Document Title	Specification of Flash Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	025
Document Classification	Standard

Document Version	3.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
02.11.2011	3.2.0	AUTOSAR Administration	• References to HW specific errors corrected • Range of configuration parameters adapted • Consistency checking reformulated • Module short name changed
19.10.2010	3.1.0	AUTOSAR Administration	• Configuration parameter FlsDefaultMode added • Container with SPI reference added • Check for NULL pointer added
30.11.2009	3.0.0	AUTOSAR Administration	• References to AUTOSAR Standard Errors added • Range of configuration parameters restricted • Multiplicity of notification routines corrected • Several typing and formatting errors corrected • Legal disclaimer revised
23.06.2008	2.2.2	AUTOSAR Administration	Legal disclaimer revised
23.01.2008	2.2.1	AUTOSAR Administration	Table formatting corrected

Document Change History

Date	Version	Changed by	Change Description
11.12.2007	2.2.0	AUTOSAR Administration	• NULL pointer check added to Fls_Compare • NULL pointer check detailed (in general) • Restriction removed to allow re-initialization of module • Tables in chapters 8 and 10 generated from UML model • Document meta information extended • Small layout adaptations made
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

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 internal and external drivers. Hence the API is semantically identical.⌋ (BSW12147, BSW12148)

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.
Fls, FLS	Official AUTOSAR abbreviation for the module flash driver (different writing depending on the context, same meaning).
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
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] General Requirements on SPAL,
AUTOSAR_SRS_SPALGeneral.pdf

[5] Requirements on Flash Driver
AUTOSAR_SRS_FlashDriver.pdf

[6] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_SRS_MemoryHWAbstractionLayer.pdf

[7] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[8] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

3.2 Related standards and norms

[9] 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. ⌋ (BSW00380)

[FLS179] ⌈Pre- and post-compile configuration parameters shall be located outside the source code of the module to allow for automatic (tool based) configuration. ⌋ (BSW159, BSW00380, BSW00419)

5.1.2 Header file structure

[FLS107] ⌈The Fls module shall comply with the following file structure:

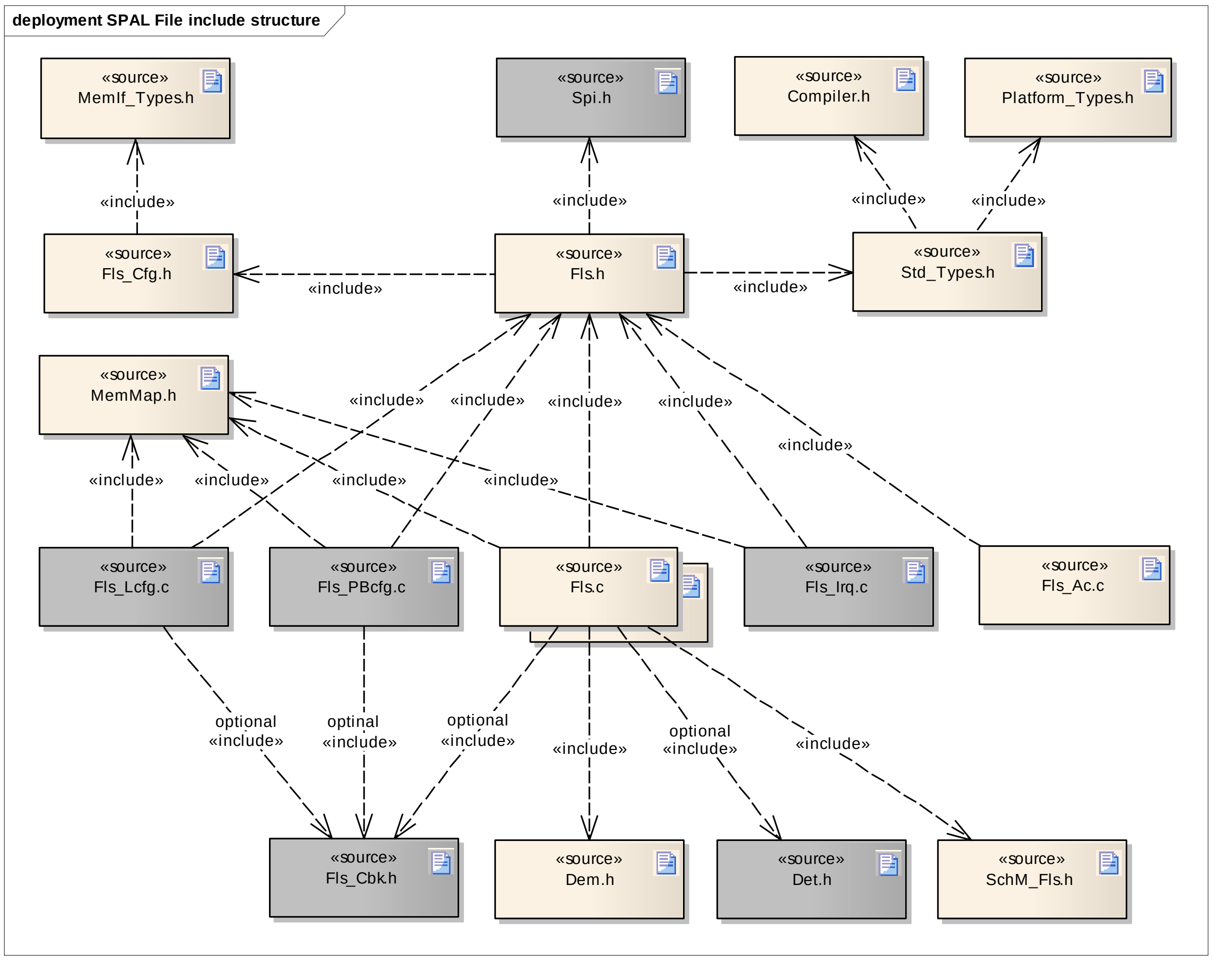

Figure 1: File include structure

Note: The files shown in grey are optional and might not be present for all implementations and/or configurations of a specific implementation of the Fls module.
⌋ (BSW00381, BSW00412, BSW00409, BSW00346, BSW158, BSW00301)

[FLS308] ⌈Types and definitions common to several flash driver instances shall be given in the header file `MemIf_Types.h`. ⌋ ()

[FLS309] ⌈Types and definitions specific for one flash driver shall be given in the header file `Fls.h`. ⌋ ()

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

Requirement	Satisfied by
-	FLS337
-	FLS033
-	FLS256
-	FLS248
-	FLS341
-	FLS273
-	FLS340
-	FLS343
-	FLS336
-	FLS303
-	FLS215
-	FLS196
-	FLS235
-	FLS262
-	FLS157
-	FLS348
-	FLS035
-	FLS329
-	FLS249
-	FLS110
-	FLS200
-	FLS240
-	FLS322
-	FLS257
-	FLS326
-	FLS161
-	FLS165
-	FLS217
-	FLS146
-	FLS253
-	FLS117
-	FLS302
-	FLS272
-	FLS364
-	FLS333
-	FLS258
-	FLS066
-	FLS344

-	FLS255
-	FLS345
-	FLS209
-	FLS269
-	FLS360
-	FLS346
-	FLS308
-	FLS356
-	FLS214
-	FLS261
-	FLS335
-	FLS328
-	FLS361
-	FLS334
-	FLS358
-	FLS330
-	FLS211
-	FLS309
-	FLS359
-	FLS332
-	FLS304
-	FLS260
-	FLS137
-	FLS001
-	FLS327
-	FLS251
-	FLS145
-	FLS362
-	FLS347
-	FLS247
-	FLS363
-	FLS109
-	FLS158
-	FLS349
-	FLS065
-	FLS325
-	FLS216
-	FLS208
-	FLS166
-	FLS167
-	FLS036
-	FLS147

-	FLS339
-	FLS331
-	FLS320
-	FLS357
-	FLS263
-	FLS099
-	FLS252
-	FLS250
-	FLS259
-	FLS338
-	FLS342
-	FLS254
-	FLS321
BSW00300	FLS366
BSW00301	FLS107
BSW00302	FLS366
BSW00304	FLS366
BSW00306	FLS366
BSW00307	FLS366
BSW00308	FLS366
BSW00309	FLS366
BSW00312	FLS366
BSW00314	FLS366
BSW00321	FLS366
BSW00323	FLS026, FLS020, FLS021, FLS027, FLS015, FLS098, FLS097
BSW00324	FLS366
BSW00325	FLS193
BSW00326	FLS366
BSW00327	FLS007, FLS317, FLS318, FLS319, FLS313, FLS314, FLS315, FLS316, FLS310, FLS311, FLS312
BSW00328	FLS366
BSW00330	FLS366
BSW00331	FLS267, FLS317, FLS318, FLS319, FLS313, FLS314, FLS315, FLS316, FLS310, FLS311, FLS312
BSW00333	FLS366
BSW00334	FLS366
BSW00336	FLS366
BSW00337	FLS007, FLS317, FLS318, FLS319, FLS313, FLS314, FLS315, FLS316, FLS310, FLS311, FLS312
BSW00338	FLS077
BSW00339	FLS366
BSW00341	FLS366
BSW00342	FLS366

BSW00344	FLS366
BSW00346	FLS107
BSW00347	FLS366
BSW00348	FLS366
BSW00350	FLS162, FLS077
BSW00353	FLS366
BSW00355	FLS366
BSW00359	FLS366
BSW00360	FLS366
BSW00361	FLS366
BSW00369	FLS267
BSW00370	FLS366
BSW00371	FLS366
BSW00375	FLS366
BSW00378	FLS366
BSW00380	FLS179, FLS159
BSW00381	FLS107
BSW00385	FLS007, FLS004, FLS317, FLS318, FLS319, FLS313, FLS314, FLS315, FLS316, FLS310, FLS311, FLS312
BSW00386	FLS163, FLS162, FLS077
BSW00387	FLS366
BSW00398	FLS366
BSW004	FLS205, FLS206
BSW00401	FLS366
BSW00404	FLS014
BSW00405	FLS014
BSW00406	FLS268
BSW00409	FLS160, FLS107
BSW00412	FLS107
BSW00415	FLS366
BSW00416	FLS366
BSW00417	FLS366
BSW00419	FLS179
BSW00420	FLS366
BSW00421	FLS154, FLS006, FLS106, FLS104, FLS105
BSW00422	FLS366
BSW00423	FLS366
BSW00424	FLS366
BSW00426	FLS366
BSW00427	FLS366
BSW00428	FLS366
BSW00429	FLS366

BSW00431	FLS366
BSW00433	FLS366
BSW00434	FLS366
BSW005	FLS366
BSW006	FLS366
BSW007	FLS366
BSW009	FLS366
BSW010	FLS366
BSW101	FLS014
BSW12057	FLS014
BSW12063	FLS366
BSW12064	FLS366
BSW12067	FLS366
BSW12069	FLS366
BSW12075	FLS003, FLS002
BSW12078	FLS366
BSW12083	FLS366
BSW12107	FLS144
BSW12125	FLS086
BSW12129	FLS233, FLS232, FLS234
BSW12132	FLS048
BSW12134	FLS236, FLS239, FLS238, FLS098, FLS097
BSW12135	FLS026, FLS027, FLS226, FLS225, FLS223
BSW12136	FLS020, FLS021, FLS221, FLS220, FLS218
BSW12137	FLS230, FLS183, FLS229
BSW12138	FLS034, FLS184
BSW12141	FLS056
BSW12143	FLS023, FLS030, FLS268, FLS324, FLS323, FLS100
BSW12144	FLS037, FLS039, FLS038
BSW12145	FLS040
BSW12147	FLS088
BSW12148	FLS088
BSW12149	FLS366
BSW12158	FLS055
BSW12159	FLS026, FLS020, FLS021, FLS027, FLS098, FLS097
BSW12160	FLS022
BSW12163	FLS366
BSW12169	FLS155
BSW12184	FLS040
BSW12193	FLS141, FLS140
BSW12194	FLS212, FLS213
BSW12265	FLS191

BSW12267	FLS366
BSW12448	FLS026, FLS020, FLS021, FLS027, FLS015, FLS098, FLS097
BSW12461	FLS086
BSW12462	FLS366
BSW12463	FLS366
BSW12468	FLS366
BSW13300	FLS143
BSW13301	FLS150, FLS241, FLS153, FLS152, FLS244, FLS151, FLS243, FLS186
BSW13302	FLS156, FLS155, FLS187
BSW13303	FLS040
BSW13304	FLS040
BSW157	FLS164, FLS006
BSW158	FLS107
BSW159	FLS179
BSW160	FLS366
BSW161	FLS366
BSW162	FLS366
BSW164	FLS193
BSW167	FLS205, FLS206
BSW168	FLS366
BSW170	FLS366
BSW171	FLS183, FLS184, FLS185, FLS186, FLS187
BSW172	FLS366

7 Functional specification

7.1 General design rules

[FLS001] ⌈The FLS module shall offer asynchronous services for operations on flash memory (read/erase/write). ⌋ ( )

[FLS002] ⌈The FLS module shall not buffer data. The FLS module shall use application data buffers that are referenced by a pointer passed via the API. ⌋ (BSW12075)

[FLS003] ⌈The FLS module shall not ensure data consistency of the given application buffer. ⌋ (BSW12075)

It is the responsibility of the FLS module's environment to ensure consistency of flash data during a flash read or write operation.

[FLS205] ⌈The FLS module shall check static configuration parameters statically (at the latest during compile time) for correctness. ⌋ (BSW167, BSW004)

[FLS206] ⌈The FLS module shall validate the version information in the FLS module header and source files for consistency (e.g. by comparing the version information in the module header and source files with a pre-processor macro). ⌋ (BSW167, BSW004)

[FLS208] ⌈The FLS module shall combine all available flash memory areas into one linear address space (denoted by the parameters `FlsBaseAddress` and `FlsTotalSize`). ⌋ ( )

[FLS209] ⌈The FLS module shall map the address and length parameters for the read, write, erase and compare functions as "virtual" addresses 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`. ⌋ (BSW00409)

[FLS161] ⌈Development error values are of type uint8. ⌋ ( )

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

[FLS004] ⌈

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	0x06
Erase verification (blank check) failed	Development	FLS_E_VERIFY_ERASE_FAILED	0x07
Write verification (compare) failed	Development	FLS_E_VERIFY_WRITE_FAILED	0x08
Timeout exceeded	Development	FLS_E_TIMEOUT	0x09
API service called with NULL pointer	Development	FLS_E_PARAM_POINTER	0x0a
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

⌋ (BSW00385)

[FLS310] ⌈The following development error codes shall be reported when an API service is called with a wrong parameter: `FLS_E_PARAM_CONFIG`, `FLS_E_PARAM_ADDRESS`, `FLS_E_PARAM_LENGTH`, `FLS_E_PARAM_DATA`. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS311] ⌈The development error code `FLS_E_UNINIT` shall be reported when an API service is called prior to module initialization. Exceptions are the functions `Fls_Init` and `Fls_GetVersionInfo`. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS312] ⌈The development error code `FLS_E_BUSY` shall be reported when an API service is called while the module is still busy. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS313] ⌈The development error code `FLS_E_VERIFY_ERASE_FAILED` shall be reported when the erase verification (blankcheck) failed. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS314] ⌈The development error code `FLS_E_VERIFY_WRITE` shall be reported when the write verification (compare) failed. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS361] ⌈The development error code `FLS_E_TIMEOUT` shall be reported when the timeout supervision of a read, write, erase or compare job failed. ⌋ ( )

[FLS315] ⌈The production error code `FLS_E_ERASE_FAILED` shall be reported when the flash erase function failed. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS316] ⌈The production error code `FLS_E_WRITE_FAILED` shall be reported when the flash write function failed. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS317] ⌈The production error code `FLS_E_READ_FAILED` shall be reported when the flash read function failed. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS318] ⌈The production error code `FLS_E_COMPARE_FAILED` shall be reported when the flash compare function failed. ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

[FLS319] ⌈The production error code `FLS_E_UNEXPECTED_FLASH_ID` shall be reported when the expected flash ID is not matched (see FLS144). ⌋ (BSW00337, BSW00385, BSW00327, BSW00331)

Note: FLS313, FLS314 and FLS361 describe development errors although from their behaviour those errors may also occur in a production system. Since erase verification (blankcheck, FLS022, FLS055), write verification (FLS056) and timeout supervision (FLS272, FLS359, FLS360) will have a significant impact on the systems performance and since data consistency in a production system will most likely be ensured by other means (like e.g. checksums, signatures, diagnostic timeouts) it was a design decision from the working group to make these features available only during the development phase (i.e. when development error detection is enabled). This way anyone who wants to use these features (and pay the price) can do so also in a production system by leaving development error detection enabled whilst anyone who doesn't want to have the overhead can simply switch those features off.

7.3 Error detection

[FLS077] ⌈The detection of development errors shall be configurable (on/off) at pre-compile time. The switch `FlsDevErrorDetect` (see chapter 10) shall activate or deactivate the detection of all development errors. ⌋ (BSW00338, BSW00386, BSW00350)

[FLS162] ⌈If the `FlsDevErrorDetect` 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. ⌋ (BSW00386, BSW00350)

[FLS163] ⌈The detection of production code errors cannot be switched off. ⌋ (BSW00386)

7.4 Error notification

[FLS164] ⌈Detected development errors shall be reported to `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `FlsDevErrorDetect` is set (see chapter 10). ⌋ (BSW157)

[FLS006] ⌈Production relevant errors shall be reported to the Diagnostic Event Manager. ⌋ (BSW157, BSW00421)

[FLS267] ⌈The error codes shall not be used as return values of the called function. ⌋ (BSW00369, BSW00331)

[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. ⌋ (BSW00337, BSW00385, BSW00327)

7.5 External flash driver

[FLS144] ⌈During the initialization of the external flash driver, the FLS module shall check the hardware ID of the external flash device against the corresponding published parameter. If a hardware ID mismatch occurs, the FLS module shall report the error code `FLS_E_UNEXPECTED_FLASH_ID` to the Diagnostic Event Manager (DEM), set the FLS module status to `FLS_E_UNINIT` and shall not initialize itself. ⌋ (BSW12107)

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 FLS module's implementer shall place the code of the flash access routines into a separate C-module `Fls_ac.c`. ⌋ ( )

[FLS215] ⌈The FLS module's flash access routines shall only disable interrupts and wait for the completion of the erase / write command if necessary (that is if it has to be ensured that no other code is executed in the meantime). ⌋ ( )

[FLS211] ⌈The FLS module's implementer shall keep the execution time for the flash access code as short as possible. ⌋ ( )

[FLS140] ⌈The FLS module's 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 if the FLS module is configured to load the flash access code to RAM on job start. ⌋ (BSW12193)

[FLS141] ⌈The FLS module's 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 if the FLS module is configured to load the flash access code to RAM on job start. ⌋ (BSW12193)

[FLS212] ⌈The FLS module's main processing routine shall execute the flash access code routines. ⌋ (BSW12194)

[FLS213] ⌈The FLS module's main processing routine shall access the flash access code routines by means of the respective function pointer contained in the FLS module'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. ⌋ (BSW12194)

[FLS143] ⌈After an erase or write job has been finished or canceled, the FLS module's 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. ⌋ (BSW13300)

[FLS214] ⌈The FLS module shall only load the access code to the RAM if the access code cannot be executed out of flash ROM. ⌋ ( )

7.7 Support for Debugging

[FLS302] ⌈The module's status, mode and the job result shall be made available for debugging (reading). Therefore those variables shall be implemented as global variables. ⌋ ( )

[FLS303] ⌈The type definitions and declarations of all variables which shall be used for debugging shall be given in the modules header file `Fls.h`. ⌋ ( )

[FLS304] ⌈All variables which shall be used for debugging shall be described in detail in the modules description file. ⌋ ( )

7.8 Consistency checks

[FLS364] ⌈The Fls module shall perform inter module checks to avoid integration of incompatible files: all included header files shall be checked by pre-processing directives. The Fls module shall thereby verify that `<MODULENAME>_AR_RELEASE_MAJOR_VERSION` and `<MODULENAME>_AR_RELEASE_MINOR_VERSION` are identical to the expected values, where `<MODULENAME>` is the module abbreviation of the external module, which provides the included header file. If the values are not identical, an error shall be raised at compile time. ⌋ ( )

Note: The configuration tool shall check all configuration parameters for being within the expected bounds. Also the dependencies between configuration parameters shall be checked by the configuration tool during system generation or during the build process (for details see chapter 10).

8 API specification

8.1 Imported types

[FLS248] ⌈

Module	***Imported Type***
Dem	Dem_EventIdType
	Dem_EventStatusType
MemIf	MemIf_JobResultType
	MemIf_ModeType
	MemIf_StatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ ( )

[FLS320] ⌈The following type definitions shall be imported from Std_Types.h:

- Std_ReturnType
- Std_VersionInfoType⌋ ( )

[FLS321] ⌈The following type definitions shall be imported from Dem_Types.h:

- Dem_EventIdType⌋ ( )

[FLS322] ⌈The following type definitions shall be imported from MemIf_Types.h:

- MemIf_ModeType
- MemIf_StatusType
- MemIf_JobResultType⌋ ( )

8.2 Type definitions

8.2.1 Fls_ConfigType

Name:	`Fls_ConfigType`	
Type:	`Structure`	
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

Name:	`Fls_AddressType`

Type:	`uint`		
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.		

[FLS216] ⌈The type Fls_AddressType shall have 0 as lower limit for each flash device. ⌋ ( )

[FLS217] ⌈The FLS module shall add a device specific base address to the address type Fls_AddressType if necessary. ⌋ ( )

8.2.3 Fls_LengthType

Name:	`Fls_LengthType`		
Type:	`uint`		
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

[FLS249] ⌈

Service name:	Fls_Init	

Syntax:	`void Fls_Init(`
	`    const Fls_ConfigType* ConfigPtr`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to flash driver configuration set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the Flash Driver.	

⌋ ( )

[FLS014] ⌈The function `Fls_Init` shall initialize the FLS module (software) and all flash memory relevant registers (hardware) with parameters provided in the given configuration set. ⌋ (BSW00404, BSW00405, BSW101, BSW12057)

[FLS191] ⌈The function `Fls_Init` shall store the pointer to the given configuration set in a variable in order to allow the FLS module access to the configuration set contents during runtime. ⌋ (BSW12265)

[FLS086] ⌈The function `Fls_Init` shall initialize all FLS 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. ⌋ (BSW12125, BSW12461)

[FLS015] ⌈If development error detection for the module Fls is enabled: the function `Fls_Init` shall check the (hardware specific) contents of the given configuration set for being within the allowed range. If this is not the case, it shall raise the development error `FLS_E_PARAM_CONFIG`. ⌋ (BSW00323, BSW12448)

[FLS323] ⌈The function `Fls_Init` shall set the FLS module state to `MEMIF_IDLE` after having finished the FLS module initialization. ⌋ (BSW12143)

[FLS324] ⌈The function `Fls_Init` shall set the flash job result to `MEMIF_JOB_OK` after having finished the FLS module initialization. ⌋ (BSW12143)

[FLS268] ⌈If development error detection for the module Fls is enabled: the function `Fls_Init` shall check that the FLS module is currently not busy (FLS module state is not `MEMIF_BUSY`). If this check fails, the function `Fls_Init` shall raise the development error `FLS_E_BUSY`. ⌋ (BSW12143, BSW00406)

[FLS048] ⌈If supported by hardware, the function `Fls_Init` shall set the flash memory erase/write protection as provided in the configuration set. ⌋ (BSW12132)

[FLS325] ⌈If variant is VARIANT-PRE-COMPILE, a NULL pointer shall be passed to the initialization routine. ⌋ ( )

[FLS326] ⌈If variant is VARIANT-PRE-COMPILE, the check for the NULL pointer shall be omitted. ⌋ ( )

8.3.2 Fls_Erase

[FLS250] ⌈

Service name:	Fls_Erase

Syntax:	`Std_ReturnType Fls_Erase(`
	`    Fls_AddressType TargetAddress,`

	`Fls_LengthType Length`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
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 (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: erase command has been accepted E_NOT_OK: erase command has not been accepted
Description:	Erases flash sector(s).	

⌋ ()

[FLS218] ⌈The job of the function `Fls_Erase` shall erase one or more complete flash sectors. ⌋ (BSW12136)

[FLS327] ⌈The function `Fls_Erase` shall copy the given parameters to FLS module internal variables and initiate an erase job. ⌋ ()

[FLS328] ⌈After initiating the erase job, the function `Fls_Erase` shall set the FLS module status to `MEMIF_BUSY`. ⌋ ()

[FLS329] ⌈After initiating the erase job, the function `Fls_Erase` shall set the job result to `MEMIF_JOB_PENDING`. ⌋ ()

[FLS330] ⌈After initiating the erase job, the function `Fls_Erase` shall return with `E_OK`. ⌋ ()

[FLS220] ⌈The FLS module shall execute the job of the function `Fls_Erase` asynchronously within the FLS module's main function. ⌋ (BSW12136)

[FLS221] ⌈The job of the function `Fls_Erase` shall erase a flash memory block starting from `FlsBaseAddress` + `TargetAddress` of size `Length.`

Note: `Length` will be rounded up to the next full sector boundary since only complete flash sectors can be erased. ⌋ (BSW12136)

[FLS020] ⌈If development error detection for the module Fls is enabled: the function `Fls_Erase` shall check that the erase start address (`FlsBaseAddress + TargetAddress`) is aligned to a flash sector boundary and that it lies within the specified lower and upper flash address boundaries. If this check fails, the function `Fls_Erase` shall reject the erase request, raise the development error

FLS_E_PARAM_ADDRESS and return with E_NOT_OK. ⌋ (BSW00323, BSW12448, BSW12136, BSW12159)

[FLS021] ⌈If development error detection for the module Fls is enabled: the function Fls_Erase 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 this check fails, the function Fls_Erase shall reject the erase request, raise the development error FLS_E_PARAM_LENGTH and return with E_NOT_OK. ⌋ (BSW00323, BSW12448, BSW12136, BSW12159)

[FLS065] ⌈If development error detection for the module Fls is enabled: the function Fls_Erase shall check that the FLS module has been initialized. If this check fails, the function Fls_Erase shall reject the erase request, raise the development error FLS_E_UNINIT and return with E_NOT_OK. ⌋ ()

[FLS023] ⌈If development error detection for the module Fls is enabled: the function Fls_Erase shall check that the FLS module is currently not busy. If this check fails, the function Fls_Erase shall reject the erase request, raise the development error FLS_E_BUSY and return with E_NOT_OK. ⌋ (BSW12143)

[FLS145] ⌈If possible, e.g. with interrupt controlled implementations, the FLS module shall start the first round of the erase job directly within the function Fls_Erase to reduce overall runtime. ⌋ ()

8.3.3 Fls_Write

[FLS251] ⌈

Service name:	Fls_Write	

Syntax:	`Std_ReturnType Fls_Write(`
	`    Fls_AddressType TargetAddress,`
	`    const uint8* SourceAddressPtr,`
	`    Fls_LengthType Length`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
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
	SourceAddressPtr	Pointer to source data buffer
	Length	Number of bytes to write Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters (inout):	None	
Parameters (out):	None	

Return value:	Std_ReturnType	E_OK: write command has been accepted E_NOT_OK: write command has not been accepted
Description:	Writes one or more complete flash pages.	

⌋ ( )

[FLS223] ⌈The job of the function `Fls_Write` shall write one or more complete flash pages to the flash device. ⌋ (BSW12135)

[FLS331] ⌈The function `Fls_Write` shall copy the given parameters to Fls module internal variables and initiate a write job. ⌋ ( )

[FLS332] ⌈After initiating the write job, the function `Fls_Write` shall set the FLS module status to `MEMIF_BUSY`. ⌋ ( )

[FLS333] ⌈After initiating the write job, the function `Fls_Write` shall set the job result to `MEMIF_JOB_PENDING`. ⌋ ( )

[FLS334] ⌈After initiating the write job, the function `Fls_Write` shall return with `E_OK`. ⌋ ( )

[FLS225] ⌈The FLS module shall execute the write job of the function `Fls_Write` asynchronously within the FLS module's main function. ⌋ (BSW12135)

[FLS226] ⌈The job of the function `Fls_Write` shall program a flash memory block with data provided via `SourceAddressPtr` starting from `FlsBaseAddress + TargetAddress` of size `Length.` ⌋ (BSW12135)

[FLS026] ⌈If development error detection for the module Fls is enabled: the function `Fls_Write` shall check that the write start address (`FlsBaseAddress + TargetAddress`) is aligned to a flash page boundary and that it lies within the specified lower and upper flash address boundaries. If this check fails, the function `Fls_Write` shall reject the write request, raise the development error `FLS_E_PARAM_ADDRESS` and return with `E_NOT_OK.` ⌋ (BSW12448, BSW00323, BSW12135, BSW12159)

[FLS027] ⌈If development error detection for the module Fls is enabled: the function `Fls_Write` shall check that the write length is greater than 0, 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 this check fails, the function `Fls_Write` shall reject the write request, raise the development error `FLS_E_PARAM_LENGTH` and return with `E_NOT_OK.` ⌋ (BSW12448, BSW00323, BSW12135, BSW12159)

[FLS066] ⌈If development error detection for the module Fls is enabled: the function `Fls_Write` shall check that the FLS module has been initialized. If this check fails, the function `Fls_Write` shall reject the write request, raise the development error `FLS_E_UNINIT` and return with `E_NOT_OK.`⌋()

[FLS030] ⌈If development error detection for the module Fls is enabled: the function `Fls_Write` shall check that the FLS module is currently not busy. If this check fails, the function `Fls_Write` shall reject the write request, raise the development error `FLS_E_BUSY` and return with `E_NOT_OK.` ⌋ (BSW12143)

[FLS157] ⌈If development error detection for the module Fls is enabled: the function `Fls_Write` shall check the given data buffer pointer for not being a null pointer. If the data buffer pointer is a null pointer, the function `Fls_Write` shall reject the write request, raise the development error `FLS_E_PARAM_DATA` and return with `E_NOT_OK.`⌋()

[FLS146] ⌈If possible, e.g. with interrupt controlled implementations, the FLS module shall start the first round of the write job directly within the function `Fls_Write` to reduce overall runtime. ⌋()

8.3.4 Fls_Cancel

[FLS252] ⌈

Service name:	Fls_Cancel

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

Service ID[hex]:	0x03
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Cancels an ongoing job.

⌋()

[FLS229] ⌈The function `Fls_Cancel` shall cancel an ongoing flash read, write, erase or compare job. ⌋ (BSW12137)

[FLS230] ⌈The function `Fls_Cancel` shall abort a running job synchronously so that directly after returning from this function a new job can be started. ⌋ (BSW12137)

Note: The function `Fls_Cancel` is synchronous in its behaviour but at the same time asynchronous w.r.t. the underlying hardware: The job of the `Fls_Cancel` function (i.e. make the module ready for a new job request) is finished when it returns to the caller (hence it's synchronous) but on the other hand e.g. an erase job might still be ongoing in the hardware device (hence it's asynchronous w.r.t. the hardware).

[FLS335] ⌈The function `Fls_Cancel` shall reset the FLS module's internal job processing variables (like address, length and data pointer). ⌋()

[FLS336] ⌈The function `Fls_Cancel` shall set the FLS module state to `MEMIF_IDLE`. ⌋()

[FLS033] ⌈The function `Fls_Cancel` shall set the job result to `MEMIF_JOB_CANCELED` if the job result currently has the value `MEMIF_JOB_PENDING`. Otherwise the function `Fls_Cancel` shall leave the job result unchanged. ⌋()

[FLS147] ⌈If configured, the function `Fls_Cancel` shall call the error notification function to inform the caller about the cancellation of a job. ⌋()

Note: The content of the affected flash memory cells will be undefined when canceling an ongoing job with the function `Fls_Cancel`.

[FLS183] ⌈The function `Fls_Cancel` shall be pre-compile time configurable `On/Off` by the configuration parameter `FlsCancelApi.` ⌋(BSW171, BSW12137)

[FLS356] ⌈If development error detection for the module Fls is enabled: the function `Fls_Cancel` shall check that the FLS module has been initialized. If this check fails, the function `Fls_Cancel` shall raise the development error `FLS_E_UNINIT` and return`.` ⌋()

8.3.5 Fls_GetStatus

[FLS253] ⌈

Service name:	Fls_GetStatus	

Syntax:	`MemIf_StatusType Fls_GetStatus(`
	`    void`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	MemIf_StatusType	--
Description:	Returns the driver state.	

⌋()

[FLS034] ⌈The function `Fls_GetStatus` shall return the FLS module state synchronously. ⌋ (BSW12138)

[FLS184] ⌈The function `Fls_GetStatus` shall be pre-compile time configurable `On/Off` by the configuration parameter `FlsGetStatusApi.` ⌋ (BSW12138, BSW171)

[FLS357] ⌈If development error detection for the module Fls is enabled: the function `Fls_GetStatus` shall check that the FLS module has been initialized. If this check fails, the function `Fls_GetStatus` shall return with `MEMIF_UNINIT.` ⌋ ( )

8.3.6 Fls_GetJobResult

[FLS254] ⌈

Service name:	Fls_GetJobResult	

Syntax:	`MemIf_JobResultType Fls_GetJobResult(`
	`    void`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	MemIf_JobResultType	--
Description:	Returns the result of the last job.	

⌋ ( )

[FLS035] ⌈The function `Fls_GetJobResult` shall return the result of the last job synchronously⌋ ( )

[FLS036] ⌈The erase, write, read and compare functions shall share the same job result, i.e. only the result of the last job can be queried. The FLS module shall overwrite the job result with `MEMIF_JOB_PENDING` if the FLS module has accepted a new job. ⌋ ( )

[FLS185] ⌈The function `Fls_GetJobResult` shall be pre-compile time configurable `On/Off` by the configuration parameter `FlsGetJobResultApi.` ⌋ (BSW171)

[FLS358] ⌈If development error detection for the module Fls is enabled: the function `Fls_GetJobResult` shall check that the FLS module has been initialized. If this check fails, the function `Fls_GetJobResult` shall raise the development error `FLS_E_UNINIT` and return with `MEMIF_JOB_FAILED.` ⌋ ( )

8.3.7 Fls_Read

[FLS256] ⌈

Service name:	Fls_Read	
Syntax:	`Std_ReturnType Fls_Read( Fls_AddressType SourceAddress, uint8* TargetAddressPtr, Fls_LengthType Length )`	
Service ID[hex]:	0x07	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
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 (inout):	None	
Parameters (out):	TargetAddressPtr	Pointer to target data buffer
Return value:	Std_ReturnType	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted
Description:	Reads from flash memory.	

⌋ ( )

[FLS236] ⌈The function `Fls_Read` shall read from flash memory. ⌋ (BSW12134)

[FLS337] ⌈The function `Fls_Read` shall copy the given parameters to FLS module internal variables and initiate a read job. ⌋ ( )

[FLS338] ⌈After initiating a read job, the function `Fls_Read` shall set the FLS module status to `MEMIF_BUSY`. ⌋ ( )

[FLS339] ⌈After initiating a read job, the function `Fls_Read` shall set the FLS module job result to `MEMIF_JOB_PENDING`. ⌋ ( )

[FLS340] ⌈After initiating a read job, the function `Fls_Read` shall return with `E_OK`. ⌋ ( )

[FLS238] ⌈The FLS module shall execute the read job of the function `Fls_Read` asynchronously within the FLS module's main function. ⌋ (BSW12134)

[FLS239] ⌈The read job of the function `Fls_Read` shall copy a continuous flash memory block starting from `FlsBaseAddress + SourceAddress` of size `Length` to the buffer pointed to by `TargetAddressPtr`. ⌋ (BSW12134)

[FLS097] ⌈If development error detection for the module Fls is enabled: the function `Fls_Read` shall check that the read start address (`FlsBaseAddress + SourceAddress`) lies within the specified lower and upper flash address boundaries. If this check fails, the function `Fls_Read` shall reject the read job, raise development error `FLS_E_PARAM_ADDRESS` and return with `E_NOT_OK`. ⌋ (BSW00323, BSW12448, BSW12134, BSW12159)

[FLS098] ⌈If development error detection for the module Fls is enabled: the function `Fls_Read` 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 this check fails, the function `Fls_Read` shall reject the read job, raise the development error `FLS_E_PARAM_LENGTH` and return with `E_NOT_OK`. ⌋ (BSW00323, BSW12448, BSW12134, BSW12159)

[FLS099] ⌈If development error detection for the module Fls is enabled: the function `Fls_Read` shall check that the driver has been initialized. If this check fails, the function `Fls_Read` shall reject the read request, raise the development error `FLS_E_UNINIT` and return with `E_NOT_OK`. ⌋ ()

[FLS100] ⌈If development error detection for the module Fls is enabled: the function `Fls_Read` shall check that the driver is currently not busy. If this check fails, the function `Fls_Read` shall reject the read request, raise the development error `FLS_E_BUSY` and return with `E_NOT_OK`. ⌋ (BSW12143)

[FLS158] ⌈If development error detection for the module Fls is enabled: the function `Fls_Read` shall check the given data buffer pointer for not being a null pointer. If the data buffer pointer is a null pointer, the function `Fls_Read` shall reject the read request, raise the development error `FLS_E_PARAM_DATA` and return with `E_NOT_OK`. ⌋ ()

[FLS240] ⌈The FLS module's environment shall only call the function `Fls_Read` after the FLS module has been initialized. ⌋ ()

8.3.8 Fls_Compare

[FLS257] ⌈

Service name:	Fls_Compare

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

Service ID[hex]:	0x08
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant

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 (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: compare command has been accepted E_NOT_OK: compare command has not been accepted
Description:	Compares the contents of an area of flash memory with that of an application data buffer.	

⌋ ( )

[FLS241] ⌈The function `Fls_Compare` shall compare the contents of an area of flash memory with that of an application data buffer. ⌋ (BSW13301)

[FLS341] ⌈The function `Fls_Compare` shall copy the given parameters to Fls module internal variables and initiate a compare job. ⌋ ( )

[FLS342] ⌈After initiating the compare job, the function Fls_Compare shall set the status to `MEMIF_BUSY`. ⌋ ( )

[FLS343] ⌈After initiating the compare job, the function Fls_Compare shall set the job result to `MEMIF_JOB_PENDING`. ⌋ ( )

[FLS344] ⌈After initiating the compare job, the function Fls_Compare shall return with E_OK. ⌋ ( )

[FLS243] ⌈The FLS module shall execute the job of the function `Fls_Compare` asynchronously within the FLS module's main function. ⌋ (BSW13301)

[FLS244] ⌈The job of the function `Fls_Compare` shall compare a continuous flash memory block starting from `FlsBaseAddress + SourceAddress` of size `Length` with the buffer pointed to by `TargetAddressPtr`. ⌋ (BSW13301)

[FLS150] ⌈If development error detection for the module Fls is enabled: the function `Fls_Compare` shall check that the compare start address (`FlsBaseAddress + SourceAddress`) lies within the specified lower and upper flash address boundaries. If this check fails, the function `Fls_Compare` shall reject the compare job, raise the development error FLS_E_PARAM_ADDRESS and return with `E_NOT_OK.` ⌋ (BSW13301)

[FLS151] ⌈If If development error detection for the module Fls is enabled: the function `Fls_Compare` 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 this check fails, the function `Fls_Compare` shall reject the compare job, raise the development error `FLS_E_PARAM_LENGTH` and return with `E_NOT_OK.` ⌋ (BSW13301)

[FLS152] ⌈If development error detection for the module Fls is enabled: the function `Fls_Compare` shall check that the driver has been initialized. If this check fails, the function `Fls_Compare` shall reject the compare job, raise the development error `FLS_E_UNINIT` and return with `E_NOT_OK.` ⌋ (BSW13301)

[FLS153] ⌈If development error detection for the module Fls is enabled: the function `Fls_Compare` shall check that the driver is currently not busy. If this check fails, the function `Fls_Compare` shall reject the compare job, raise the development error `FLS_E_BUSY` and return with `E_NOT_OK.` ⌋ (BSW13301)

[FLS273] ⌈If development error detection for the module Fls is enabled: the function `Fls_Compare` shall check the given data buffer pointer for not being a null pointer. If the data buffer pointer is a null pointer, the function `Fls_Compare` shall reject the request, raise the development error `FLS_E_PARAM_DATA` and return with `E_NOT_OK.` ⌋ ()

[FLS186] ⌈The function `Fls_Compare` shall be pre-compile time configurable `On/Off` by the configuration parameter `FlsCompareApi.` ⌋ (BSW171, BSW13301)

8.3.9 Fls_SetMode

[FLS258] ⌈

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 (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Sets the flash driver's operation mode.	

⌋ ()

[FLS155] ⌈The function `Fls_SetMode` shall set the FLS module's operation mode to the given "Mode" parameter. ⌋ (BSW12169, BSW13302)

[FLS156] ⌈If development error detection for the module Fls is enabled: the function `Fls_SetMode` shall check that the FLS module is currently not busy. If this check fails, the function `Fls_SetMode` shall reject the set mode request and raise the development error code `FLS_E_BUSY`. ⌋ (BSW13302)

[FLS187] ⌈The function `Fls_SetMode` shall be pre-compile time configurable `On/Off` by the configuration parameter `FlsSetModeApi`. ⌋ (BSW171, BSW13302)

8.3.10 Fls_GetVersionInfo

[FLS259] ⌈

Service name:	Fls_GetVersionInfo	
Syntax:	`void Fls_GetVersionInfo( Std_VersionInfoType* VersioninfoPtr )`	
Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	VersioninfoPtr	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information of this module.	

⌋ ()

[FLS165] ⌈The function `Fls_GetVersionInfo` shall return the version information of the FLS module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋ ()

[FLS166] ⌈The function `Fls_GetVersionInfo` shall be pre-compile time configurable `On/Off` by the configuration parameter `FlsVersionInfoApi`. ⌋ ()

[FLS247] ⌈If source code for caller and callee of the function `Fls_GetVersionInfo` is available, the FLS module should realize this function as a macro. The FLS module should define this macro in the module's header file. ⌋ ()

[FLS363] ⌈If development error detection for the module `Fls` is enabled: the function `Fls_GetVersionInfo` shall raise the development error `FLS_E_PARAM_POINTER` if the argument is a NULL pointer and return without any action. ⌋ ()

8.4 Call-back notifications

This chaper lists all functions provided by the Fls module to lower layer modules.

Note: There are no callback functions to lower layer modules provided by the Flash Driver since this module is at the lowest (software) layer.

[FLS193] ⌈Depending on implementation, callback routines provided and/or invoked by the FLS module may be called on interrupt level. The module providing those routines has therefore to make sure that their runtime is reasonably short, i.e. since callbacks may be propagated upward through several software layers. ⌋ (BSW164, BSW00325)

8.5 Scheduled functions

This chapter lists all functions provided by the Fls module and called directly by the Basic Software Module Scheduler.

[FLS269] ⌈The Fls module shall provide only one scheduled function. Reading from / writing to flash memory cannot usually be done simultaneously and the overhead for synchronizing two scheduled functions would outweigh the benefits. ⌋ ( )

8.5.1 Fls_MainFunction

[FLS255] ⌈

Service name:	Fls_MainFunction

Syntax:	`void Fls_MainFunction(`
	`    void`
	`)`

Service ID[hex]:	0x06
Timing:	FIXED_CYCLIC
Description:	Performs the processing of jobs.

⌋ ( )

[FLS037] ⌈The function `Fls_MainFunction` shall perform the processing of the flash read, write, erase and compare jobs. ⌋ (BSW12144)

[FLS038] ⌈When a job has been initiated, the FLS module's environment shall call the function `Fls_MainFunction` cyclically until the job is finished. ⌋ (BSW12144)

Note: The function Fls_MainFunction may also be called cyclically if no job is currently pending.

[FLS039] ⌈The function `Fls_MainFunction` shall return without any action if no job is pending. ⌋ (BSW12144)

[FLS040] ⌈The function `Fls_MainFunction` shall only process as much data in one call cycle as statically configured for the current job type (read, write or compare) and the current FLS module's operating mode (normal, fast). ⌋ (BSW13303, BSW13304, BSW12145, BSW12184)

[FLS104] ⌈The function `Fls_MainFunction` shall set the job result to `MEMIF_JOB_FAILED` and report the error code `FLS_E_ERASE_FAILED` to the DEM if a flash erase job fails due to a hardware error. ⌋ (BSW00421)

[FLS105] ⌈The function `Fls_MainFunction` shall set the job result to `MEMIF_JOB_FAILED` and report the error code `FLS_E_WRITE_FAILED` to the DEM if a flash write job fails due to a hardware error. ⌋ (BSW00421)

[FLS106] ⌈The function `Fls_MainFunction` shall set the job result to `MEMIF_JOB_FAILED` and report the error code `FLS_E_READ_FAILED` to the DEM if a flash read job fails due to a hardware error. ⌋ (BSW00421)

[FLS154] ⌈The function `Fls_MainFunction` shall set the job result to `MEMIF_JOB_FAILED` and report the error code `FLS_E_COMPARE_FAILED` to the DEM if a flash compare job fails due to a hardware error. ⌋ (BSW00421)

[FLS200] ⌈The function `Fls_MainFunction` shall set the job result to `MEMIF_BLOCK_INCONSISTENT` if the compared data from a flash compare job are not equal. ⌋ ( )

[FLS022] ⌈If development error detection for the module Fls is enabled: After a flash block has been erased, the function `Fls_MainFunction` shall compare the contents of the addressed memory area against the value of an erased flash cell to check that the block has been completely erased. If this check fails, the function `Fls_MainFunction` shall set the FLS module's job result to `MEMIF_JOB_FAILED` and raise development error `FLS_E_VERIFY_ERASE_FAILED`. ⌋ (BSW12160)

[FLS055] ⌈If development error detection for the module Fls is enabled: Before writing a flash block, the function `Fls_MainFunction` shall compare the contents of the addressed memory area against the value of an erased flash cell to check that the block has been completely erased. If this check fails, the function `Fls_MainFunction` shall set the FLS module's job result to `MEMIF_JOB_FAILED` and raise development error `FLS_E_VERIFY_ERASE_FAILED`. ⌋ (BSW12158)

[FLS056] ⌈If development error detection for the module Fls is enabled: After writing a flash block, the function `Fls_MainFunction` shall compare the contents of the reprogrammed memory area against the contents of the provided application buffer to check that the block has been completely reprogrammed. If this check fails, the function `Fls_MainFunction` shall set the FLS module's job result to

MEMIF_JOB_FAILED and raise the development error FLS_E_VERIFY_WRITE_FAILED. ⌋ (BSW12141)

[FLS345] ⌈After a read, erase, write or compare job has been finished, the function Fls_MainFunction shall set the FLS module's job result to MEMIF_JOB_OK if it is currently in state MEMIF_JOB_PENDING. Otherwise, it shall leave the result unchanged. ⌋ ()

[FLS346] ⌈After a read, erase, write or compare job has been finished, the function Fls_MainFunction shall set the FLS module's state to MEMIF_IDLE and call the job end notification function if configured (see FLS307_Conf). ⌋ ()

[FLS232] ⌈The configuration parameter FlsUseInterrupts shall switch between interrupt and polling controlled job processing if this is supported by the flash memory hardware. ⌋ (BSW12129)

[FLS233] ⌈The FLS module's implementer shall locate the interrupt service routine in Fls_Irq.c. ⌋ (BSW12129)

[FLS234] ⌈If interrupt controlled job processing is supported and enabled with the configuration parameter FlsUseInterrupts, the interrupt service routine 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. ⌋ (BSW12129)

[FLS235] ⌈The function Fls_MainFunction shall process jobs without hardware interrupt support (e.g. read jobs). ⌋ ()

[FLS272] ⌈If development error detection for the module Fls is enabled: the function Fls_MainFunction shall provide a timeout monitoring for the currently running job, that is it shall supervise the deadline of the read / compare / erase or write job. ⌋ ()

[FLS359] ⌈If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the configured maximum erase time (see FLS298_Conf FlsEraseTime) has been exceeded. If this is the case, the function Fls_MainFunction shall raise the development error FLS_E_TIMEOUT. ⌋ ()

[FLS360] ⌈If development error detection for the module Fls is enabled: the function Fls_MainFunction shall check, whether the expected maximum write time (see note below) has been exceeded. If this is the case, the function Fls_MainFunction shall raise the development error FLS_E_TIMEOUT. ⌋ ()

Note: The expected maximum write time depends on the current mode of the Fls module (see FLS258), the configured number of bytes to write in this mode (see FLS278_Conf and FLS277_Conf respectively), the size of a single flash page (see

FLS281_Conf) and last the maximum time to write one flash page (see FLS301_Conf). The number of bytes to write divided by the size of one flash page yields the number of pages to write in one cycle. This multiplied with the maximum write time for one flash page gives you the expected maximum write time.

[FLS362] ⌈If development error detection for the module Fls is enabled: the function `Fls_MainFunction` shall check, whether the expected maximum read / compare time (see note below) has been exceeded. If this is the case, the function `Fls_MainFunction` shall raise the development error `FLS_E_TIMEOUT`.⌋()

Note: There are no published timings for read / compare (these would mostly depend on whether the flash device is internal or external e.g. connected via SPI). The solution would be similar as for write jobs above: the configured number of bytes to read (and to compare) is coupled to the expected read / compare times which should be supervised by the Fls_MainFunction. If this is not detailed enough there are two possibilities:
- *specify expected read / compare times (difficult because of the dependency mentioned above)*
- *leave read / compare jobs out of the timeout supervision (change FLS272).*

[FLS117] ⌈If development error detection for the module Fls is enabled: the function `Fls_MainFunction` shall check that the FLS module has been initialized. If this check fails, the function `Fls_MainFunction` shall raise the development error `FLS_E_UNINIT`.⌋()

[FLS196] ⌈The function `Fls_MainFunction` shall at the most issue one sector erase command (to the hardware) in each cycle. ⌋()

Note: The requirement above shall ensure that maximum one sector is erased sequentially within one cycle of the driver's main function. If the hardware is capable of erasing more than one sector in parallel, this shall not be restricted by this specification.

8.6 Expected Interfaces

This chapter lists all functions the Fls module requires from other modules.

8.6.1 Mandatory Interfaces

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

[FLS260] ⌈

API function	***Description***
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by

	BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.

⌋ ( )

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.

[FLS261] ⌈

API function	***Description***
Det_ReportError	Service to report development errors.

⌋ ( )

8.6.3 Configurable interfaces

In this chapter, all interfaces are listed for which the target function can 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. ⌋ ( )

[FLS262] ⌈

Service name:	Fee_JobEndNotification

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

Sync/Async:	Synchronous
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callback function is called when a job has been completed with a positive result.

⌋ ( )

[FLS167] ⌈The FLS module shall call the callback function `Fee_JobEndNotification` when the module has completed a job with a positive result:

- Read job finished & OK
- Write job finished & OK
- Erase job finished & OK
- Compare job finished & memory blocks are the same⌋ ( )

[FLS263] ⌈

Service name:	Fee_JobErrorNotification

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

Sync/Async:	Synchronous
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callback function is called when a job has been canceled or finished with negative result.

⌋ ( )

[FLS347] ⌈The FLS module shall call the callback function `Fee_JobErrorNotification` when the module has finished a job with a negative result:

- Read job failed
- Write job failed
- Erase job failed
- Compare job failed⌋ ( )

[FLS348] ⌈The FLS module shall call the callback function `Fee_JobErrorNotification` when the module has canceled an ongoing job:

- Read job aborted
- Write job aborted
- Erase job aborted
- Compare job aborted⌋ ( )

[FLS349] ⌈The FLS module shall call the callback function `Fee_JobErrorNotification` when the module has finished a compare job and the memory blocks differ:

- Compare job finished and memory blocks differ⌋ ( )

9 Sequence diagrams

9.1 Initialization

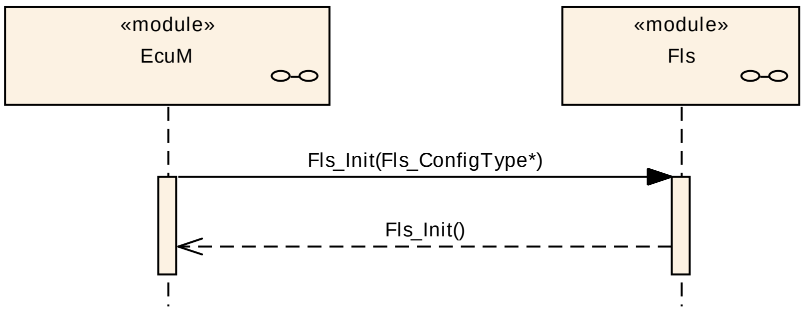

Figure 2: Flash driver initialization sequence

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`.

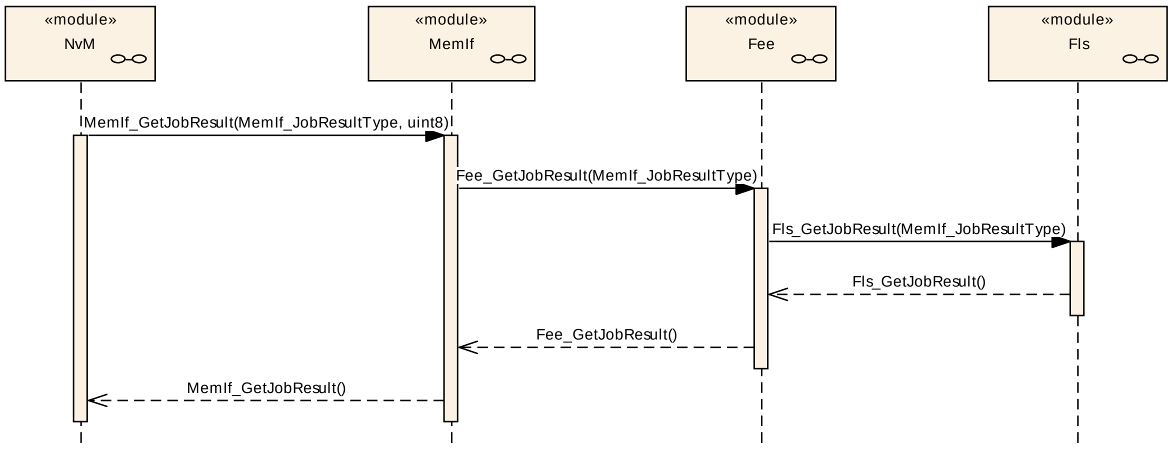
«module»
NvM
«module»
MemIf
«module»
Fee
«module»
Fls
MemIf_GetJobResult(MemIf_JobResultType, uint8)
Fee_GetJobResult(MemIf_JobResultType)
Fls_GetJobResult(MemIf_JobResultType)
Fls_GetJobResult()
Fee_GetJobResult()
MemIf_GetJobResult()

Figure 3: Fls_GetJobResult

9.3 Asynchronous functions

The following sequence diagram shows the flash write function (with the configuration option `FlsAcLoadOnJobStart` 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.

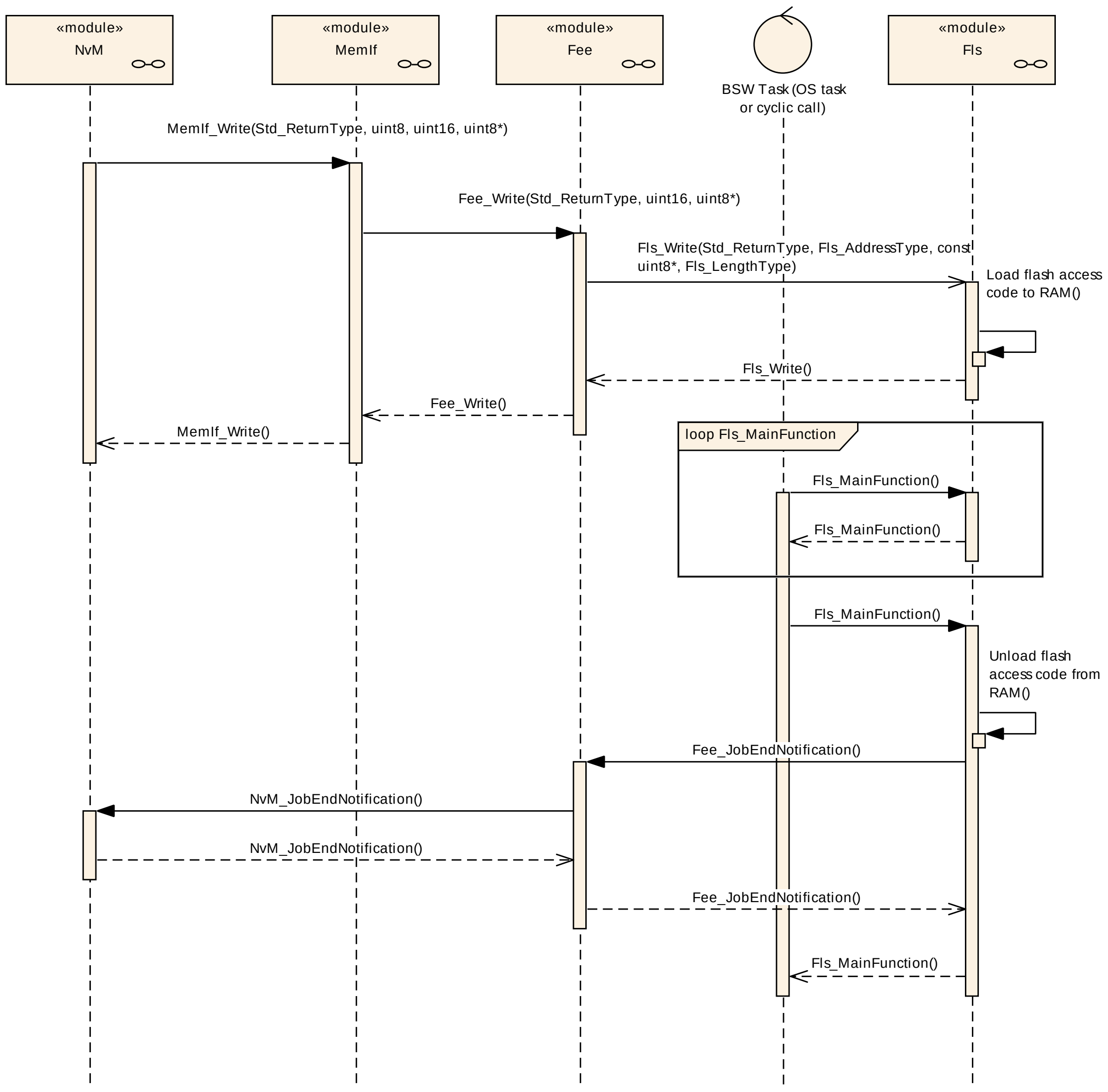

Figure 4: Flash write sequence, flash access code loaded on job start

9.4 Canceling a running job

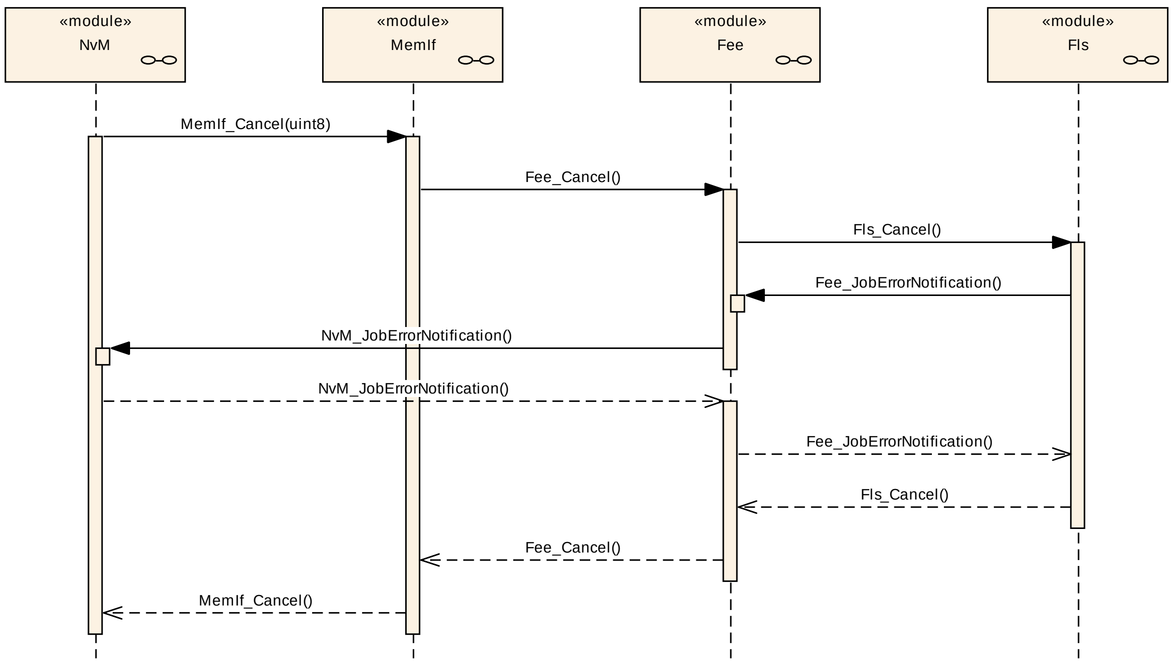

Figure 5: Canceling a running flash job

Note: *The FLS module's environment shall not call the function `Fls_Cancel` during a running `Fls_MainFunction` invocation.*

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 "Flash Driver".

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 [7]
  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

[FLS203] ⌈VARIANT-PRE-COMPILE
Only parameters with “Pre-compile time” configuration are allowed in this variant. ⌋ ( )

[FLS204] ⌈VARIANT-POST-BUILD
Parameters with “Pre-compile time”, “Link time” and “Post-build time” are allowed in this variant. ⌋ ( )

[FLS350] ⌈The initialization function of the FLS module shall always have a pointer as a parameter, even though for Variant VARIANT_PRECOMPILE no configuration set shall be given. Instead a null pointer shall be passed to the initialization function. ⌋ ( )

[FLS351] ⌈Only one interface for initialization shall be implemented (in contradiction to BSW00414) and it shall not depend on the modules configuration which interface the calling software module shall use. ⌋ ( )

10.2.2 Fls

SWS Item	**FLS001_Conf :**
Module Name	*Fls*
Module Description	Configuration of the Fls (internal or external flash driver) module. Its multiplicity describes the number of flash drivers present, so there will be one container for each flash driver in the ECUC template. When no flash driver is present then the multiplicity is 0.

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
FlsConfigSet	1	Container for runtime configuration parameters of the flash driver. Implementation Type: Fls_ConfigType.
FlsGeneral	1	Container for general parameters of the flash driver. These parameters are always pre-compile.
FlsPublishedInformation	1	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.

[⌈The table above 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. ⌋ (BSW00345, BSW12132)

10.2.3 FlsGeneral

SWS Item	**FLS172_Conf :**
Container Name	FlsGeneral{Fls_ModuleConfiguration}
Description	Container for general parameters of the flash driver. These parameters are always pre-compile.
Configuration Parameters	

SWS Item	**FLS284_Conf :**		
Name	FlsAcLoadOnJobStart {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. true: Flash access code loaded on job start / unloaded on job end or error. false: Flash access code not loaded to / unloaded from RAM at all.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS169_Conf :**

Name	FlsBaseAddress {FLS_BASE_ADDRESS}		
Description	The flash memory start address (see also FLS208 and FLS209). FLS169_Conf: This parameter defines the lower boundary for read / write / erase and compare jobs.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS285_Conf :**		
Name	FlsCancelApi {FLS_CANCEL_API}		
Description	Compile switch to enable and disable the Fls_Cancel function. true: API supported / function provided. false: API not supported / function not provided		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS286_Conf :**		
Name	FlsCompareApi {FLS_COMPARE_API}		
Description	Compile switch to enable and disable the Fls_Compare function. true: API supported / function provided. false: API not supported / function not provided		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS287_Conf :**		
Name	FlsDevErrorDetect {FLS_DEV_ERROR_DETECT}		
Description	Pre-processor switch to enable and disable development error detection (see FLS077). true: Development error detection enabled. false: Development error detection disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	true		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS288_Conf :**
Name	FlsDriverIndex
Description	Index of the driver, used by FEE.
Multiplicity	1

Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 254		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS289_Conf :**		
Name	FlsGetJobResultApi {FLS_GET_JOB_RESULT_API}		
Description	Compile switch to enable and disable the Fls_GetJobResult function. true: API supported / function provided. false: API not supported / function not provided		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS290_Conf :**		
Name	FlsGetStatusApi {FLS_GET_STATUS_API}		
Description	Compile switch to enable and disable the Fls_GetStatus function. true: API supported / function provided. false: API not supported / function not provided		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS291_Conf :**		
Name	FlsSetModeApi {FLS_SET_MODE_API}		
Description	Compile switch to enable and disable the Fls_SetMode function. true: API supported / function provided. false: API not supported / function not provided		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS170_Conf :**
Name	FlsTotalSize {FLS_TOTAL_SIZE}
Description	The total amount of flash memory in bytes (see also FLS208 and FLS209). FLS170_Conf: This parameter in conjunction with FLS_BASE_ADDRESS defines the upper boundary for read / write / erase and compare jobs.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS292_Conf :**		
Name	FlsUseInterrupts {FLS_USE_INTERRUPTS}		
Description	Job processing triggered by hardware interrupt. true: Job processing triggered by interrupt (hardware controlled). false: Job processing not triggered by interrupt (software controlled)		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module dependency: Only available if supported by underlying flash hardware		

SWS Item	**FLS293_Conf :**		
Name	FlsVersionInfoApi {FLS_VERSION_INFO_API}		
Description	Pre-processor switch to enable / disable the API to read out the modules version information. true: Version info API enabled. false: Version info API disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.4 FlsConfigSet

SWS Item	**FLS174_Conf :**
Container Name	FlsConfigSet{Fls_ConfigSet} [Multi Config Container]
Description	Container for runtime configuration parameters of the flash driver. Implementation Type: Fls_ConfigType.
Configuration Parameters	

SWS Item	**FLS270_Conf :**
Name	FlsAcErase {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.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**FLS305_Conf :**		
Name	FlsAcWrite {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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**FLS306_Conf :**		
Name	FlsCallCycle {FLS_CALL_CYCLE}		
Description	Cycle time of calls of the flash driver's main function (in seconds).		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. 1		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module dependency: Only relevant if deadline monitoring for internal functionality has to be done in software (e.g. erase / write timings)		

SWS Item	**FLS318_Conf :**		
Name	FlsDefaultMode {FLS_DEFAULT_MODE}		
Description	This parameter is the default FLS device mode after initialization. Implementation Type: MemIf_ModeType.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	MEMIF_MODE_FAST	The driver is working in fast mode (fast read access / SPI burst access).	
	MEMIF_MODE_SLOW	The driver is working in slow mode. (default)	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**FLS307_Conf :**		
Name	FlsJobEndNotification {FLS_JOB_END_NOTIFICATION}		
Description	Mapped to the job end notification routine provided by some upper layer module, typically the Fee module.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**FLS274_Conf :**		
Name	FlsJobErrorNotification {FLS_JOB_ERROR_NOTIFICATION}		
Description	Mapped to the job error notification routine provided by some upper layer module, typically the Fee module.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**FLS275_Conf :**		
Name	FlsMaxReadFastMode {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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	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.

SWS Item	**FLS276_Conf :**		
Name	FlsMaxReadNormalMode {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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	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.		

SWS Item	**FLS277_Conf :**		
Name	FlsMaxWriteFastMode {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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	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.		

SWS Item	**FLS278_Conf :**		
Name	FlsMaxWriteNormalMode {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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	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.		

SWS Item	**FLS279_Conf :**		
Name	FlsProtection {FLS_PROTECTION}		
Description	Erase/write protection settings. Only relevant if supported by hardware.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module dependency: Only relevant if supported by hardware.		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FlsDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
FlsExternalDriver	0..1	This container is present for external Flash drivers only. Internal Flash drivers do not use the parameter listed in this container, hence its multiplicity is 0 for internal drivers.
FlsSectorList	1	List of flashable sectors and pages.

[FLS352] ⌈The table above specifies the parameters that shall be located in an external data structure of type `Fls_ConfigType`.⌋ ( )

[FLS353] ⌈The organization and location of the data structure `Fls_ConfigType` shall be up to the implementer.⌋ ( )

[FLS354] ⌈The type declaration for `Fls_ConfigType` shall be located in the file `Fls.h`.⌋ ( )

[FLS355] ⌈Hardware or implementation specific parameters can be added to `Fls_ConfigType` if necessary.⌋ ( )

10.2.5 FlsDemEventParameterRefs

SWS Item	**FLS310_Conf :**
Container Name	FlsDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**FLS314_Conf :**		
Name	FLS_E_COMPARE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "Flash compare failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**FLS311_Conf :**		
Name	FLS_E_ERASE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "Flash erase failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**FLS313_Conf :**
Name	FLS_E_READ_FAILED
Description	Reference to the DemEventParameter which shall be issued when the error "Flash read failed (HW)" has

	occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**FLS315_Conf :**		
Name	FLS_E_UNEXPECTED_FLASH_ID		
Description	Reference to the DemEventParameter which shall be issued when the error "Expected hardware ID not matched" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**FLS312_Conf :**		
Name	FLS_E_WRITE_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "Flash write failed (HW)" has occurred.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.6 FlsExternalDriver

SWS Item	**FLS316_Conf :**
Container Name	FlsExternalDriver
Description	This container is present for external Flash drivers only. Internal Flash drivers do not use the parameter listed in this container, hence its multiplicity is 0 for internal drivers.
Configuration Parameters	

SWS Item	**FLS317_Conf :**		
Name	FlsSpiReference		
Description	Reference to SPI sequence (required for external Flash drivers).		
Multiplicity	1..*		
Type	Reference to [SpiSequence]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

No Included Containers

10.2.7 FlsSectorList

SWS Item	**FLS201_Conf :**
Container Name	FlsSectorList{Fls_SectorList}
Description	List of flashable sectors and pages.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FlsSector	1..*	Configuration description of a flashable sector

10.2.8 FlsSector

SWS Item	**FLS202_Conf :**
Container Name	FlsSector{Fls_Sector}
Description	Configuration description of a flashable sector
Configuration Parameters	

SWS Item	**FLS280_Conf :**		
Name	FlsNumberOfSectors {FLS_NUMBER_OF_SECTORS}		
Description	Number of continuous sectors with identical values for FlsSectorSize and FlsPageSize. The parameter FlsSectorStartAddress denotes the start address of the first sector.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**FLS281_Conf :**		
Name	FlsPageSize {FLS_PAGE_SIZE}		
Description	Size of one page of this sector. Implementation Type: Fls_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module dependency: The sector size has to be an integer multiple of the page size.		

SWS Item	**FLS282_Conf :**		
Name	FlsSectorSize {FLS_SECTOR_SIZE}		
Description	Size of this sector. Implementation Type: Fls_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module dependency: The sector size has to be an integer multiple of the page size.		

SWS Item	**FLS283_Conf :**		
Name	FlsSectorStartaddress {FLS_SECTOR_STARTADDRESS}		
Description	Start address of this sector. Implementation Type: Fls_AddressType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.3 Published Information

[FLS365] ⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋ ( )

Additional module-specific published parameters are listed below if applicable.

10.3.1 FlsPublishedInformation

SWS Item	**FLS178_Conf :**
Container Name	FlsPublishedInformation
Description	Additional published parameters not covered by CommonPublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
Configuration Parameters	

SWS Item	**FLS294_Conf :**		
Name	FlsAcLocationErase {FLS_AC_LOCATION_ERASE}		
Description	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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS295_Conf :**		
Name	FlsAcLocationWrite {FLS_AC_LOCATION_WRITE}		
Description	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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS296_Conf :**		
Name	FlsAcSizeErase {FLS_AC_SIZE_ERASE}		
Description	Number of bytes in RAM needed for the erase flash access code.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS297_Conf :**		
Name	FlsAcSizeWrite {FLS_AC_SIZE_WRITE}		
Description	Number of bytes in RAM needed for the write flash access code.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS298_Conf :**
Name	FlsEraseTime {FLS_ERASE_TIME}
Description	Maximum time to erase one complete flash sector.

Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS299_Conf :**		
Name	FlsErasedValue {FLS_ERASED_VALUE}		
Description	The contents of an erased flash memory cell.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS300_Conf :**		
Name	FlsExpectedHwId {FLS_EXPECTED_HW_ID}		
Description	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.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS198_Conf :**		
Name	FlsSpecifiedEraseCycles {FLS_SPECIFIED_ERASE_CYCLES}		
Description	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, voltage, ...) 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).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

SWS Item	**FLS301_Conf :**	
Name	FlsWriteTime {FLS_WRITE_TIME}	
Description	Maximum time to program one complete flash page.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	`0 .. INF`	
Default value	--	

ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: module		

No Included Containers

11 Changes w.r.t. Release 3.0

11.1 Deleted SWS Items

SWS Item	*Rationale*
FLS177	Second occurrence (copy-paste) of requirement deleted.
FLS049	Requirement made into a note (ID deleted).

11.2 Replaced SWS Items

SWS Item	*replaced by SWS Item*	*Rationale*
FLS073	FLS308, FLS309	Splitting of requirements
FLS004	FLS310, FLS311, FLS312, FLS313, FLS314, FLS315, FLS316, FLS317, FLS318, FLS319	Splitting of requirements (Note: FLS004 cannot be removed since it's used as a link for the generated table)
FLS248	FLS320, FLS321, FLS322	Splitting of requirements (Note: FLS248 cannot be removed since it's used as a link for the generated table)
FLS016	FLS323, FLS324	Splitting of requirements
FLS271	FLS325, FLS326	Splitting of requirements
FLS219	FLS327, FLS328, FLS329, FLS330	Splitting of requirements
FLS224	FLS331, FLS332, FLS333, FLS334	Splitting of requirements
FLS032	FLS335, FLS336	Splitting of requirements
FLS237	FLS337, FLS338, FLS339, FLS340	Splitting of requirements
FLS242	FLS341, FLS342, FLS343, FLS344	Splitting of requirements
FLS052	FLS345, FLS346	Splitting of requirements
FLS168	FLS347, FLS348, FLS349	Splitting of requirements
FLS194	FLS350, FLS351	Splitting of requirements
FLS173	FLS352, FLS353, FLS354, FLS355	Splitting of requirements

11.3 Changed SWS Items

SWS Item	*Rationale*
FLS273	Coloring in chaptzer 12.4 changed (to automatic).
FLS032	Name of module state corrected (leftover from R1.0)
FLS174_Conf	Type of configuration parameter changed (FLS270_Conf, FLS305_Conf)
FLS203, FLS204	Wording about variants adapted
FLS307_Conf, FLS274_Conf	Multiplicity of notification functions.
FLS315, FLS316, FLS317, FLS318. FLS319	Copy-paste-error with production errors fixed.
FLS357	DET error removed from requirement.

FLS169_Conf, FLS170_Conf, FLS198_Conf, FLS270_Conf, FLS275_Conf, FLS276_Conf, FLS277_Conf, FLS278_Conf, FLS279_Conf, FLS280_Conf, FLS281_Conf, FLS282_Conf, FLS283_Conf, FLS294_Conf, FLS295_Conf, FLS296_Conf, FLS297_Conf, FLS299_Conf, FLS305_Conf,	Range of configuration parameters restricted to meaningful min. & max. values.
FLS004	Development error for timeout supervision added.

11.4 Added SWS Items

SWS Item	***Rationale***
FLS356	Added check for initialization to Fls_Cancel.
FLS357	Added check for initialization to Fls_GetStatus.
FLS358	Added check for initialization to Fls_GetJobResult.
FLS359, FLS360, FLS361, FLS362	Requirements for timeout supervision added
FLS365	Rework of Published Information

12 Changes w.r.t. Release 4.0

12.1 Deleted SWS Items

SWS Item	***Rationale***
FLS111	Superfluous requirement deleted.

12.2 Replaced SWS Items

SWS Item	***replaced by SWS Item***	***Rationale***

12.3 Changed SWS Items

SWS Item	***Rationale***
FLS306_Conf	RParameter range adapted
FLS321	Dem_Data_Types.h renamed to Dem_Types.h
FLS040	Erase job removed from requirement.
FLS364	
FLS312_Conf FLS313_Conf FLS314_Conf FLS315_Conf	
FLS298_Conf FLS301_Conf	

12.4 Added SWS Items

SWS Item	***Rationale***
FLS318_Conf	Configuration parameter added.
FLS316_Conf, FLS317_Conf	Container with SPI reference added
FLS363	DET error if NULL pointer is passed as an argument.
FLS364	

13 Not applicable requirements

[FLS366] ⌈These requirements are not applicable to this specification.⌋ (BSW00344, BSW170, BSW00387, BSW00398, BSW00375, BSW00416, BSW168, BSW00423, BSW00424, BSW00426, BSW00427, BSW00428, BSW00429, BSW00431, BSW00433, BSW00434, BSW00336, BSW00339, BSW00422, BSW00420, BSW00417, BSW161, BSW162, BSW00324, BSW005, BSW00415, BSW00326, BSW00342, BSW160, BSW007, BSW00300, BSW00347, BSW00307, BSW00314, BSW00370, BSW00348, BSW00353, BSW00361, BSW00302, BSW00328, BSW00312, BSW006, BSW00304, BSW00355, BSW00378, BSW00306, BSW00308, BSW00309, BSW00371, BSW00359, BSW00360, BSW00330, BSW009, BSW00401, BSW172, BSW010, BSW00333, BSW00321, BSW00341, BSW00334, BSW12267, BSW12163, BSW12462, BSW12463, BSW12468, BSW12069, BSW12063, BSW12064, BSW12067, BSW12078, BSW12078, BSW12083, BSW12149)