtime and memory efficiency	Not applicable (requirement on implementation, not on specification)
[BSW12092] Access to drivers	Not applicable (requirement on system design, not on a single module)
[BSW12265] Configuration data shall be kept constant	FLS191
[BSW12264] Specification of configuration items	FLS172, FLS174

Document: Requirements on Flash Driver

Requirement	***Satisfied by***
[BSW12132] Flash driver static configuration	FLS048, FLS171
[BSW12133] Publication of flash properties	FLS177, FLS178
[BSW12134] Flash read function	FLS095, FLS096, FLS097, FLS098
[BSW12135] Flash write function	FLS024, FLS025, FLS026, FLS027
[BSW12136] Flash erase function	FLS018, FLS019, FLS020, FLS021
BSW13301 Flash compare function	FLS148, FLS149, FLS150, FLS151., FLS152, FLS153, FLS186
[BSW12137] Flash cancel function	FLS031, FLS183
[BSW12138] Flash driver status function	FLS034, FLS184
BSW13302 Flash driver mode selection function	FLS155, FLS156, FLS187
[BSW12159] Flash address check	FLS020, FLS021, FLS026, FLS027, FLS097, FLS098
[BSW12158] Flash blank check	FLS055
[BSW12141] Flash write verification	FLS056
[BSW12160] Flash erase verification	FLS022
[BSW12143] Flash driver job management	FLS016, FLS017, FLS023, FLS030, FLS032, FLS100
[BSW12144] Flash driver job processing function	FLS037, FLS038, FLS039, FLS190
BSW13303Job processing – normal mode	FLS040
BSW13304 Job processing – fast mode	FLS040
[BSW12193] Load flash access code to RAM on job start	FLS140, FLS141
[BSW12194] Execute flash access code from RAM	FLS142
BSW13300 Remove flash access code from RAM	FLS143
[BSW12147] Functional scope	FLS088
[BSW12182] External flash driver static configuration	FLS174
[BSW12107] Check Flash type	FLS144
[BSW12145] Flash driver job processing execution time	FLS040, FLS175, FLS181, FLS176, FLS182
[BSW12083] Use HIS specification as basis	Not applicable (the module provides comparable functionality but different API and different design rules)
[BSW12184] Limit read access blocking times	FLS040
[BSW12148] Common Flash API	FLS088
[BSW12149] Microcontroller independency	Not applicable (requirement on implementation, not on specification)

7 Functional specification

7.1 General design rules

FLS001: The flash driver shall offer asynchronous services for operations on flash memory (read/erase/write).

FLS002: The flash driver shall not buffer data. It shall use application data buffers that are referenced by a pointer passed via the API.

FLS003: The flash driver shall not ensure data consistency of the given application buffer. It is the responsibility of the application to ensure consistency of flash data during a flash read or write operation.

FLS053: onfiguration parameters shall be checked statically (at the latest during compile time) for correctness. The version information in the module header and source files shall be validated and consistent (e.g. by comparing the version information in the module header and source files with a pre-processor macro).

FLS069: This flash driver offers a superset of the HIS flash driver in order to allow for usage within a Flash-EEPROM-Emulation (i.e. cancel function, read function). It does not specify the same API because of problems with standardization of the HIS API.

FLS118: If the flash memory consists of several separate flash memory areas, this shall be handled internally by the flash driver. Therefore the flash driver shall combine all available flash memory areas into one linear address space (denoted by the parameters `FLS_BASE_ADDRESS` and `FLS_TOTAL_SIZE`). The address and length parameters for the read, write and erase functions thus become “virtual” addresses which the flash driver has to map to the physical addresses according to the physical structure of the flash memory areas. As long as the restrictions regarding the alignment of those addresses are met it is allowed that a read, write or erase job crosses the boundaries of a physical flash memory area.

7.2 Error classification

FLS160: Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.

FLS161: Development error values are of type uint8.

FLS004: The flash driver shall be able to detect the following errors and exceptions depending on its configuration (development/production):

Type or error	*Relevance*	*Related error code*	*Value [hex]*
API service called with wrong parameter	Development	FLS_E_PARAM_CONFIG FLS_E_PARAM_ADDRESS FLS_E_PARAM_LENGTH FLS_E_PARAM_DATA	0x01 0x02 0x03 0x04
API service called without module initialization	Development	FLS_E_UNINIT	0x05
API service called while driver still busy	Development	FLS_E_BUSY	0x05
Erase verification (blank check) failed	Development	FLS_E_VERIFY_ERASE_FAILED	0x07
Write verification (compare) failed	Development	FLS_E_VERIFY_WRITE_FAILED	0x08
Flash erase failed (HW)	Production	FLS_E_ERASE_FAILED	Assigned by DEM
Flash write failed (HW)	Production	FLS_E_WRITE_FAILED	Assigned by DEM
Flash read failed (HW)	Production	FLS_E_READ_FAILED	Assigned by DEM
Flash compare failed (HW)	Production	FLS_E_COMPARE_FAILED	Assigned by DEM
Expected hardware ID not matched (see [FLS144])	Production	FLS_E_UNEXPECTED_FLASH_ID	Assigned by DEM

7.3 Error detection

FLS077: The detection of all development errors shall be configurable (on/off) at pre.compile time. The preprocessor switch *FLS_DEV_ERROR_DETECT* (see chapter 10) shall activate or deactivate the detection of all development errors.

FLS162: If the FLS_DEV_ERROR_DETECT switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.2 and chapter 8.3.

FLS163: The detection of production code errors cannot be switched off.

7.4 Error notification

FLS164: Detected development errors shall be reported to the Development Error Tracer (DET) if the pre-processor switch FLS_DEV_ERROR_DETECT is set (see chapter 10). The error codes shall not be used as return values of the called function.

FLS006: Production relevant errors shall be reported to the Diagnostic Event Manager (DEM). The error codes shall not be used as return values of the called function.

FLS007: Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the flash driver's

implementation documentation. The classification and enumeration shall be compatible with the errors listed above [FLS004].

7.5 External flash driver

FLS144: During initialization of the external flash driver, the hardware ID of the external flash device shall be checked against the corresponding published parameter. A mismatch shall be reported to the Diagnostic Event Manager (DEM) with the error code `FLS_E_UNEXPECTED_FLASH_ID`. In this case the flash driver shall not be initialized, i.e. the flash driver's status shall be set to `FLS_E_UNINIT`.

FLS189: A complete list of required parameters is specified in the SPI Handler/Driver Software Specification (Chapter Configuration Specification, marked as "SPI User").

7.6 Loading, executing and removing the flash access code

Technical background information: Flash technology or flash memory segmentation may require that the routines that access the flash hardware (internal erase and write routines) are executed from RAM because reading the flash - for instruction fetch needed for code execution - is not allowed while programming the flash.

FLS137: The code of the flash access routines shall be placed into a separate C-module Fls_ac.c.

FLS139: The flash access routines have to ensure that they can not be interrupted, so interrupts have to be disabled during their execution. Therefore execution time for the flash access code has to be kept as short as possible.

FLS140: If the flash driver is configured to load the flash access code to RAM on job start, the flash drivers erase routine shall load the flash access code for erasing the flash memory to the location in RAM pointed to by the erase function pointer contained in the flash drivers configuration set.

FLS141: If the flash driver is configured to load the flash access code to RAM on job start, the flash drivers write routine shall load the flash access code for writing the flash memory to the location in RAM pointed to by the write function pointer contained in the flash drivers configuration set.

FLS142: The flash driver's main processing routine shall execute the flash access code routines. Access to the routines shall be done by means of the respective function pointer contained in the flash driver's configuration set (post-compile parameters) regardless whether the flash access code routines have been loaded to RAM or whether they can be executed directly from (flash) ROM.

FLS143: After an erase or write job has been finished or canceled the flash drivers main processing routine shall unload (i.e. overwrite) the flash access code (internal erase / write routines) from RAM if they have been loaded to RAM by the flash driver.

FLS195: The access code shall only be loaded to RAM if necessary that is if that code cannot be executed out of flash ROM. Also blocking (disabling interrupts and waiting for the completion of the erase / write command) shall only be done if necessary that is if it has to be ensured that no other code is executed in the meantime.

8 API specification

8.1 Imported types

8.1.1 Standard types

In this chapter all types included from the following files are listed:

- Std_Types.h

- Std_ReturnType
- Std_VersionInfoType

8.1.2 MemIf types

In this chapter all types included from module MemIf are listed.

- MemIf_ModeType
- MemIf_StatusType
- MemIf_JobResultType

8.2 Type definitions

8.2.1 Fls_ConfigType

Fls_configType

Type:	Struct	
Range:	`Hardware dependend structure`	Structure to hold the flash driver configuration set. The contents of the initialisation data structure are specific to the flash memory hardware.
Description:	A pointer to such a structure is provided to the flash driver initialization routine for configuration of the driver and flash memory hardware.	

8.2.2 Fls_AddressType

Type:	uint8, uint16, uint32	
Range:	`8 / 16 / 32 bits`	Size depends on target platform and flash device.
Description:	Used as address offset from the configured flash base address to access a certain flash memory area. **FLS197:** Each flash device shall have 0 as lower limit for this address offset, a device specific base address shall be added by the driver if necessary.	

8.2.3 Fls_LengthType

Type:	Fls_AddressType	
Range:	Same as Fls_AddressType	Shall be the same type as Fls_AddressType because of arithmetic operations. Size depends on target platform and flash device.
Description:	Specifies the number of bytes to read/write/erase/compare.	

8.3 Function definitions

8.3.1 Fls_Init

Fls_Init

Service name:	Fls_Init
Syntax:	`void Fls_Init ( const Fls_ConfigType *ConfigPtr )`
Service ID [hex]:	0x00
Behaviour:	Synchronous
Re-entrancy	Non re-entrant
Parameters (in):	`ConfigPtr` Pointer to flash driver configuration set.
Parameters (out):	None --
Return value:	None
Description:	**FLS014:** This function initializes the flash driver (software) and all flash memory relevant registers (hardware) with parameters provided in the given configuration set. **FLS191:** This routine shall store the pointer to the given configuration set in a local variable in order to allow the module access to the configuration set contents during runtime. **FLS086:** The routine shall initialize all module global variables and those controller registers that are needed for controlling the flash device and that do not influence or depend on other (hardware) modules. Registers that can influence or depend on other modules shall be initialized by a common system module. **FLS015:** If development error detection is enabled the (hardware specific) contents of the given configuration set shall be checked for being within the allowed range. Errors shall be reported to the Debug Error Tracer (DET) with the error value `FLS_E_PARAM_CONFIG`. **FLS016:** After having finished the module initialization the flash driver state shall be set to `MEMIF_IDLE` and the flash job result shall be set to `MEMIF_JOB_OK`. **FLS017:** If development error detection is enabled the routine shall check that the flash driver and hardware are not yet initialized (driver state `MEMIF_UNINIT`) or the driver is currently busy (`MEMIF_BUSY`). Errors shall be reported to the Development Error Tracer (DET) with the corresponding error value (`FLS_E_UININIT` respectively `FLS_E_BUSY`). **FLS048:** If supported by hardware, this routine sets the flash memory erase/write

	protection as provided in the configuration set.
Caveats:	None
Configuration:	None

8.3.2 Fls_Erase

Fls_Erase

Service name:	Fls_Erase	
Syntax:	`Std_ReturnType Fls_Erase ( Fls_AddressType TargetAddress, Fls_LengthType Length )`	
Service ID [hex]:	0x01	
Behaviour:	Asynchronous	
Re-entrancy:	Non re-entrant	
Parameters (in):	TargetAddress	Target address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: `FLS_SIZE` – 1
	Length	Number of bytes to erase Min.: 1 Max.: `FLS_SIZE – TargetAddress`
Parameters (out):	None	--
Return value:	`E_OK`: erase command has been accepted `E_NOT_OK`: erase command has not been accepted	
Description:	**FLS018:** Service for erasing one or more complete flash sectors. This service shall copy the given parameters to driver internal variables, initiate an erase job, set the driver status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return with `E_OK`. **FLS019:** The erase job shall be executed asynchronously within the flash driver's main function. A flash memory block starting from `FLS_BASE_ADDRESS` + `TargetAddress` of size `Length` shall be erased. Note: `Length` will be rounded up to the next full sector boundary, since only complete flash sectors can be erased. **FLS020:** If development error detection is enabled the routine shall check that the erase start address (`FLS_BASE_ADDRESS` + `TargetAddress`) is aligned to a flash sector boundary and that it lies within the specified lower and upper flash address boundaries. If not, the erase job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_ADDRESS`. **FLS021:** If development error detection is enabled the routine shall check that the erase length is greater than 0 and that the erase end address (erase start address + length) is aligned to a flash sector boundary and that it lies within the specified upper flash address boundary. If not, the erase job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_LENGTH`. **FLS065:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the erase request shall be rejected with the	

	return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_UNINIT`. **FLS023:** If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the erase request shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_BUSY`. **FLS145:** If possible, e.g. with interrupt controlled implementations, the first round of the erase job shall be started directly form this routine to reduce overall runtime.
Caveats:	- The flash driver shall have been initialized before this service is called. - Only one read, write, erase or compare job can be accepted at a time.
Configuration:	None

8.3.3 Fls_Write

Fls_Write

Service name:	Fls_Write	
Syntax:	`Std_ReturnType Fls_Write ( Fls_AddressType TargetAddress, const uint8 *SourceAddressPtr, Fls_LengthType Length )`	
Service ID [hex]:	0x02	
Behaviour:	Asynchronous	
Re-entrancy:	Non re-entrant	
	`TargetAddress`	Target address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: `FLS_SIZE` – 1
	`SourceAddressPtr`	Pointer to source data buffer
	`Length`	Number of bytes to write Min.: 1 Max.: `FLS_SIZE – TargetAddress`
Parameters (out):	None	--
Return value:	`E_OK`: write command has been accepted `E_NOT_OK`: write command has not been accepted	
Description:	**FLS024:** Service for writing one or more complete flash pages. This service shall copy the given parameters to driver internal variables, initiate a write job, set the driver status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return with `E_OK`. **FLS025:** The write job shall be executed asynchronously within the flash driver's main function. A flash memory block starting from `FLS_BASE_ADDRESS` + `TargetAddress` of size `Length` shall be programmed with the data provided via `SourceAddressPtr`. **FLS026:** If development error detection is enabled the routine shall check that the write start address (`FLS_BASE_ADDRESS + TargetAddress`) is aligned to a flash page boundary and that it lies within the specified lower and upper flash address boundaries. If not, the write job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_ADDRESS`. **FLS027:** If development error detection is enabled the routine shall check that the write length is greater than 0 and that the write end address (write start address + length) is aligned to a flash page boundary and that it lies within the specified upper flash address boundary. If not, the write job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_LENGTH`. **FLS066:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the write request shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_UNINIT`. **FLS030:** If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the write request shall be rejected with the return	

	value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_BUSY. **FLS157:** If development error detection is enabled, the routine shall check the given data buffer pointer for not being a NULL pointer. If this error is encountered the write request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_DATA. **FLS146:** If possible, e.g. with interrupt controlled implementations, the first round of the write job shall be started directly form this routine to reduce overall runtime.
Caveats:	- The flash driver shall have been initialized before this service is called. - Only one read, write, erase or compare job can be accepted at a time. - The flash memory area shall have been erased before.
Configuration:	None

8.3.4 Fls_Cancel

Fls_Cancel

Service name:	Fls_Cancel

Syntax:	`void Fls_Cancel`
	`(`
	`  void`
	`)`

Service ID [hex]:	0x03
Behaviour:	Synchronous
Re-entrancy:	Non re-entrant
Parameters (in):	None --
Parameters (out):	None --
Return value:	None
Description:	**FLS031:** Service for canceling an ongoing flash read, write, erase or compare job. This function shall abort a running job synchronously so that directly after returning from this function a new job can be started. **FLS032:** The function shall reset the driver's internal job processing variables (like address, length and data pointer) and set the driver state to FLS_IDLE. **FLS033:** The routine shall set the job result to MEMIF_JOB_CANCELED, if the job result currently has the value MEMIF_JOB_PENDING. Otherwise it shall leave the job result unchanged. **FLS147:** If configured, the routine shall call the error notification function to inform the caller about the cancellation of the job.
Caveats:	- The states and data of the affected flash memory cells are undefined. - This function must only be called from one source (e.g. Flash-EEPROM-Emulation).
Configuration:	**FLS183:** This function shall be configurable at pre-compile time using the parameter FLS_CANCEL_API

8.3.5 Fls_GetStatus

Fls_GetStatus

Service name:	Fls_GetStatus
Syntax:	`MemIf_StatusType Fls_GetStatus ( void )`
Service ID [hex]:	0x04
Behaviour:	Synchronous
Re-entrancy:	Re-entrant
Parameters (in):	None --
Parameters (out):	None --
Return value:	MemIf_StatusType
Description:	**FLS034:** This service shall return the driver state synchronously.
Caveats:	None
Configuration:	**FLS184:** This function shall be configurable at pre-compile time using the parameter `FLS_GET_STATUS_API`

8.3.6 Fls_GetJobResult

Fls_GetJobResult

Service name:	Fls_GetJobResult
Syntax:	`MemIf_JobResultType Fls_GetJobResult ( void )`
Service ID [hex]:	0x05
Behaviour:	Synchronous
Re-entrancy:	Re-entrant
Parameters (in):	None --
Parameters (out):	None --
Return value:	MemIf_JobResultType
Description:	**FLS035:** This service shall return the result of the last job synchronously. **FLS036:** The erase, write, read and compare services share the same job result, i.e. only the result of the last job can be queried. Every new job that has been accepted by the flash driver overwrites the job result with `MEMIF_JOB_PENDING`.
Caveats:	None
Configuration:	**FLS185:** This function shall be configurable at pre-compile time using the parameter `FLS_GET_JOB_RESULT_API`

8.3.7 Fls_MainFunction

Fls_MainFunction

Service name:	Fls_MainFunction
Syntax:	`void Fls_MainFunction ( void )`
Service ID [hex]:	0x06
Behaviour:	Synchronous
Re-entrancy:	Non re-entrant
Parameters (in):	None --
Parameters (out):	None --
Return value:	None
Description:	**FLS037:** This function shall perform the processing of the flash read, write, erase and compare jobs. **FLS038:** When a job has been initiated, this function has to be called cyclically until the job is finished . **FLS039:** The function might also be called cyclically if no job is pending. In this case the function shall return without any action. **FLS040:** The flash driver main function shall only process as much data in one call cycle as statically configured for the current job type (read, write, erase or compare) and the current operating mode (normal, fas). **FLS104:** If a flash erase job fails due to a hardware error, the job result shall be set to `MEMIF_JOB_FAILED` and an error shall be reported to the DEM with the error code `FLS_E_ERASE_FAILED`. **FLS105:** If a flash write job fails due to a hardware error, the job result shall be set to `MEMIF_JOB_FAILED` and an error shall be reported to the DEM with the error code `FLS_E_WRITE_FAILED`. **FLS106:** If a flash read job fails due to a hardware error, the job result shall be set to `MEMIF_JOB_FAILED` and an error shall be reported to the DEM with the error code `FLS_E_READ_FAILED`. **FLS154:** If a flash compare job fails due to a hardware error, the job result shall be set to `MEMIF_JOB_FAILED` and an error shall be reported to the DEM with the error code `FLS_E_COMPARE_FAILED`. **FLS200:** If a flash compare job yields differences within the compared memory blocks, the job result shall be set to `MEMIF_BLOCK_INCONSISTENT`. **FLS022:** After erasing a flash block, if development error detection is enabled, the routine shall compare the contents of the addressed memory area against the value of an erased flash cell to check whether the block has been completely erased. If not, the job result shall be set to `MEMIF_JOB_FAILED` and an error shall be reported to the DET with the error code `FLS_E_VERIFY_ERASE_FAILED`. **FLS055:** Before writing a flash block, if development error detection is enabled, the routine shall compare the contents of the addressed memory area against the value of an erased flash cell to check whether the block has been completely

	erased. If not, the write job shall be aborted, the job result shall be set to MEMIF_JOB_FAILED and the error shall be reported to the DET with the error code FLS_E_VERIFY_ERASE_FAILED. **FLS056:** After writing a flash block, if development error detection is enabled, the routine shall compare the contents of the reprogrammed memory area against the contents of the provided application buffer to check whether the block has been completely reprogrammed. If not, the job result shall be set to MEMIF_JOB_FAILED and an error shall be reported to the DET with the error code FLS_E_VERIFY_WRITE_FAILED. **FLS052:** After a read, erase, write or compare job has been finished, the function shall set the job result to MEMIF_JOB_OK if it is currently MEMIF_JOB_PENDING else it shall leave the result unchanged. Also it shall set the driver state to MEMIF_IDLE and call the job end notification function if configured [FLS103]. **FLS072:** The configuration parameter FLS_USE_INTERRUPTS shall switch between interrupt and polling controlled job processing if this is supported by the flash memory hardware. If interrupt controlled job processing is supported and enabled, the interrupt service routine located in Fls_Irq.c shall reset the interrupt flag, check for errors reported by the underlying hardware, reload the hardware finite state machine for the next round of the pending job or call the appropriate notification routine if the job is finished or aborted. The function Fls_MainFunction is still required for processing of jobs without hardware interrupt support (e.g. read jobs) and for timeout supervision. **FLS117:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not an error shall be reported to the DET with the error code FLS_E_UNINIT. **FLS196:** In each cycle of this function at the most one flash sector shall be erased.
Caveats:	None
Configuration:	None

8.3.8 Fls_Read

Fls_Read

Service name:	Fls_Read	
Syntax:	`Std_ReturnType Fls_Read ( Fls_AddressType SourceAddress, uint8 *TargetAddressPtr, Fls_LengthType Length )`	
Service ID [hex]:	0x07	
Behaviour:	Asynchronous	
Re-entrancy:	Non re-entrant	
Parameters (in):	SourceAddress	Source address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE – 1

	Length	Number of bytes to read Min.: 1 Max.: FLS_SIZE – SourceAddress
Parameters (out):	TargetAddressPtr	Pointer to target data buffer
Return value:	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted	
Description:	**FLS095:** Service for reading from flash memory. This service shall copy the given parameters to driver internal variables, initiate a read job, set the driver status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK. **FLS096:** The read job shall be executed asynchronously within the flash driver's main function. A continuous flash memory block starting from FLS_BASE_ADDRESS + SourceAddress of size Length shall be copied to the buffer pointed to by TargetAddressPtr. **FLS097:** If development error detection is enabled the routine shall check that the read start address (FLS_BASE_ADDRESS + SourceAddress) lies within the specified lower and upper flash address boundaries. If not, the read job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_ADDRESS. **FLS098:** If development error detection is enabled the routine shall check that the read length is greater than 0 and that the read end address (read start address + length) lies within the specified upper flash address boundary. If not, the read job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_LENGTH. **FLS099:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the read request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_UNINIT. **FLS100:** If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the read request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_BUSY. **FLS158:** If development error detection is enabled, the routine shall check the given data buffer pointer for not being a NULL pointer. If this error is encountered the write request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_DATA.	
Caveats:	- The flash driver shall have been initialized before this service is called. - Only one read, write or erase job can be accepted at a time.	
Configuration:	None	

8.3.9 Fls_Compare

Service name:	Fls_Compare

Syntax:	`Std_ReturnType Fls_Compare`
	`(`
	`  Fls_AddressType  SourceAddress,`
	`  const uint8  *TargetAddressPtr,`
	`  Fls_LengthType  Length`
	`)`

Service ID [hex]:	0x08

Behaviour:	Asynchronous	
Re-entrancy:	Non re-entrant	
Parameters (in):	`SourceAddress`	Source address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: `FLS_SIZE` – 1
	`TargetAddressPtr`	Pointer to target data buffer
	`Length`	Number of bytes to compare Min.: 1 Max.: `FLS_SIZE – SourceAddress`
Parameters (out):	None	--
Return value:	`E_OK`: compare command has been accepted `E_NOT_OK`: compare command has not been accepted	
Description:	**FLS148:** Service for comparing the contents of an area of flash memory with that of an application data buffer. This service shall copy the given parameters to driver internal variables, initiate a compare job, set the driver status to `MEMIF_BUSY`, set the job result to `MEMIF_JOB_PENDING` and return with `E_OK`. **FLS149:** The compare job shall be executed asynchronously within the flash driver's main function. A continuous flash memory block starting from `FLS_BASE_ADDRESS` + `SourceAddress` of size `Length` shall be compared to the buffer pointed to by `TargetAddressPtr`. **FLS150:** If development error detection is enabled the routine shall check that the compare start address (`FLS_BASE_ADDRESS` + `SourceAddress`) lies within the specified lower and upper flash address boundaries. If not, the compare job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_ADDRESS`. **FLS151:** If development error detection is enabled the routine shall check that the given length is greater than 0 and that the compare end address (compare start address + length) lies within the specified upper flash address boundary. If not, the compare job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_LENGTH`. **FLS152:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the compare request shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_UNINIT`. **FLS153:** If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the compare request shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_BUSY`.	
Caveats:	- The flash driver shall have been initialized before this service is called. - Only one read, write, erase or compare job can be accepted at a time.	
Configuration:	**FLS186:** This function shall be configurable at pre-compile time using the parameter `FLS_COMPARE_API`	

8.3.10 Fls_SetMode

Service name:	Fls_SetMode

Syntax:	`void Fls_SetMode`
	`(`
	`  MemIf_ModeType Mode`

	)	
Service ID [hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	non reentrant	
Parameters (in):	`Mode`	`MEMIF_MODE_SLOW`: Slow read access / normal SPI access. `MEMIF_MODE_FAST`: Fast read access / SPI burst access.
Parameters (out):	None	--
Return value:	None	--
Description:	**FLS155:** This service shall set the flash driver's operation mode to the given "Mode" parameter. **FLS156:** If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the mode change request shall be rejected and the error shall be reported to the DET with the error code `FLS_E_BUSY`.	
Caveats:	This service shall not be called during a running operation.	
Configuration:	**FLS187:** This function shall be configurable at pre-compile time using the parameter `FLS_SET_MODE_API`	

8.3.11 Fls_GetVersionInfo

Service name:	Fls_ GetVersionInfo	

Syntax:	`void Fls_GetVersionInfo`
	`(`
	`  Std_VersionInfoType *VersioninfoPtr`
	`)`	

Service ID [hex]:	0x10	
Sync/Async:	Synchronous	
Re-entrancy:	Non re-entrant	
Parameters (in):	none	--
Parameters (out):	`VersioninfoPtr`	Pointer to where to store the version information of this module.
Return value:	none	--
Description:	**FLS165:** This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). **FLS166:** This function shall be pre compile time configurable `On/Off` by the configuration parameter: `<MODULE PREFIX>_VERSION_INFO_API` Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.	
Caveats:	None	
Configuration:	**FLS188:** This function shall be configurable at pre-compile time using the parameter `FLS_VERSION_INFO_API`	

8.4 Call-back notifications

FLS193: Depending on implementation, callback routines provided and/or invoked by this module may be called on interrupt level. The module providing those routines therefore has to make sure that their runtime is reasonably short, i.e. since callbacks may be propagated upward through several software layers.

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

See chapter 8.3.7

FLS190: This module shall only provide one scheduled function, since reading from / writing to flash memory cannot usually be done simultaneously and the overhead for synchronizing the two would outweigh the benefits.

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

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

API function	*Module*	*Description*
Dem_ReportErrorStatus	Dem	Routine to report production relevant error events by event ID.

Note: If the flash device is connected via SPI, also the SPI interfaces are required to fulfill the modules core functionality. Which interfaces are needed exactly shall not be detailed further in this specification.

8.6.2 Optional Interfaces

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

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
Det_ReportError	Det	Development error notification	FLS_DEV_ERROR_DETECT

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The names of these kind of interfaces is not fixed because they are configurable.

FLS109: The job processing callback notifications shall be configurable as function pointers within the initialization data structure (`Fls_ConfigType`).

FLS110: The callback notifications shall have no parameters and no return value.

FLS111: If a job processing callback notification is configured as null pointer, the corresponding callback routine shall not be executed.

Name:	Fee_JobEndNotification

Syntax:	`void Fee_JobEndNotification`
	`(`
	`  void`
	`)`

Reentrancy:	Don't care
Parameters (in):	None --
Parameters (out):	None --
Return value:	None --
Description:	**FLS167:** This callback function shall be called when a job has been completed with a positive result: • Read job finished & OK • Write job finished & OK • Erase job finished & OK • Compare job finished & memory blocks are the same
Caveats:	None
Configuration:	None

Name:	Fee_JobErrorNotification

Syntax:	`void Fee_JobErrorNotification`
	`(`
	`  void`
	`)`

Reentrancy:	Don't care
Parameters (in):	None --
Parameters (out):	None --
Return value:	None --
Description:	**FLS168:** This callback function shall be called when a job has been cancelled or finished with negative result: • Read job aborted or failed • Write job aborted or failed • Erase job aborted or failed • Compare job aborted or failed • Compare job finished and memory blocks differ
Caveats:	None
Configuration:	None

9 Sequence diagrams

9.1 Initialization

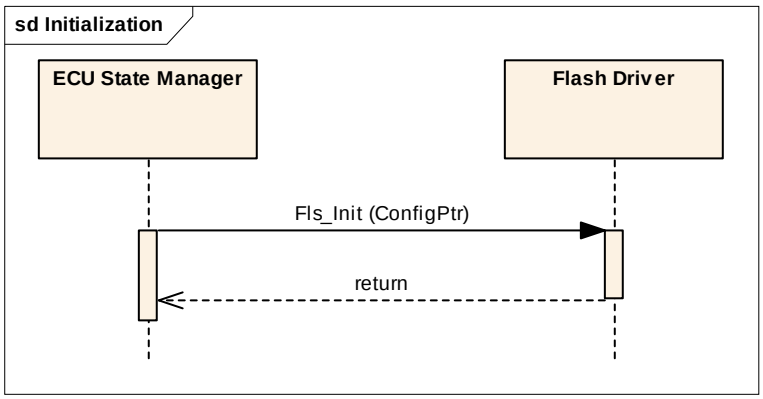

Figure 2: Flash driver initialization sequence

9.2 Synchronous functions

The following sequence diagram shows the function Fls_GetJobResult as an example for the synchronous functions of this module. The same sequence applies also to the functions Fls_GetStatus and Fls_SetMode.

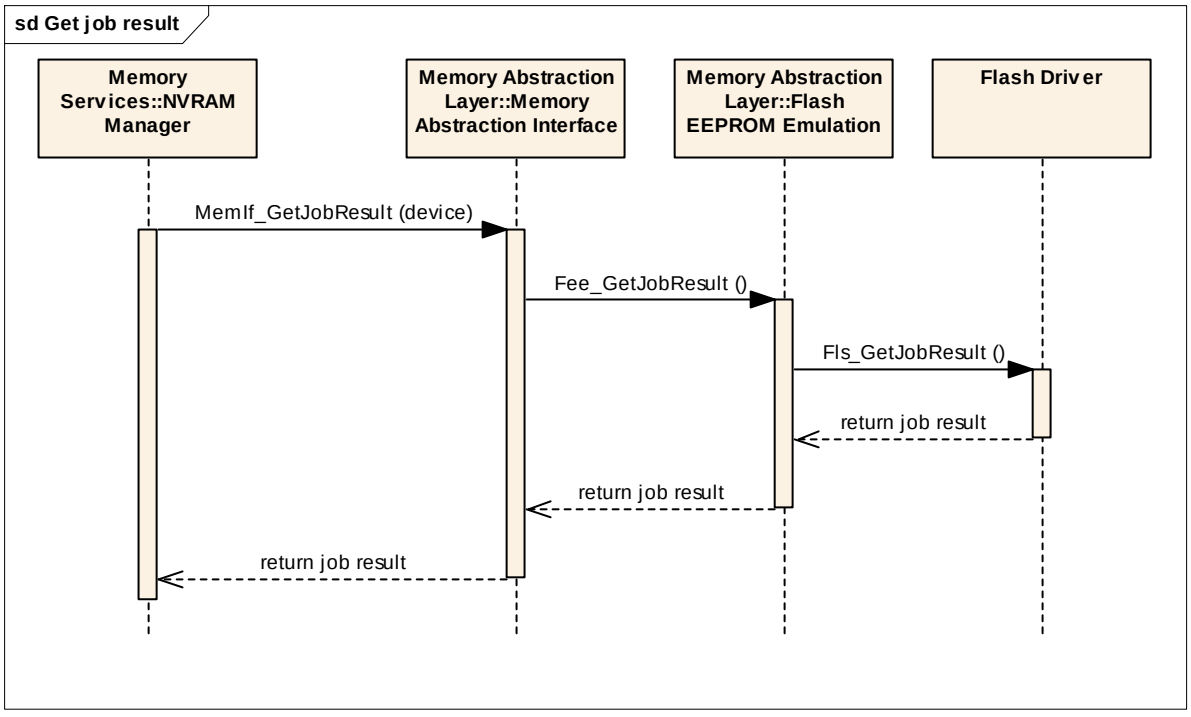

Figure 3: Fls_GetJobResult

9.3 Asynchronous functions

The following sequence diagram shows the flash write function (with the configuration option FLS_AC_LOAD_ON_JOB_START set) as an example for the asynchronous functions of this module. The same sequence applies to the erase, read and compare jobs, with the only difference that for the read and compare jobs no flash access code needs to be loaded to / unloaded from RAM.

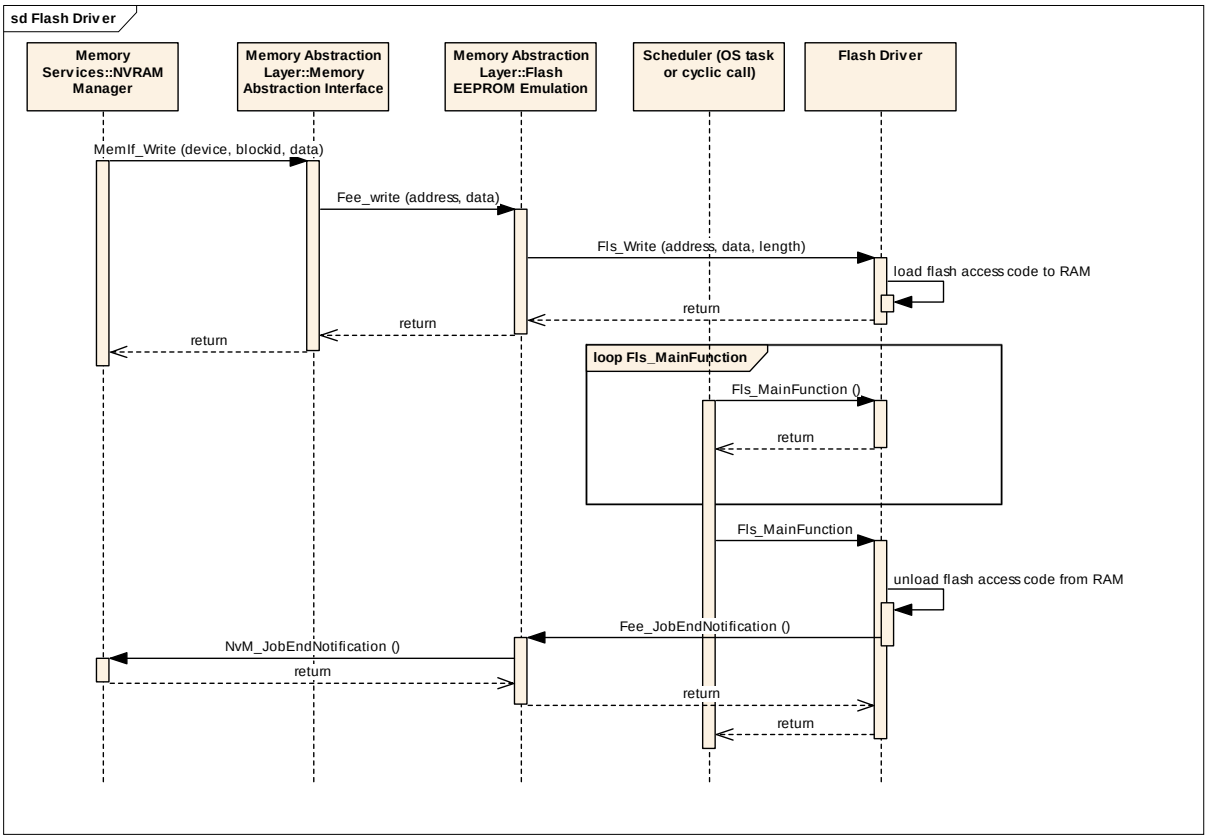

Figure 4: Flash write sequence, flash access code loaded on job start

9.4 Canceling a running job

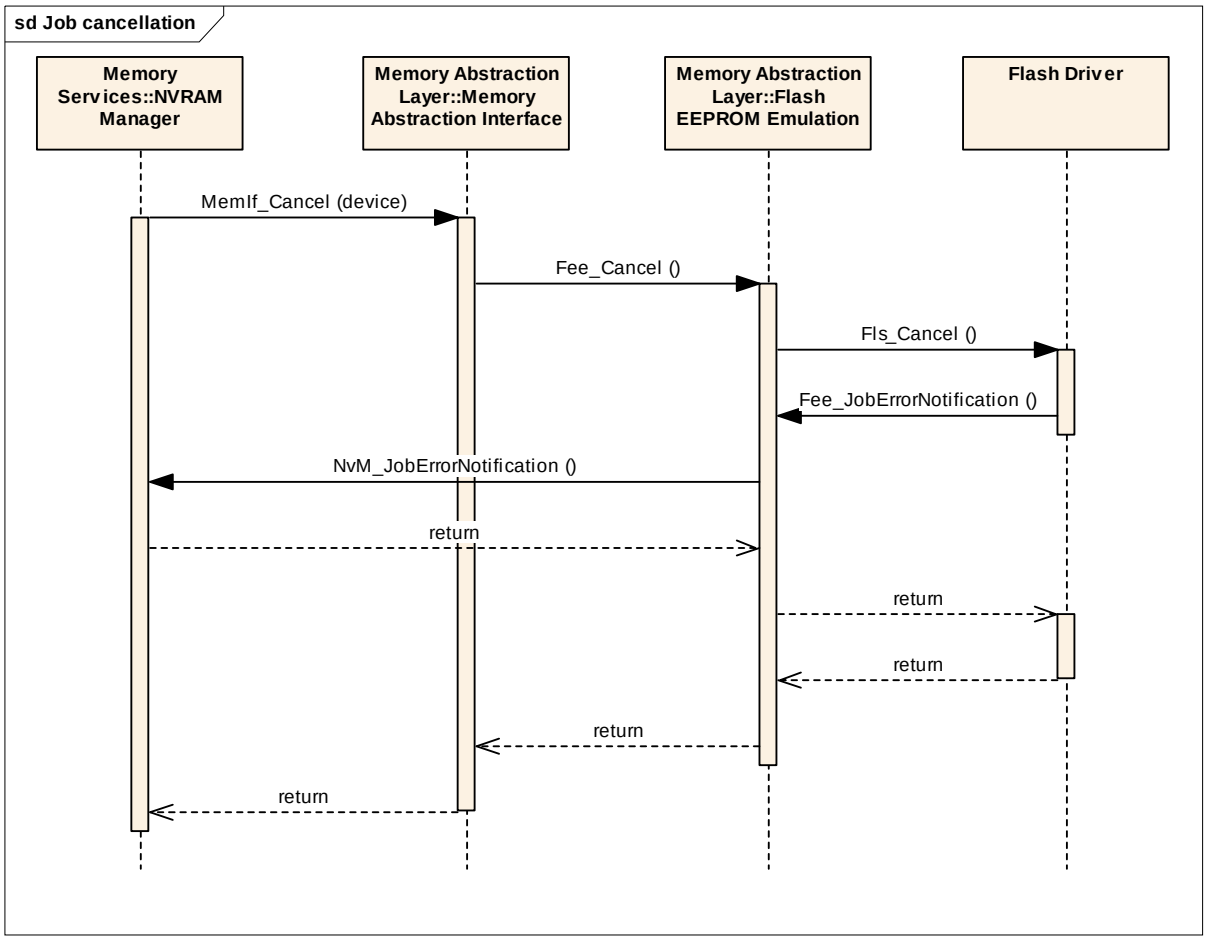

Figure 5: Canceling a running flash job

FLS049: The `Fls_Cancel()` function shall not be called during a running `Fls_MainFunction()` function.

This can be achieved by one of the following scheduling configurations:

- Possibility 1: The job functions of the NVRAM manager and the flash driver are synchronized (e.g. called sequentially within one task)
- Possibility 2: The task that calls the `Fls_MainFunction()` function can not be preempted by another task.

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 Flash Driver.

Chapter 10.3 specifies published information of the module <Module Name>.

10.1 How to read this chapter

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

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

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

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	*Description*
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	*Description*
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	*Description*
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* – the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* – the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 10.2 and Chapter 10.3.

10.2.1 Variants

Variant PC: Only pre-compile time parameters

Variant PB: Fls_ConfigSet (see FLS174) as post build time configurable

FLS194: The initialization function of this module shall always have a pointer as a parameter, even though for Variant PC no configuration set shall be given. Instead a NULL pointer shall be passed to the initialization function. This means that, in

contradiction to BSW00414 only one interface for initialization shall be implemented and it shall not depend on the modules configuration, which interface the calling software module shall use.

10.2.2 Fls_ModuleConfiguration

The following table specifies parameters that shall be configured during system generation. These parameters shall be located in the file Fls_Cfg.h. Further hardware or implementation specific parameters can be added if necessary.

SWS Item	**FLS172:**
Container Name	Fls_ModuleConfiguration
Description	Configuration parameters for module flash driver.
Configuration Parameters	

Name	FLS_DEV_ERROR_DETECT		
Description	Pre-processor switch for enabling the development error detection and reporting (see FLS077).		
Type	#define		
Unit	--		
Range	ON	Development error detection and reporting enabled	
	OFF	Development error detection and reporting disabled	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_USE_INTERRUPTS		
Description	Job processing triggered by hardware interrupt		
Type	#define		
Unit	--		
Range	ON	Job processing triggered by interrupt (hardware controlled)	
	OFF	Job processing not triggered by interrupt (software controlled)	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	Only available if supported by underlying flash hardware		

Name	FLS_BASE_ADDRESS		
Description	The flash memory start address (see also FLS118). **FLS169:** This parameter defines the lower boundary for read / write / erase and compare jobs.		
Type	#define		
Unit	--		
Range	--	Hardware specific	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--

Scope	Module		
Dependency	None		

Name	FLS_TOTAL_SIZE		
Description	The total amount of flash memory in bytes (see also FLS118). **FLS170:** This parameter in conjunction with FLS_BASE_ADDRESS defines the upper boundary for read / write / erase and compare jobs.		
Type	#define		
Unit	--		
Range	--	Hardware specific	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_AC_LOAD_ON_JOB_START		
Description	The flash driver shall load the flash access code to RAM whenever an erase or write job is started and unload (overwrite) it after that job has been finished or canceled.		
Type	#define		
Unit	--		
Range	ON	Flash access code loaded on job start / unloaded on job end or error.	
	OFF	Flash access code not loaded to / unloaded from RAM at all.	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_VERSION_INFO_API		
Description	Compile switch to enable / disable the version information function.		
Type	#define		
Unit	--		
Range	ON	API supported / function provided	
	OFF	API not supported / function not provided	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_CANCEL_API		
Description	Compile switch to enable / disable the Fls_Cancel function.		
Type	#define		
Unit	--		
Range	ON	API supported / function provided	
	OFF	API not supported / function not provided	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_COMPARE_API		
Description	Compile switch to enable / disable the Fls_Compare function.		
Type	#define		

Unit	--		
Range	ON	API supported / function provided	
	OFF	API not supported / function not provided	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_SET_MODE_API		
Description	Compile switch to enable / disable the Fls_SetMode function.		
Type	#define		
Unit	--		
Range	ON	API supported / function provided	
	OFF	API not supported / function not provided	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_GET_STATUS_API		
Description	Compile switch to enable / disable the Fls_GetStatus function.		
Type	#define		
Unit	--		
Range	ON	API supported / function provided	
	OFF	API not supported / function not provided	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Name	FLS_GET_JOB_RESULT_API		
Description	Compile switch to enable / disable the Fls_GetJobResult function.		
Type	#define		
Unit	--		
Range	ON	API supported / function provided	
	OFF	API not supported / function not provided	
Configuration Class	***Pre-compile***	X	All variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
Fls_ConfigSet	1..*	Module initialization

10.2.3 Fls_ConfigSet

FLS173: The following table specifies the parameters that shall be located in an external data structure of type Fls_ConfigType. The organization and location of this data structure shall be up to the implementer. The type declaration shall be located in

the file Fls.h. Further hardware or implementation specific parameters can be added if necessary.

SWS Item	**FLS174:**
Container Name	Fls_ConfigSet
Description	Configuration parameters for module initialization.
Configuration Parameters	

Name	FLS_PROTECTION		
Description	Erase / write protection settings (see FLS048)		
Type	Hardware specific		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	Only relevant if supported by hardware.		

Name	FLS_CALL_CYCLE		
Description	Cycle time of calls of the flash driver's main function.		
Type	Ticks (integer)		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	Only relevant if deadline monitoring for internal functionality has to be done in software (e.g. erase / write timings)		

Name	FLS_MAX_WRITE_NORMAL_MODE		
Description	The maximum number of bytes to write in one cycle of the flash driver's job processing function in normal mode.		
Type	integer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	**FLS176:** This value has to correspond to the settings in FLS_PAGE_LIST. The minimum number is defined by the size of one flash page and therefore depends on the underlying flash device.		

Name	FLS_MAX_WRITE_FAST_MODE		
Description	The maximum number of bytes to write in one cycle of the flash driver's job processing function in fast mode.		
Type	integer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		

Dependency	**FLS182:** This value has to correspond to the settings in FLS_PAGE_LIST. The minimum number is defined by the size of one flash page and therefore depends on the underlying flash device.

Name	FLS_MAX_READ_NORMAL_MODE		
Description	The maximum number of bytes to read or compare in one cycle of the flash driver's job processing function in normal mode.		
Type	integer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	The minimum number might depend on the underlying flash device or communication driver, e.g. if the access to an external flash device is done via SPI and the minimum transfer size on SPI is four bytes.		

Name	FLS_MAX_READ_FAST_MODE		
Description	The maximum number of bytes to read or compare in one cycle of the flash driver's job processing function in fast mode.		
Type	integer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	The minimum number might depend on the underlying flash device or communication driver, e.g. if the access to an external flash device is done via SPI and the minimum transfer size on SPI is four bytes.		

Name	FLS_AC_ERASE		
Description	Address offset in RAM to which the erase flash access code shall be loaded. Used as function pointer to access the erase flash access code.		
Type	Hardware specific		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	None		

Name	FLS_AC_WRITE		
Description	Address offset in RAM to which the write flash access code shall be loaded. Used as function pointer to access the write flash access code.		
Type	Hardware specific		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	None		

Name	FLS_JOB_END_NOTIFICATION

Description	Mapped to the job end notification routine provided by the upper layer module, typically the Flash EEPROM Emulation module.		
Type	Function pointer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	None		

Name	FLS_JOB_ERROR_NOTIFICATION		
Description	Mapped to the job error notification routine provided by the upper layer module, typically the Flash EEPROM Emulation module.		
Type	Function pointer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	None		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
Fls_SectorList	1	Module / None

10.2.4 Fls_SectorList

SWS Item	**FLS201:**
Container Name	Fls_SectorList
Description	List of flashable sectors and pages.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
Fls_Sector	1..n	Module / None

10.2.5 Fls_Sector

SWS Item	**FLS202:**
Container Name	Fls_Sector
Description	Configuration description of a flashable sector
Configuration Parameters	

Name	FLS_SECTOR_STARTADDRESS
Description	Start address of this sector
Type	Fls_AddressType (see chapter 8.2.2)

Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	None		

Name	FLS_SECTOR_SIZE		
Description	Size of this sector		
Type	Fls_LengthType (see chapter 8.2.3)		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	The sector size has to be an integer multiple of the page size.		

Name	FLS_PAGE_SIZE		
Description	Size of one page of this sector		
Type	Fls_LengthType (see chapter 8.2.3)		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	The sector size has to be an integer multiple of the page size.		

Name	FLS_NUMBER_OF_SECTORS		
Description	Number of continuous sectors with the above characteristics.		
Type	integer		
Unit	--		
Range	--	Implementation specific	
Configuration Class	***Pre-compile***	X	Variant PC
	Link time	--	--
	Post Build	X	Variant PB
Scope	Module		
Dependency	None		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
--	--	--

10.3 Published Information

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

FLS177: The following table specifies the information that shall be published in the module's description file. Further hardware or implementation specific information can be added if necessary.

SWS Item	**FLS178:**	
Information elements		
Information element name	***Type / Range***	***Information element description***
FLS_VENDOR_ID	#define / uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
FLS_MODULE_ID	#define / uint8	Module ID of this module from Module List
FLS_AR_MAJOR_VERSION	#define / uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
FLS_AR_MINOR_VERSION	#define / uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
FLS_AR_PATCH_VERSION	#define / uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
FLS_SW_MAJOR_VERSION	#define / uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
FLS_SW_MINOR_VERSION	#define / uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
FLS_SW_PATCH_VERSION	#define / uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
FLS_ERASED_VALUE	Integer / --	The contents of an erased flash memory cell.
FLS_ERASE_TIME	µsec / --	Maximum time to erase one complete flash sector.
FLS_WRITE_TIME	µsec / --	Maximum time to program one complete flash page.
FLS_AC_SIZE_ERASE	Integer / --	Number of bytes in RAM needed for the erase flash access code.
FLS_AC_SIZE_WRITE	Integer / --	Number of bytes in RAM needed for the write flash access code.
FLS_AC_LOCATION_ERASE	Integer / --	Position in RAM, to which the erase flash access code has to be loaded. Only relevant if the erase flash access code is not position independent. If this information is not provided it is assumed that the erase flash access code is position independent and that therefore the RAM position can be freely configured.
FLS_AC_LOCATION_WRITE	Integer / --	Position in RAM, to which the write flash access code has to be loaded. Only relevant if the write flash access code is not position independent. If this information is not provided it is assumed that the write flash access code is position independent and that therefore the RAM position can be freely configured.
FLS_EXPECTED_HW_ID	-- / --	Unique identifier of the hardware device that is expected by this driver (the device for which this driver has been implemented). Only relevant for external flash drivers.

FLS_SPECIFIED_ERASE_CYCLES	Integer / --	Number of erase cycles specified for the flash device (usually given in the device data sheet). **FLS198:** If the number of specified erase cycles depends on the operating environment (temperature, voitage, …) during reprogramming of the flash device, the minimum number for which a data retention of at least 15 years over the temperature range from -40°C .. +125°C can be guaranteed shall be given. Note: If there are different numbers of specified erase cycles for different flash sectors of the device this parameter has to be extended to a parameter list (similar to the sector list above).

11 Changes to Release 1

11.1 Deleted SWS Items

SWS Item	*Rationale*
FLS074	New SWS template (didn't fit with new document structure)
FLS045	New SWS template (didn't fit with new document structure)
FLS115	New SWS template (didn't fit with new document structure)

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
FLS112	FLS167	New SWS template (copy-paste didn't work on the tags)
FLS113	FLS168	New SWS template (copy-paste didn't work on the tags)
FLS043	FLS169	New SWS template (copy-paste didn't work on the tags)
FLS044	FLS170	New SWS template (copy-paste didn't work on the tags)
FLS085	FLS171	New SWS template (copy-paste didn't work on the tags)
FLS103	FLS173	New SWS template (copy-paste didn't work on the tags)
FLS050	FLS175	New SWS template (copy-paste didn't work on the tags)
FLS051	FLS176	New SWS template (copy-paste didn't work on the tags)
FLS119	FLS177	New SWS template (copy-paste didn't work on the tags)
FLS106	FLS179	New SWS template (copy-paste didn't work on the tags)

11.3 Changed SWS Items

SWS Item	*Rationale*
FLS077	New SWS template
FLS016, FLS017, FLS018, FLS024, FLS033, FLS036. FLS104, FLS105, FLS106, FLS022, FLS055, FLS056. FLS052, FLS095	New memory hardware abstraction architecture (reference to memory abstraction interface instead of flash interface)
FLS031, FlS036, FLS037, FLS040, FLS052	Added functionality (Read, Compare, SetMode functions)
FLS015	NULL pointer check removed (see FLS194)
FLS018, FLS024	Replaced `MEMIF_E_BUSY` with `MEMIF_BUSY`

11.4 Added SWS Items

SWS Item	*Rationale*
FLS145, FLS146	Bugzilla entry #4873
FLS147	Bugzilla entry #4507
FLS148, FLS149, FLS150, FLS151, FLS152, FLS153, FLS154, FLS155, FLS156, FLS181, FLS182	RfC #6793: Same functionality as EEPROM driver. Compare and SetMode functions added to the flash driver.
FLS157, FLS158	Bugzilla entry #4621
FLS159, FLS160, FLS161, FLS162, FLS163, FLS164, FLS167, FLS168, FLS172, FLS174, FLS178	New SWS template
FLS165, FLS166	New SWS template: GetVersionInfo function
FLS183, FLS184, FLS185, FLS186, FLS187, FLS188	RfC #6798: Scalability of flash driver
FLS190	BSW00432
FLS191	BSW12265
FLS193	To resolve review issue for BSW00325
FLS194	Added because of BSW00414
FLS195	RfC12405: Clarification of RAM loading and blocking
FLS196	RfC11088: Clarification on number of sectors to erase per main function cycle
FLS197	RfC11609: Definition of missing Fls_AddressType and Fls_LengthType.
FLS198	RfC11578: Added missing published parameter FLS_SPECIFIED_ERASE_CYCLES.
FLS200	RfC11758: Added specific result for compare job if blocks differ
FLS201, FLS202	RfC 13177: Configuration description of flash sector list added.