Document Title	Specification of Flash Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	25

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R22-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2022-11-24	R22-11	AUTOSAR Release Management	• Migrated FLS_E_BUSY to runtime error • Proper implementation of TPS_STDT_00042
2021-11-25	R21-11	AUTOSAR Release Management	• Removed SWS_Fls_00109 • FlsCallCycle renamed to FlsMainFunctionPeriod and moved it from FlsConfigSte to FlsGeneral
2020-11-30	R20-11	AUTOSAR Release Management	• Editorial changes
2019-11-28	R19-11	AUTOSAR Release Management	• Draft status of ECUC_Fls_00323 removed • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Added support for MCALMulticoreDistribution
2017-12-08	4.3.1	AUTOSAR Release Management	• Removed references to HIS • Renamed "default error" to "development error" • Introduction of runtime errors • Configuration of instance ID for instanciated modules

2016-11-30	4.3.0	AUTOSAR Release Management	• Updated tracing information • Internal buffer alignment clarified • Error handling refined, new configuration parameters added
2015-07-31	4.2.2	AUTOSAR Release Management	• Debugging support marked as obsolete • Error classification reworked • Reference to DEM removed • Description for configuration parameter FlsUseInterrupts clarified
2014-10-31	4.2.1	AUTOSAR Release Management	• Requirements linked to features and BSW requirements
2014-03-31	4.1.3	AUTOSAR Release Management	• Requirements for NULL pointer check during Fls_Init removed • Minor formatting changes
2013	4.1.2	AUTOSAR Release Management	• Timing requirement removed from module's main function • Fls_GetStatus returns MEMIF_UNINIT if module is not initialized • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneral • Scope attribute in tables in chapter 10 added • Production errors changed to extended production errors • Reuqirement IDs for type definitions added
2011-12-22	4.0.3	AUTOSAR Administration	• References to HW specification erros corrected • Range of configuration parameters adapted • Consistency checking reformulated • Module short name changed

2010-09-30	3.1.5	AUTOSAR Administration	• Configuration parameter FlsDefaultMode added • Container with SPI reference added • Check for NULL pointer added
2010-02-02	3.1.4	AUTOSAR Administration	• References to AUTOSAR Standard Errors added • Range of configuration parameters restricted • Multiplicity of notifaction routines corrected • Serveral typing and formatting errors corrected • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2008-02-01	3.0.2	AUTOSAR Administration	• Table formatting corrected
2007-12-21	3.0.1	AUTOSAR Administration	• NULL pointer check added to Fls_Compare • NULL pointer check detailed (in general) • Restriction removed to allow reinitialization of module • Tables in chapters 8 and 10 generated from UML model • Document meta information extended • small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• File include structure updated • Type usage corrected • Compare Job results adapted • API towards DEM corrected • Legal disclaimer revised • Release Notes added • "Advise for users" revised • "Revision Information" added

2006-05-16	2.0	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
2005-05-31	1.0	AUTOSAR Administration	• Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module [1] 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 EEP-ROM 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.

[SWS_Fls_00088] ⌈The functional requirements [2] and the functional scope are the same for both internal and external drivers. Hence the API is semantically identical.⌋ *(SRS_Fls_12147, SRS_Fls_12148)*

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Flash Driver module that are not included in the [3, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
DET	Default 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 Input documents & related standards and norms

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[2] Requirements on Flash Driver
AUTOSAR_SRS_FlashDriver

[3] Glossary
AUTOSAR_TR_Glossary

[4] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[5] Requirements on Memory Hardware Abstraction Layer
AUTOSAR_SRS_MemoryHWAbstractionLayer

[6] Specification of SPI Handler/Driver
AUTOSAR_SWS_SPIHandlerDriver

[7] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture

[8] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [4, SWS BSW General], which is also valid for Flash Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Flash Driver.

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 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].

5.2 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 Tracing

The following tables reference the requirements specified in <CITA-TIONS_OF_CONTRIBUTED_DOCUMENTS> and links to the fulfillment of these. Please note that if column "Satisfied by" is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_BRF_01064]	AUTOSAR BSW shall provide callback functions in order to access upper layer modules	[SWS_Fls_00110] [SWS_Fls_00147] [SWS_Fls_00167] [SWS_Fls_00262] [SWS_Fls_00263] [SWS_Fls_00273] [SWS_Fls_00347] [SWS_Fls_00348] [SWS_Fls_00349]
[RS_BRF_01076]	AUTOSAR basic software shall perform module local error recovery to the extent possible	[SWS_Fls_00272] [SWS_Fls_00359] [SWS_Fls_00360] [SWS_Fls_00361] [SWS_Fls_00362] [SWS_Fls_00371] [SWS_Fls_00373]
[RS_BRF_01144]	AUTOSAR shall support configuration parameters which allow to trade interrupt response time against runtime	[SWS_Fls_00233] [SWS_Fls_00234]
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_Fls_00205] [SWS_Fls_00206]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_Fls_NA_00366]
[SRS_BSW_00006]	The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	[SWS_Fls_NA_00366]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Fls_NA_00366]
[SRS_BSW_00009]	All Basic SW Modules shall be documented according to a common standard.	[SWS_Fls_NA_00366]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_Fls_NA_00366]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Fls_00014] [SWS_Fls_00086] [SWS_Fls_00191] [SWS_Fls_00249]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_Fls_NA_00366]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_Fls_NA_00366]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_Fls_NA_00366]

$\bigtriangledown$

Requirement	Description	Satisfied by
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_Fls_00193] [SWS_Fls_00232]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Fls_00205] [SWS_Fls_00206]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_Fls_NA_00366]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_Fls_NA_00366]
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_Fls_00183] [SWS_Fls_00184] [SWS_Fls_00185] [SWS_Fls_00186] [SWS_Fls_00187]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_Fls_NA_00366]
[SRS_BSW_00300]	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	[SWS_Fls_NA_00366]
[SRS_BSW_00302]	All AUTOSAR Basic Software Modules shall only export information needed by other modules	[SWS_Fls_NA_00366]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Fls_NA_00366]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Fls_NA_00366]
[SRS_BSW_00307]	Global variables naming convention	[SWS_Fls_NA_00366]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Fls_NA_00366]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_Fls_NA_00366]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Fls_NA_00366]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_Fls_NA_00366]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Fls_NA_00366]

△

Requirement	Description	Satisfied by
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Fls_00015] [SWS_Fls_00020] [SWS_Fls_00021] [SWS_Fls_00026] [SWS_Fls_00027] [SWS_Fls_00097] [SWS_Fls_00098] [SWS_Fls_00157] [SWS_Fls_00158] [SWS_Fls_00205] [SWS_Fls_00206] [SWS_Fls_00363]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_Fls_00193]
[SRS_BSW_00327]	Error values naming convention	[SWS_Fls_00310] [SWS_Fls_00312] [SWS_Fls_00313] [SWS_Fls_00314] [SWS_Fls_00315] [SWS_Fls_00316] [SWS_Fls_00317] [SWS_Fls_00318] [SWS_Fls_00319]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_Fls_NA_00366]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Fls_NA_00366]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Fls_00310] [SWS_Fls_00312] [SWS_Fls_00313] [SWS_Fls_00314] [SWS_Fls_00315] [SWS_Fls_00316] [SWS_Fls_00317] [SWS_Fls_00318] [SWS_Fls_00319]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_Fls_NA_00366]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_Fls_NA_00366]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Fls_NA_00366]
[SRS_BSW_00337]	Classification of development errors	[SWS_Fls_00310] [SWS_Fls_00312] [SWS_Fls_00313] [SWS_Fls_00314] [SWS_Fls_00315] [SWS_Fls_00316] [SWS_Fls_00317] [SWS_Fls_00318] [SWS_Fls_00319]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_Fls_00104] [SWS_Fls_00105] [SWS_Fls_00106] [SWS_Fls_00154] [SWS_Fls_00260] [SWS_Fls_NA_00366]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_Fls_NA_00366]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_Fls_NA_00366]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Fls_NA_00366]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Fls_NA_00366]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Fls_NA_00366]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_Fls_NA_00366]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_Fls_NA_00366]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Fls_NA_00366]
[SRS_BSW_00361]	No description	[SWS_Fls_NA_00366]
[SRS_BSW_00371]	No description	[SWS_Fls_NA_00366]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Fls_NA_00366]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Fls_NA_00366]
[SRS_BSW_00385]	List possible error notifications	[SWS_Fls_00004] [SWS_Fls_00104] [SWS_Fls_00105] [SWS_Fls_00106] [SWS_Fls_00154] [SWS_Fls_00310] [SWS_Fls_00312] [SWS_Fls_00313] [SWS_Fls_00314] [SWS_Fls_00315] [SWS_Fls_00316] [SWS_Fls_00317] [SWS_Fls_00318] [SWS_Fls_00319]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_Fls_00352]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_Fls_00248] [SWS_Fls_00368] [SWS_Fls_00369] [SWS_Fls_00370]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_Fls_NA_00366]
[SRS_BSW_00401]	Documentation of multiple instances of configuration parameters shall be available	[SWS_Fls_NA_00366]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Fls_00014]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Fls_00014]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_Fls_00065] [SWS_Fls_00066] [SWS_Fls_00099] [SWS_Fls_00240] [SWS_Fls_00268] [SWS_Fls_00356] [SWS_Fls_00358] [SWS_Fls_00382] [SWS_Fls_00383]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Fls_00259]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Fls_NA_00366]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Fls_NA_00366]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	[SWS_Fls_NA_00366]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the Dem	[SWS_Fls_NA_00366]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_Fls_NA_00366]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_Fls_NA_00366]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_Fls_NA_00366]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_Fls_NA_00366]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_Fls_NA_00366]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_Fls_NA_00366]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Fls_00269]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_Fls_NA_00366]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_Fls_00352] [SWS_Fls_00353] [SWS_Fls_00355]
[SRS_BSW_00466]	Classification of extended production errors	[SWS_Fls_00104] [SWS_Fls_00105] [SWS_Fls_00106] [SWS_Fls_00154]
[SRS_BSW_00469]	Fault detection and healing of production errors and extended production errors	[SWS_Fls_00260]
[SRS_BSW_00483]	BSW Modules shall handle buffer alignments internally	[SWS_Fls_00389]
[SRS_Fls_12107]	The external flash driver shall check if the configured flash type matches with the hardware flash ID	[SWS_Fls_00144]
[SRS_Fls_12132]	Flash driver shall be statically configurable	[SWS_Fls_00048] [SWS_Fls_00208] [SWS_Fls_00209] [SWS_Fls_00216] [SWS_Fls_00217]
[SRS_Fls_12134]	The flash driver shall provide an asynchronous read function	[SWS_Fls_00001] [SWS_Fls_00035] [SWS_Fls_00097] [SWS_Fls_00098] [SWS_Fls_00236] [SWS_Fls_00238] [SWS_Fls_00239] [SWS_Fls_00254] [SWS_Fls_00256] [SWS_Fls_00337] [SWS_Fls_00338] [SWS_Fls_00339] [SWS_Fls_00340]

▽

Requirement	Description	Satisfied by
[SRS_Fls_12135]	The flash driver shall provide an asynchronous write function	[SWS_Fls_00001] [SWS_Fls_00026] [SWS_Fls_00027] [SWS_Fls_00035] [SWS_Fls_00146] [SWS_Fls_00223] [SWS_Fls_00225] [SWS_Fls_00226] [SWS_Fls_00251] [SWS_Fls_00254] [SWS_Fls_00331] [SWS_Fls_00332] [SWS_Fls_00333] [SWS_Fls_00334] [SWS_Fls_00385]
[SRS_Fls_12136]	The flash driver shall provide an asynchronous erase function	[SWS_Fls_00001] [SWS_Fls_00020] [SWS_Fls_00021] [SWS_Fls_00035] [SWS_Fls_00145] [SWS_Fls_00218] [SWS_Fls_00220] [SWS_Fls_00221] [SWS_Fls_00250] [SWS_Fls_00254] [SWS_Fls_00327] [SWS_Fls_00328] [SWS_Fls_00329] [SWS_Fls_00330]
[SRS_Fls_12137]	The flash driver shall provide a synchronous cancel function	[SWS_Fls_00033] [SWS_Fls_00035] [SWS_Fls_00183] [SWS_Fls_00229] [SWS_Fls_00230] [SWS_Fls_00252] [SWS_Fls_00254] [SWS_Fls_00335] [SWS_Fls_00336]
[SRS_Fls_12138]	The flash driver shall provide a synchronous status function	[SWS_Fls_00034] [SWS_Fls_00184] [SWS_Fls_00253]
[SRS_Fls_12141]	The flash driver shall verify written data	[SWS_Fls_00056] [SWS_Fls_00200]
[SRS_Fls_12143]	The flash driver shall handle only one job at one time	[SWS_Fls_00002] [SWS_Fls_00003] [SWS_Fls_00023] [SWS_Fls_00030] [SWS_Fls_00033] [SWS_Fls_00036] [SWS_Fls_00100] [SWS_Fls_00323] [SWS_Fls_00324]
[SRS_Fls_12144]	The flash driver shall provide a function that has to be called for job processing	[SWS_Fls_00037] [SWS_Fls_00038] [SWS_Fls_00039] [SWS_Fls_00196] [SWS_Fls_00220] [SWS_Fls_00225] [SWS_Fls_00235] [SWS_Fls_00238] [SWS_Fls_00243] [SWS_Fls_00255] [SWS_Fls_00272] [SWS_Fls_00345] [SWS_Fls_00346] [SWS_Fls_00374] [SWS_Fls_00375] [SWS_Fls_00376] [SWS_Fls_00377] [SWS_Fls_00378] [SWS_Fls_00379]
[SRS_Fls_12145]	The job processing function of the flash driver shall process only as much data as the flash hardware can handle	[SWS_Fls_00040]
[SRS_Fls_12147]	The same requirements shall apply for an external and internal flash driver	[SWS_Fls_00088]
[SRS_Fls_12148]	The external flash driver shall have a semantically identical API as an internal flash driver	[SWS_Fls_00088]
[SRS_Fls_12149]	The source code of the external flash driver shall be independent from the underlying microcontroller	[SWS_Fls_NA_00366]
[SRS_Fls_12158]	Before writing, the flash driver shall verify if the addressed memory area has been erased	[SWS_Fls_00055]

△

Requirement	Description	Satisfied by
[SRS_Fls_12159]	The write and erase functions of the Flash driver shall check the passed address parameters	[SWS_Fls_00020] [SWS_Fls_00021] [SWS_Fls_00026] [SWS_Fls_00027] [SWS_Fls_00097] [SWS_Fls_00098] [SWS_Fls_00380] [SWS_Fls_00381] [SWS_Fls_00385]
[SRS_Fls_12160]	After execution of an erase job, the flash driver shall verify that the addressed block has been erased completely	[SWS_Fls_00022]
[SRS_Fls_12184]	The flash driver shall limit the read access blocking times to the configured time	[SWS_Fls_00040]
[SRS_Fls_12193]	The flash driver shall load the code that accesses the flash hardware to RAM whenever an erase or write job is started	[SWS_Fls_00137] [SWS_Fls_00140] [SWS_Fls_00141] [SWS_Fls_00214]
[SRS_Fls_12194]	The flash driver shall execute the code that accesses the flash hardware from RAM	[SWS_Fls_00211] [SWS_Fls_00212] [SWS_Fls_00213] [SWS_Fls_00215]
[SRS_Fls_13300]	The flash driver shall remove the code that accesses the flash hardware from RAM after the current job has been finished or canceled	[SWS_Fls_00143]
[SRS_Fls_13301]	The flash driver shall provide an asynchronous compare function	[SWS_Fls_00001] [SWS_Fls_00150] [SWS_Fls_00151] [SWS_Fls_00152] [SWS_Fls_00153] [SWS_Fls_00186] [SWS_Fls_00241] [SWS_Fls_00243] [SWS_Fls_00244] [SWS_Fls_00257] [SWS_Fls_00341] [SWS_Fls_00342] [SWS_Fls_00343] [SWS_Fls_00344]
[SRS_Fls_13302]	The flash driver shall provide a synchronous selection function	[SWS_Fls_00155] [SWS_Fls_00156] [SWS_Fls_00187] [SWS_Fls_00258]
[SRS_Fls_13303]	In normal mode, one cycle of the job processing function of the flash driver shall limit the block size to the default block size	[SWS_Fls_00040]
[SRS_Fls_13304]	In fast mode, one cycle of the job processing function of the flash driver shall limit the block size to the maximum block size	[SWS_Fls_00040]
[SRS_MemHwAb_-14005]	The FEE and EA modules shall provide upper layer modules with a virtual 32bit address space	[SWS_Fls_00209] [SWS_Fls_00216] [SWS_Fls_00217]
[SRS_SPAL_12057]	All driver modules shall implement an interface for initialization	[SWS_Fls_00014]
[SRS_SPAL_12063]	All driver modules shall only support raw value mode	[SWS_Fls_NA_00366]
[SRS_SPAL_12064]	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	[SWS_Fls_NA_00366]
[SRS_SPAL_12067]	All driver modules shall set their wake-up conditions depending on the selected operation mode	[SWS_Fls_NA_00366]
[SRS_SPAL_12069]	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	[SWS_Fls_NA_00366]

▽

△

Requirement	Description	Satisfied by
[SRS_SPAL_12078]	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	[SWS_Fls_NA_00366]
[SRS_SPAL_12163]	All driver modules shall implement an interface for de-initialization	[SWS_Fls_NA_00366]
[SRS_SPAL_12267]	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	[SWS_Fls_NA_00366]
[SRS_SPAL_12462]	The register initialization settings shall be published	[SWS_Fls_NA_00366]
[SRS_SPAL_12463]	The register initialization settings shall be combined and forwarded	[SWS_Fls_NA_00366]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 General design rules

[SWS_Fls_00001] ⌈The FLS module shall offer asynchronous services for operations on flash memory (read/erase/write).⌋*(SRS_Fls_12134, SRS_Fls_12135, SRS_Fls_12136, SRS_Fls_13301)*

[SWS_Fls_00002] ⌈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.⌋*(SRS_Fls_12143)*

[SWS_Fls_00003] ⌈The FLS module shall not ensure data consistency of the given application buffer.⌋*(SRS_Fls_12143)*

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

[SWS_Fls_00205] ⌈The FLS module shall check static configuration parameters statically (at the latest during compile time) for correctness.⌋*(SRS_BSW_00323, SRS_BSW_00167, SRS_BSW_00004)*

[SWS_Fls_00206] ⌈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).⌋*(SRS_BSW_00323, SRS_BSW_00167, SRS_BSW_00004)*

[SWS_Fls_00208] ⌈The FLS module shall combine all available flash memory areas into one linear address space (denoted by the parameters `FlsBaseAddress` and `FlsTotalSize`).⌋*(SRS_Fls_12132)*

[SWS_Fls_00209] ⌈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.⌋*(SRS_Fls_12132, SRS_MemHwAb_14005)*

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.

[SWS_Fls_00389] ⌈The FLS module shall handle data buffer alignment internally. Instead of imposing any requirements on RAM buffers' alignments (as they are uint8*), it shall handle passed pointers as being just byte-aligned.⌋*(SRS_BSW_00483)*

[SWS_Fls_00390] ⌈If more than one instance of the flash driver is used in an ECU, the individual instances have to be given a unique instance ID. This instance ID shall be configured as the parameter `FlsDriverIndex`. If only one instance of the flash driver is used in an ECU, this instance shall have the parameter `FlsDriverIndex` configured as 0.⌋*()*

7.2 External flash driver

[SWS_Fls_00144] ⌈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 Default Error Tracer (DET), set the FLS module status to `FLS_E_UNINIT` and shall not initialize itself.⌋*(SRS_Fls_-12107)*

A complete list of required parameters is specified in the SPI Handler/Driver Software Specification [6] (Chapter "Configuration Specification", marked as "SPI User").

7.3 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.

[SWS_Fls_00137] ⌈The FLS module's implementer shall place the code of the flash access routines into a separate C-module Fls_ac.c.⌋*(SRS_Fls_12193)*

[SWS_Fls_00215] ⌈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).⌋*(SRS_Fls_12194)*

[SWS_Fls_00211] ⌈The FLS module's implementer shall keep the execution time for the flash access code as short as possible.⌋*(SRS_Fls_12194)*

[SWS_Fls_00140] ⌈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.⌋*(SRS_Fls_12193)*

[SWS_Fls_00141] ⌈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.⌋*(SRS_Fls_12193)*

[SWS_Fls_00212] ⌈The FLS module's main processing routine shall execute the flash access code routines.⌋*(SRS_Fls_12194)*

[SWS_Fls_00213] ⌈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.⌋*(SRS_Fls_12194)*

[SWS_Fls_00143] ⌈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.⌋ *(SRS_Fls_13300)*

[SWS_Fls_00214] ⌈The FLS module shall only load the access code to the RAM if the access code cannot be executed out of flash ROM.⌋ *(SRS_Fls_12193)*

7.4   Error Classification

Section "Error Handling" of the document [4] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.4.1   Development Errors

[SWS_Fls_00004] ⌈

Type of error	Related error code	Error value
API service called with wrong parameter	FLS_E_PARAM_CONFIG	0x01
API service called with wrong parameter	FLS_E_PARAM_ADDRESS	0x02
API service called with wrong parameter	FLS_E_PARAM_LENGTH	0x03
API service called with wrong parameter	FLS_E_PARAM_DATA	0x04
API service called without module initialization	FLS_E_UNINIT	0x05
API service called with NULL pointer	FLS_E_PARAM_POINTER	0x0a
–	FLS_E_ALREADY_INITIALIZED	0x0b

⌋ *(SRS_BSW_00385)*

[SWS_Fls_00310] ⌈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`.⌋ *(SRS_BSW_-00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

7.4.2 Runtime Errors

[SWS_Fls_91001] ⌈

Type of error	Related error code	Error value
API service called while driver still busy	FLS_E_BUSY	0x06
Erase verification (blank check) failed	FLS_E_VERIFY_ERASE_FAILED	0x07
Write verification (compare) failed	FLS_E_VERIFY_WRITE_FAILED	0x08
Timeout exceeded	FLS_E_TIMEOUT	0x09

⌋*()*

[SWS_Fls_00312] ⌈The runtime error code `FLS_E_BUSY` shall be reported when an API service is called while the module is still busy.⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00313] ⌈The runtime error code `FLS_E_VERIFY_ERASE_FAILED` shall be reported when the erase verification function is enabled (by the compile switch `FlsEraseVerificationEnabled`) and the erase verification function (blankcheck) failed.⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00314] ⌈The runtime error code `FLS_E_VERIFY_WRITE_FAILED` shall be reported when the write verification function is enabled (by the compile switch `FlsWriteVerificationEnabled`) and the write verification function (compare) failed.⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00361] ⌈The runtime error code `FLS_E_TIMEOUT` shall be reported when the timeout supervision function is enabled (by the compile switch `FlsTimeoutSupervisionEnabled`) and the timeout supervision of a read, write, erase or compare job (in hardware) failed.⌋*(RS_BRF_01076)*

7.4.3 Transient Faults

[SWS_Fls_91002] ⌈

Type of error	Related error code	Error value
Flash erase failed (HW)	FLS_E_ERASE_FAILED	0x01
Flash write failed (HW)	FLS_E_WRITE_FAILED	0x02
Flash read failed (HW)	FLS_E_READ_FAILED	0x03
Flash compare failed (HW)	FLS_E_COMPARE_FAILED	0x04
Expected hardware ID not matched (see SWS_Fls_00144)	FLS_E_UNEXPECTED_FLASH_ID	0x05

⌋*()*

[SWS_Fls_00315] ⌈The transient fault code `FLS_E_ERASE_FAILED` shall be reported when the flash erase function failed (in hardware).⌋*(SRS_BSW_00337, SRS_BSW_-00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00316] ⌈The transient fault code `FLS_E_WRITE_FAILED` shall be reported when the flash write function failed (in hardware).⌋*(SRS_BSW_00337, SRS_BSW_-00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00317] ⌈The transient fault code `FLS_E_READ_FAILED` shall be reported when the flash read function failed (in hardware).⌋*(SRS_BSW_00337, SRS_BSW_-00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00318] ⌈The transient fault code `FLS_E_COMPARE_FAILED` shall be reported when the flash compare function failed (in hardware).⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

[SWS_Fls_00319] ⌈The transient fault code `FLS_E_UNEXPECTED_FLASH_ID` shall be reported when the expected flash ID is not matched (see [SWS_Fls_00144]).⌋*(SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00327, SRS_BSW_00331)*

7.4.4 Production Errors

There are no production errors.

7.4.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

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

[SWS_Fls_00248] ⌈

Module	Header File	Imported Type
MemIf	MemIf.h	MemIf_JobResultType (obsolete)
	MemIf.h	MemIf_ModeType (obsolete)
	MemIf.h	MemIf_StatusType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋*(SRS_BSW_00392)*

8.2 Type definitions

8.2.1 Fls_ConfigType

[SWS_Fls_00368] ⌈

Name	Fls_ConfigType	
Kind	Structure	
Elements	Hardware dependend structure	
	Type	–
	Comment	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.	
Available via	Fls.h	

⌋*(SRS_BSW_00392)*

8.2.2 Fls_AddressType

[SWS_Fls_00369] ⌈

Name	Fls_AddressType		
Kind	Type		
Derived from	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.
Available via	Fls.h

⌋*(SRS_BSW_00392)*

[SWS_Fls_00216] ⌈The type `Fls_AddressType` shall have 0 as lower limit for each flash device.⌋*(SRS_Fls_12132, SRS_MemHwAb_14005)*

[SWS_Fls_00217] ⌈The FLS module shall add a device specific base address to the address type `Fls_AddressType` if necessary.⌋*(SRS_Fls_12132, SRS_MemHwAb_-14005)*

8.2.3 Fls_LengthType

[SWS_Fls_00370] ⌈

Name	Fls_LengthType		
Kind	Type		
Derived from	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.		
Available via	Fls.h		

⌋*(SRS_BSW_00392)*

8.3 Function definitions

8.3.1 `Fls_Init`

[SWS_Fls_00249] ⌈

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.
Available via	Fls.h

⌋*(SRS_BSW_00101)*

[SWS_Fls_00014] ⌈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.⌋*(SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00101, SRS_SPAL_12057)*

[SWS_Fls_00191] ⌈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.⌋*(SRS_BSW_00101)*

[SWS_Fls_00086] ⌈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.⌋*(SRS_BSW_00101)*

[SWS_Fls_00015] ⌈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`.⌋*(SRS_BSW_00323)*

[SWS_Fls_00323] ⌈The function `Fls_Init` shall set the FLS module state to MEMIF_IDLE after having finished the FLS module initialization.⌋*(SRS_Fls_12143)*

[SWS_Fls_00324] ⌈The function `Fls_Init` shall set the flash job result to MEMIF_JOB_OK after having finished the FLS module initialization.⌋*(SRS_Fls_12143)*

[SWS_Fls_00268] ⌈If runtime error detection for the module Fls is enabled: the function `Fls_Init` shall check module initialization status. If the module has already been initialized, the function `Fls_Init` shall raise the development error `FLS_E_-ALREADY_INITIALIZED`.⌋*(SRS_BSW_00406)*

[SWS_Fls_00048] ⌈If supported by hardware, the function `Fls_Init` shall set the flash memory erase/write protection as provided in the configuration set.⌋*(SRS_Fls_-12132)*

8.3.2 Fls_Erase

[SWS_Fls_00250] ⌈

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 - Target Address
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).	
Available via	Fls.h	

⌋*(SRS_Fls_12136)*

[SWS_Fls_00218] ⌈The job of the function `Fls_Erase` shall erase one or more complete flash sectors.⌋*(SRS_Fls_12136)*

[SWS_Fls_00327] ⌈The function `Fls_Erase` shall copy the given parameters to FLS module internal variables and initiate an erase job.⌋*(SRS_Fls_12136)*

[SWS_Fls_00328] ⌈After initiating the erase job, the function `Fls_Erase` shall set the FLS module status to MEMIF_BUSY.⌋*(SRS_Fls_12136)*

[SWS_Fls_00329] ⌈After initiating the erase job, the function `Fls_Erase` shall set the job result to MEMIF_JOB_PENDING.⌋*(SRS_Fls_12136)*

[SWS_Fls_00330] ⌈After initiating the erase job, the function `Fls_Erase` shall return with E_OK.⌋*(SRS_Fls_12136)*

[SWS_Fls_00220] ⌈The FLS module shall execute the job of the function `Fls_Erase` asynchronously within the FLS module's main function.⌋*(SRS_Fls_12136, SRS_Fls_-12144)*

[SWS_Fls_00221] ⌈The job of the function `Fls_Erase` shall erase a flash memory block starting from the flash memory base address + `TargetAddress` of size Length.

Note: Length will be rounded up to the next full sector boundary since only complete flash sectors can be erased.⌋*(SRS_Fls_12136)*

[SWS_Fls_00020] ⌈If development error detection for the module Fls is enabled: the function `Fls_Erase` shall check that the erase start address (flash memory base address + `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 func-

tion `Fls_Erase` shall reject the erase request, raise the development error `FLS_E_-PARAM_ADDRESS` and return with E_NOT_OK.⌋*(SRS_BSW_00323, SRS_Fls_12136, SRS_Fls_12159)*

[SWS_Fls_00021] ⌈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.⌋*(SRS_BSW_00323, SRS_Fls_-12136, SRS_Fls_12159)*

[SWS_Fls_00065] ⌈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.⌋*(SRS_BSW_00406)*

[SWS_Fls_00023] ⌈If runtime 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.⌋*(SRS_Fls_12143)*

[SWS_Fls_00145] ⌈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.⌋*(SRS_Fls_12136)*

8.3.3 `Fls_Write`

[SWS_Fls_00251] ⌈

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.	
Available via	Fls.h	

⌋*(SRS_Fls_12135)*

[SWS_Fls_00223] ⌈The job of the function `Fls_Write` shall write one or more complete flash pages to the flash device.⌋*(SRS_Fls_12135)*

[SWS_Fls_00331] ⌈The function `Fls_Write` shall copy the given parameters to Fls module internal variables and initiate a write job.⌋*(SRS_Fls_12135)*

[SWS_Fls_00332] ⌈After initiating the write job, the function `Fls_Write` shall set the FLS module status to MEMIF_BUSY.⌋*(SRS_Fls_12135)*

[SWS_Fls_00333] ⌈After initiating the write job, the function `Fls_Write` shall set the job result to MEMIF_JOB_PENDING.⌋*(SRS_Fls_12135)*

[SWS_Fls_00334] ⌈After initiating the write job, the function `Fls_Write` shall return with E_OK.⌋*(SRS_Fls_12135)*

[SWS_Fls_00225] ⌈The FLS module shall execute the write job of the function `Fls_-Write` asynchronously within the FLS module's main function.⌋*(SRS_Fls_12135, SRS_Fls_12144)*

[SWS_Fls_00226] ⌈The job of the function `Fls_Write` shall program a flash memory block with data provided via `SourceAddressPtr` starting from the flash memory base address + `TargetAddress` of size Length.⌋*(SRS_Fls_12135)*

[SWS_Fls_00026] ⌈If development error detection for the module Fls is enabled: the function `Fls_Write` shall check that the write start address (flash memory base address + `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.⌋*(SRS_BSW_00323, SRS_Fls_12135, SRS_Fls_12159)*

[SWS_Fls_00027] ⌈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.⌋*(SRS_BSW_00323, SRS_Fls_12135, SRS_Fls_12159)*

[SWS_Fls_00066] ⌈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.⌋*(SRS_BSW_00406)*

[SWS_Fls_00030] ⌈If runtime 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.⌋*(SRS_Fls_12143)*

[SWS_Fls_00157] ⌈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.⌋*(SRS_BSW_00323)*

[SWS_Fls_00146] ⌈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.⌋*(SRS_Fls_12135)*

8.3.4 `Fls_Cancel`

[SWS_Fls_00252] ⌈

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.
Available via	Fls.h

⌋*(SRS_Fls_12137)*

[SWS_Fls_00229] ⌈The function `Fls_Cancel` shall cancel an ongoing flash read, write, erase or compare job.⌋*(SRS_Fls_12137)*

[SWS_Fls_00230] ⌈The function `Fls_Cancel` shall abort a running job synchronously so that directly after returning from this function a new job can be started.⌋*(SRS_Fls_-12137)*

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).

[SWS_Fls_00335] ⌈The function `Fls_Cancel` shall reset the FLS module's internal job processing variables (like address, length and data pointer).⌋*(SRS_Fls_12137)*

[SWS_Fls_00336] ⌈The function `Fls_Cancel` shall set the FLS module state to MEMIF_IDLE.⌋*(SRS_Fls_12137)*

[SWS_Fls_00033] ⌈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.⌋*(SRS_Fls_12137, SRS_Fls_12143)*

[SWS_Fls_00147] ⌈If configured, the function `Fls_Cancel` shall call the error notification function to inform the caller about the cancellation of a job.⌋*(RS_BRF_01064)*

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

[SWS_Fls_00183] ⌈The function `Fls_Cancel` shall be pre-compile time configurable On/Off by the configuration parameter `FlsCancelApi`.⌋*(SRS_BSW_00171, SRS_- Fls_12137)*

[SWS_Fls_00356] ⌈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.⌋*(SRS_BSW_00406)*

8.3.5 `Fls_GetStatus`

[SWS_Fls_00253] ⌈

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.	
Available via	Fls.h	

⌋*(SRS_Fls_12138)*

[SWS_Fls_00034] ⌈The function `Fls_GetStatus` shall return the FLS module state synchronously.⌋*(SRS_Fls_12138)*

[SWS_Fls_00184] ⌈The function `Fls_GetStatus` shall be pre-compile time configurable On/Off by the configuration parameter `FlsGetStatusApi`.⌋*(SRS_Fls_12138, SRS_BSW_00171)*

Note: The function `Fls_GetStatus` may be called before the module has been initialized in which case it shall return MEMIF_UNINIT.

8.3.6 Fls_GetJobResult

[SWS_Fls_00254] ⌈

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.
Available via	Fls.h

⌋*(SRS_Fls_12134, SRS_Fls_12135, SRS_Fls_12136, SRS_Fls_12137)*

[SWS_Fls_00035] ⌈The function `Fls_GetJobResult` shall return the result of the last job synchronously⌋*(SRS_Fls_12134, SRS_Fls_12135, SRS_Fls_12136, SRS_-Fls_12137)*

[SWS_Fls_00036] ⌈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.⌋*(SRS_Fls_12143)*

[SWS_Fls_00185] ⌈The function `Fls_GetJobResult` shall be pre-compile time configurable On/Off by the configuration parameter `FlsGetJobResultApi`.⌋*(SRS_-BSW_00171)*

[SWS_Fls_00358] ⌈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.⌋*(SRS_BSW_00406)*

8.3.7 Fls_Read

[SWS_Fls_00256] ⌈

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 - Source Address
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.	
Available via	Fls.h	

⌋*(SRS_Fls_12134)*

[SWS_Fls_00236] ⌈The function `Fls_Read` shall read from flash memory.⌋*(SRS_-Fls_12134)*

[SWS_Fls_00337] ⌈The function `Fls_Read` shall copy the given parameters to FLS module internal variables and initiate a read job.⌋*(SRS_Fls_12134)*

[SWS_Fls_00338] ⌈After initiating a read job, the function `Fls_Read` shall set the FLS module status to MEMIF_BUSY.⌋*(SRS_Fls_12134)*

[SWS_Fls_00339] ⌈After initiating a read job, the function `Fls_Read` shall set the FLS module job result to MEMIF_JOB_PENDING.⌋*(SRS_Fls_12134)*

[SWS_Fls_00340] ⌈After initiating a read job, the function `Fls_Read` shall return with E_OK.⌋*(SRS_Fls_12134)*

[SWS_Fls_00238] ⌈The FLS module shall execute the read job of the function `Fls_Read` asynchronously within the FLS module's main function.⌋*(SRS_Fls_12134, SRS_Fls_12144)*

[SWS_Fls_00239] ⌈The read job of the function `Fls_Read` shall copy a continuous flash memory block starting from the flash memory base address + `SourceAddress` of size `Length` to the buffer pointed to by `TargetAddressPtr`.⌋*(SRS_Fls_12134)*

[SWS_Fls_00097] ⌈If development error detection for the module Fls is enabled: the function `Fls_Read` shall check that the read start address (flash memory base address + `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.⌋*(SRS_BSW_-00323, SRS_Fls_12134, SRS_Fls_12159)*

[SWS_Fls_00098] ⌈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.⌋*(SRS_-BSW_00323, SRS_Fls_12134, SRS_Fls_12159)*

[SWS_Fls_00099] ⌈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.⌋*(SRS_BSW_00406)*

[SWS_Fls_00100] ⌈If runtime 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.⌋*(SRS_Fls_12143)*

[SWS_Fls_00158] ⌈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.⌋*(SRS_BSW_00323)*

[SWS_Fls_00240] ⌈The FLS module's environment shall only call the function `Fls_-Read` after the FLS module has been initialized.⌋*(SRS_BSW_00406)*

8.3.8 `Fls_Compare`

[SWS_Fls_00257] ⌈

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 - Source Address
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.	
Available via	Fls_Com.h	

⌋*(SRS_Fls_13301)*

[SWS_Fls_00241] ⌈The function `Fls_Compare` shall compare the contents of an area of flash memory with that of an application data buffer.⌋*(SRS_Fls_13301)*

[SWS_Fls_00341] ⌈The function `Fls_Compare` shall copy the given parameters to Fls module internal variables and initiate a compare job.⌋*(SRS_Fls_13301)*

[SWS_Fls_00342] ⌈After initiating the compare job, the function `Fls_Compare` shall set the status to MEMIF_BUSY.⌋*(SRS_Fls_13301)*

[SWS_Fls_00343] ⌈After initiating the compare job, the function `Fls_Compare` shall set the job result to MEMIF_JOB_PENDING.⌋*(SRS_Fls_13301)*

[SWS_Fls_00344] ⌈After initiating the compare job, the function `Fls_Compare` shall return with E_OK.⌋*(SRS_Fls_13301)*

[SWS_Fls_00243] ⌈The FLS module shall execute the job of the function `Fls_-Compare` asynchronously within the FLS module's main function.⌋*(SRS_Fls_13301, SRS_Fls_12144)*

[SWS_Fls_00244] ⌈The job of the function `Fls_Compare` shall compare a continuous flash memory block starting from the flash memory base address + `SourceAddress` of size `Length` with the buffer pointed to by `TargetAddressPtr`.⌋*(SRS_Fls_13301)*

[SWS_Fls_00150] ⌈If development error detection for the module Fls is enabled: the function `Fls_Compare` shall check that the compare start address (flash memory base address + `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.⌋ *(SRS_Fls_13301)*

[SWS_Fls_00151] ⌈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.⌋*(SRS_Fls_13301)*

[SWS_Fls_00152] ⌈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.⌋*(SRS_Fls_13301)*

[SWS_Fls_00153] ⌈If runtime 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.⌋*(SRS_Fls_13301)*

[SWS_Fls_00273] ⌈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.⌋*(RS_BRF_01064)*

[SWS_Fls_00186] ⌈The function `Fls_Compare` shall be pre-compile time configurable On/Off by the configuration parameter `FlsCompareApi`.⌋*(SRS_BSW_00171, SRS_Fls_13301)*

8.3.9 `Fls_SetMode`

[SWS_Fls_00258] ⌈

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.	
Available via	Fls.h	

⌋*(SRS_Fls_13302)*

[SWS_Fls_00155] ⌈The function `Fls_SetMode` shall set the FLS module's operation mode to the given "Mode" parameter.⌋*(SRS_Fls_13302)*

[SWS_Fls_00156] ⌈If runtime 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`.⌋*(SRS_Fls_13302)*

[SWS_Fls_00187] ⌈The function `Fls_SetMode` shall be pre-compile time configurable On/Off by the configuration parameter `FlsSetModeApi`.⌋*(SRS_BSW_00171, SRS_Fls_13302)*

8.3.10 Fls_GetVersionInfo

[SWS_Fls_00259] ⌈

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.	
Available via	Fls.h	

⌋*(SRS_BSW_00407)*

[SWS_Fls_00363] ⌈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.⌋*(SRS_-BSW_00323)*

8.3.11 Fls_BlankCheck

[SWS_Fls_00371] ⌈

Service Name	Fls_BlankCheck	

Syntax	Std_ReturnType Fls_BlankCheck (
	   Fls_AddressType TargetAddress,
	   Fls_LengthType Length
	)	

Service ID [hex]	0x0a	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TargetAddress	Address in flash memory from which the blank check should be started. Min.: 0 Max.: FLS_SIZE - 1
	Length	Number of bytes to be checked for erase pattern. Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: request for blank checking has been accepted by the module E_NOT_OK: request for blank checking has not been accepted by the module

▽

△

Description	The function Fls_BlankCheck shall verify, whether a given memory area has been erased but not (yet) programmed. The function shall limit the maximum number of checked flash cells per main function cycle to the configured value FlsMaxReadNormalMode or FlsMaxReadFastMode respectively.
Available via	Fls.h

⌋*(RS_BRF_01076)*

[SWS_Fls_00373] ⌈The function `Fls_BlankCheck` shall verify, whether a given memory area has been erased but not (yet) re-programmed.⌋*(RS_BRF_01076)*

[SWS_Fls_00374] ⌈The function `Fls_BlankCheck` shall copy the given parameters to FLS module internal variables and initiate the verification job.⌋*(SRS_Fls_12144)*

[SWS_Fls_00375] ⌈After initiating the verification job, the function `Fls_BlankCheck` shall set the FLS module status to MEMIF_BUSY.⌋*(SRS_Fls_12144)*

[SWS_Fls_00376] ⌈After initiating the verification job, the function `Fls_BlankCheck` shall set the FLS module job result to MEMIF_JOB_PENDING.⌋*(SRS_Fls_12144)*

[SWS_Fls_00377] ⌈After initiating the verification job, the function `Fls_BlankCheck` shall return with E_OK.⌋*(SRS_Fls_12144)*

[SWS_Fls_00378] ⌈The FLS module shall execute the verification job of the function `Fls_BlankCheck` asynchronously within the FLS module's main function.⌋*(SRS_Fls_12144)*

[SWS_Fls_00379] ⌈The verification job of the function `Fls_BlankCheck` shall check, that the continuous flash memory area starting from the flash memory base address + `TargetAddress` of size `Length` is erased.⌋*(SRS_Fls_12144)*

[SWS_Fls_00380] ⌈If development error detection for the module FLS is enabled; the function `Fls_BlankCheck` shall check that the verification start address (flash memory base address + `TargetAddress`) lies within the specified lower and upper flash address boundaries. If this check fails, the function `Fls_BlankCheck` shall reject the verification job, raise the development error `FLS_E_PARAM_ADDRESS` and return with E_NOT_OK.⌋*(SRS_Fls_12159)*

[SWS_Fls_00381] ⌈If development error detection for the module FLS is enabled: the function `Fls_BlankCheck` shall check that the given length is greater than 0 and that the verification end address (verification start address + length) lies within the specified upper flash address boundary. If this check fails, the function `Fls_BlankCheck` shall reject the verification job, raise the development error `FLS_E_PARAM_LENGTH` and return with E_NOT_OK.⌋*(SRS_Fls_12159)*

[SWS_Fls_00382] ⌈If development error detection for the module FLS is enabled: the function `Fls_BlankCheck` shall check that the driver has been initialized. If this check fails, the function `Fls_BlankCheck` shall reject the verification request, raise the development error `FLS_E_UNINIT` and return with E_NOT_OK.⌋*(SRS_BSW_00406)*

[SWS_Fls_00383] ⌈If runtime error detection for the module FLS is enabled: the function `Fls_BlankCheck` shall check that the driver is currently not busy. If this check

fails, the function `Fls_BlankCheck` shall reject the verification request, raise the development error `FLS_E_BUSY` and return with E_NOT_OK.⌋*(SRS_BSW_00406)*

8.4 Callback notifications

This is a list of functions provided for other 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 [7].

[SWS_Fls_00193] ⌈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.⌋*(SRS_-BSW_00164, SRS_BSW_00325)*

8.5 Scheduled functions

This chapter lists all functions provided by the Fls module and called directly by the Basic Software Module Scheduler. The following functions shall have no return value and no paramete. All functions shall be non reentrant.

[SWS_Fls_00269] ⌈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.⌋*(SRS_-BSW_00432)*

8.5.1 `Fls_MainFunction`

[SWS_Fls_00255] ⌈

Service Name	Fls_MainFunction

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

Service ID [hex]	0x06
Description	Performs the processing of jobs.
Available via	SchM_Fls.h

⌋*(SRS_Fls_12144)*

[SWS_Fls_00037] ⌈The function `Fls_MainFunction` shall perform the processing of the flash read, write, erase and compare jobs.⌋*(SRS_Fls_12144)*

[SWS_Fls_00038] ⌈When a job has been initiated, the FLS module's environment shall call the function `Fls_MainFunction` cyclically until the job is finished.⌋*(SRS_Fls_-12144)*

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

[SWS_Fls_00039] ⌈The function `Fls_MainFunction` shall return without any action if no job is pending.⌋*(SRS_Fls_12144)*

[SWS_Fls_00040] ⌈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).⌋*(SRS_Fls_-13303, SRS_Fls_13304, SRS_Fls_12145, SRS_Fls_12184)*

[SWS_Fls_00104] ⌈The function `Fls_MainFunction` shall set the job result to MEMIF_JOB_FAILED and report the error code `FLS_E_ERASE_FAILED` to the DET if a flash erase job fails due to a hardware error.⌋*(SRS_BSW_00339, SRS_BSW_00385, SRS_BSW_00466)*

[SWS_Fls_00105] ⌈The function `Fls_MainFunction` shall set the job result to MEMIF_JOB_FAILED and report the error code `FLS_E_WRITE_FAILED` to the DET if a flash write job fails due to a hardware error.⌋*(SRS_BSW_00339, SRS_BSW_00385, SRS_BSW_00466)*

[SWS_Fls_00106] ⌈The function `Fls_MainFunction` shall set the job result to MEMIF_JOB_FAILED and report the error code `FLS_E_READ_FAILED` to the DET if a flash read job fails due to a hardware error.⌋*(SRS_BSW_00339, SRS_BSW_00385, SRS_BSW_00466)*

[SWS_Fls_00154] ⌈The function `Fls_MainFunction` shall set the job result to MEMIF_JOB_FAILED and report the error code `FLS_E_COMPARE_FAILED` to the DET if a flash compare job fails due to a hardware error.⌋*(SRS_BSW_00339, SRS_BSW_-00385, SRS_BSW_00466)*

[SWS_Fls_00385] ⌈If the underlying flash technology requires a certain alignment of the read address or length information and if the address and/or length parameter for a read or compare Job are not correctly aligned, the function `Fls_MainFunction` shall internally compensate for this missing alignment, that is the function `Fls_Main-Function` shall provide byte-wise read access to the flash memory, regardless of any alignment restrictions imposed by the Hardware.⌋*(SRS_Fls_12135, SRS_Fls_12159)*

[SWS_Fls_00200] ⌈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.⌋*(SRS_Fls_12141)*

[SWS_Fls_00022] ⌈If erase verification is enabled (compile switch `FlsEraseVer-ificationEnabled` set to TRUE): 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 mod-

ule's job result to MEMIF_JOB_FAILED and raise the runtime error `FLS_E_VERIFY_-ERASE_FAILED`.⌋*(SRS_Fls_12160)*

[SWS_Fls_00055] ⌈If erase verification is enabled (compile switch `FlsEraseVerificationEnabled` set to TRUE): 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 the runtime error `FLS_E_VERIFY_ERASE_FAILED`.⌋*(SRS_Fls_12158)*

[SWS_Fls_00056] ⌈If write verification is enabled (compile switch `FlsWriteVerificationEnabled` set to TRUE): 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 runtime error `FLS_E_VERIFY_WRITE_FAILED`.⌋*(SRS_Fls_12141)*

[SWS_Fls_00345] ⌈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.⌋*(SRS_Fls_12144)*

[SWS_Fls_00346] ⌈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 [ECUC_Fls_00307]).⌋*(SRS_Fls_12144)*

[SWS_Fls_00232] ⌈The configuration parameter `FlsUseInterrupts` shall switch between interrupt and polling controlled job processing if this is supported by the flash memory hardware.⌋*(SRS_BSW_00164)*

[SWS_Fls_00233] ⌈The FLS module's implementer shall locate the interrupt service routine in Fls_Irq.c.⌋*(RS_BRF_01144)*

[SWS_Fls_00234] ⌈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.⌋*(RS_BRF_01144)*

[SWS_Fls_00235] ⌈The function `Fls_MainFunction` shall process jobs without hardware interrupt support (e.g. read jobs).⌋*(SRS_Fls_12144)*

[SWS_Fls_00272] ⌈If timeout supervision is enabled (compile switch `FlsTimeoutSupervisionEnabled` set to TRUE): 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.⌋*(SRS_Fls_12144, RS_BRF_01076)*

[SWS_Fls_00359] ⌈If timeout supervision is enabled (compile switch `FlsTimeout-SupervisionEnabled` set to TRUE): the function `Fls_MainFunction` shall check, whether the configured maximum erase time (see [ECUC_Fls_00298] `FlsErase-Time`) has been exceeded. If this is the case, the function `Fls_MainFunction` shall raise the runtime error `FLS_E_TIMEOUT`.⌋*(RS_BRF_01076)*

[SWS_Fls_00360] ⌈If timeout supervision is enabled (compile switch `FlsTimeout-SupervisionEnabled` set to TRUE): 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 runtime error `FLS_E_-TIMEOUT`.⌋*(RS_BRF_01076)*

Note: The expected maximum write time depends on the current mode of the Fls module (see [SWS_Fls_00258]), the configured number of bytes to write in this mode (see [ECUC_Fls_00278] and [ECUC_Fls_00277] respectively), the size of a single flash page (see [ECUC_Fls_00281]) and last the maximum time to write one flash page (see [ECUC_Fls_00301]). 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.

[SWS_Fls_00362] ⌈If timeout supervision is enabled (compile switch `FlsTimeout-SupervisionEnabled` set to TRUE): 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 runtime error `FLS_E_TIMEOUT`.⌋*(RS_BRF_01076)*

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 [SWS_Fls_00272]).

[SWS_Fls_00196] ⌈The function `Fls_MainFunction` shall at the most issue one sector erase command (to the hardware) in each cycle.⌋*(SRS_Fls_12144)*

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

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

8.6.1 Mandatory interfaces

Note: This section defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_Fls_00260] ⌈

API Function	Header File	Description
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋*(SRS_BSW_00469, SRS_BSW_00339)* Note: If the flash device is connected via SPI, also the SPI interfaces [6] 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 section defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_Fls_00261] ⌈

API Function	Header File	Description
Det_ReportError	Det.h	Service to report development errors.

⌋*()*

8.6.3 Configurable interfaces

In this section, all interfaces are listed where the target function could be configured. The target function is usually a callback function. The names of this kind of interfaces are not fixed because they are configurable.

[SWS_Fls_00110] ⌈The callback notifications shall have no parameters and no return value.⌋*(RS_BRF_01064)*

[SWS_Fls_00262] ⌈

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.
Available via	Fee.h

⌋*(RS_BRF_01064)*

[SWS_Fls_00167] ⌈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

⌋*(RS_BRF_01064)*

[SWS_Fls_00263] ⌈

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.
Available via	Fee.h

⌋*(RS_BRF_01064)*

[SWS_Fls_00347] ⌈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

⌋*(RS_BRF_01064)*

[SWS_Fls_00348] ⌈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

⌋*(RS_BRF_01064)*

[SWS_Fls_00349] ⌈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

⌋*(RS_BRF_01064)*

9 Sequence diagrams

9.1 Initialization
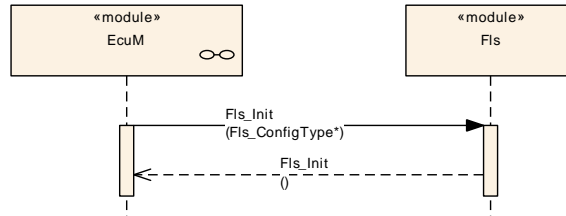
Figure 9.1: Flash driver initialization sequence

9.2 Synchronous functions

The following sequence diagram shows the function `Fls_GetJobResult` as an example for the synchronous functions of this module. The same sequence applies also to the functions `Fls_GetStatus` and `Fls_SetMode`.
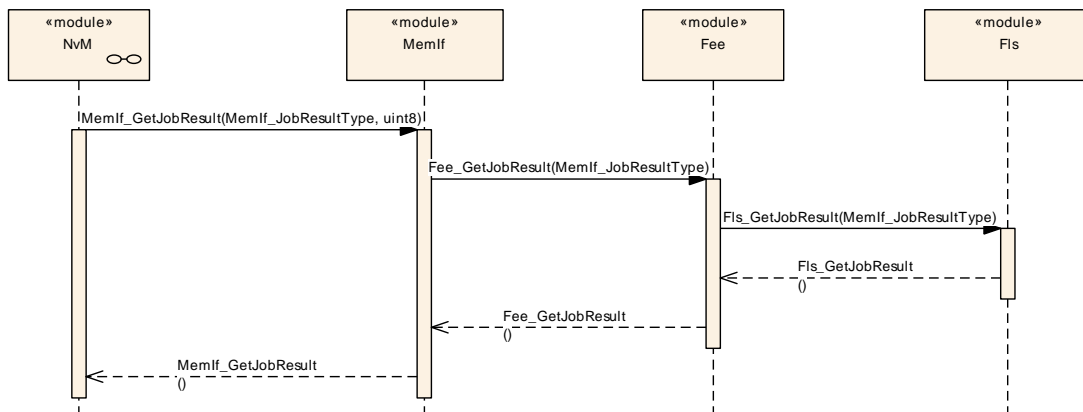
Figure 9.2: `Fls_GetJobResult`

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.
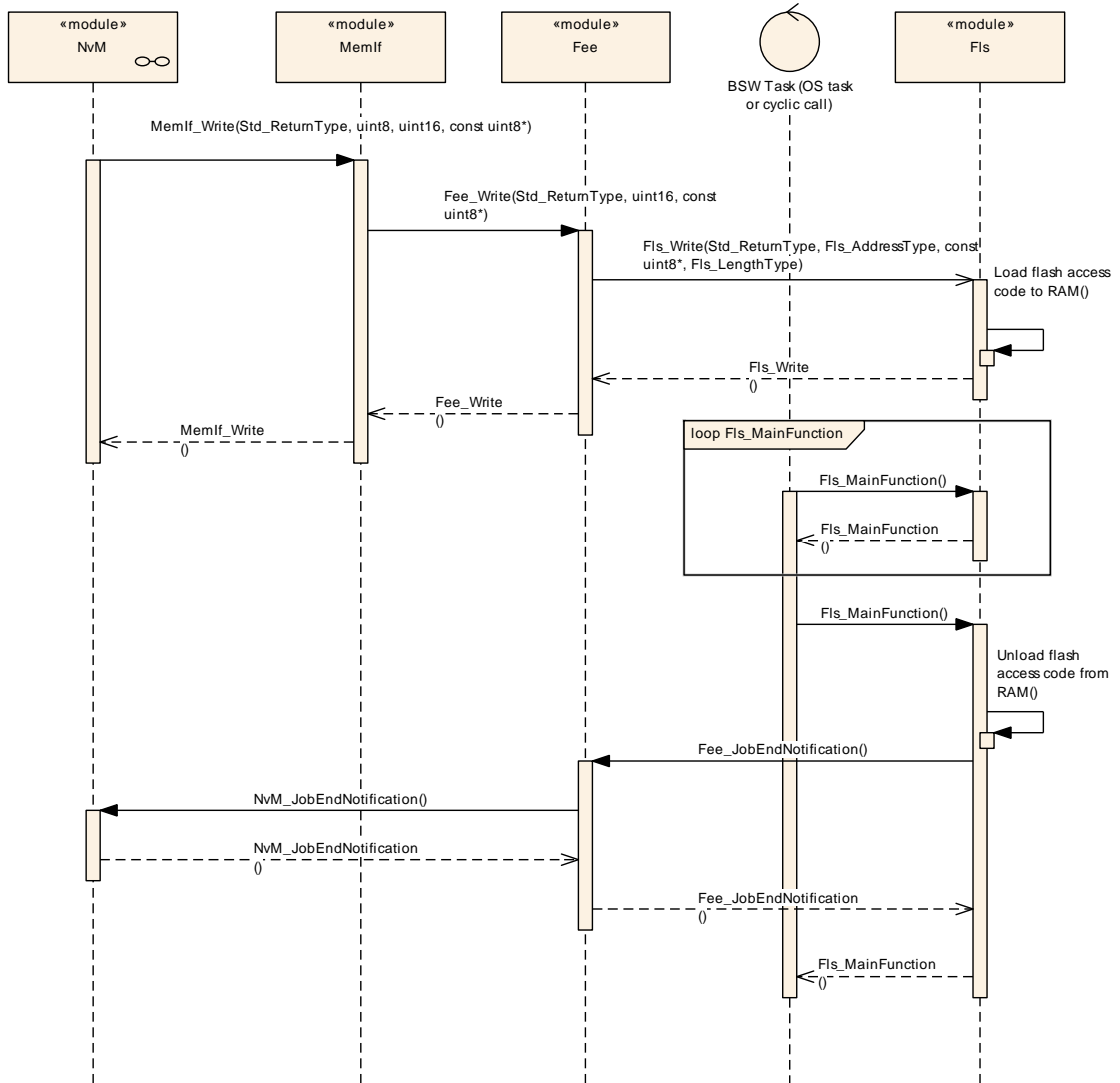
Figure 9.3: Flash write sequence, flash access code loaded on job start

9.4 Canceling a running job
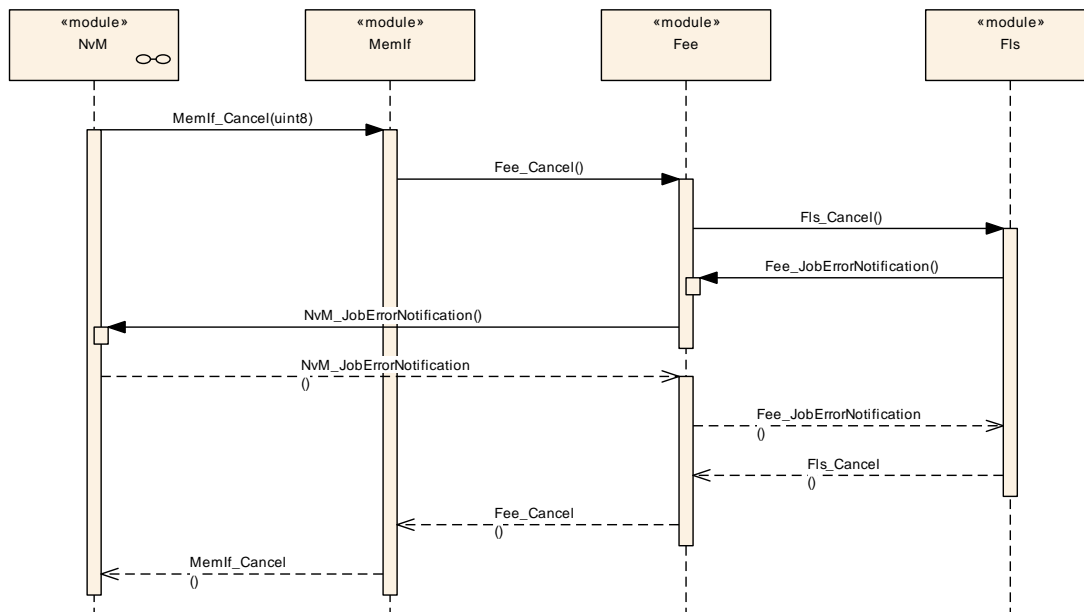
Figure 9.4: Canceling a running flash job

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 Fls.

Chapter 10.3 specifies published information of the module Fls.

10.1 How to read this chapter

For details refer to the chapter 10.1 "Introduction to configuration specification" in SWS_BSWGeneral [4].

10.2 Containers and configuration parameters

The following chapters summarize all configuration [8] parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

10.2.1 Fls

SWS Item	[ECUC_Fls_00001]
Module Name	Fls
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.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

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 Common PublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
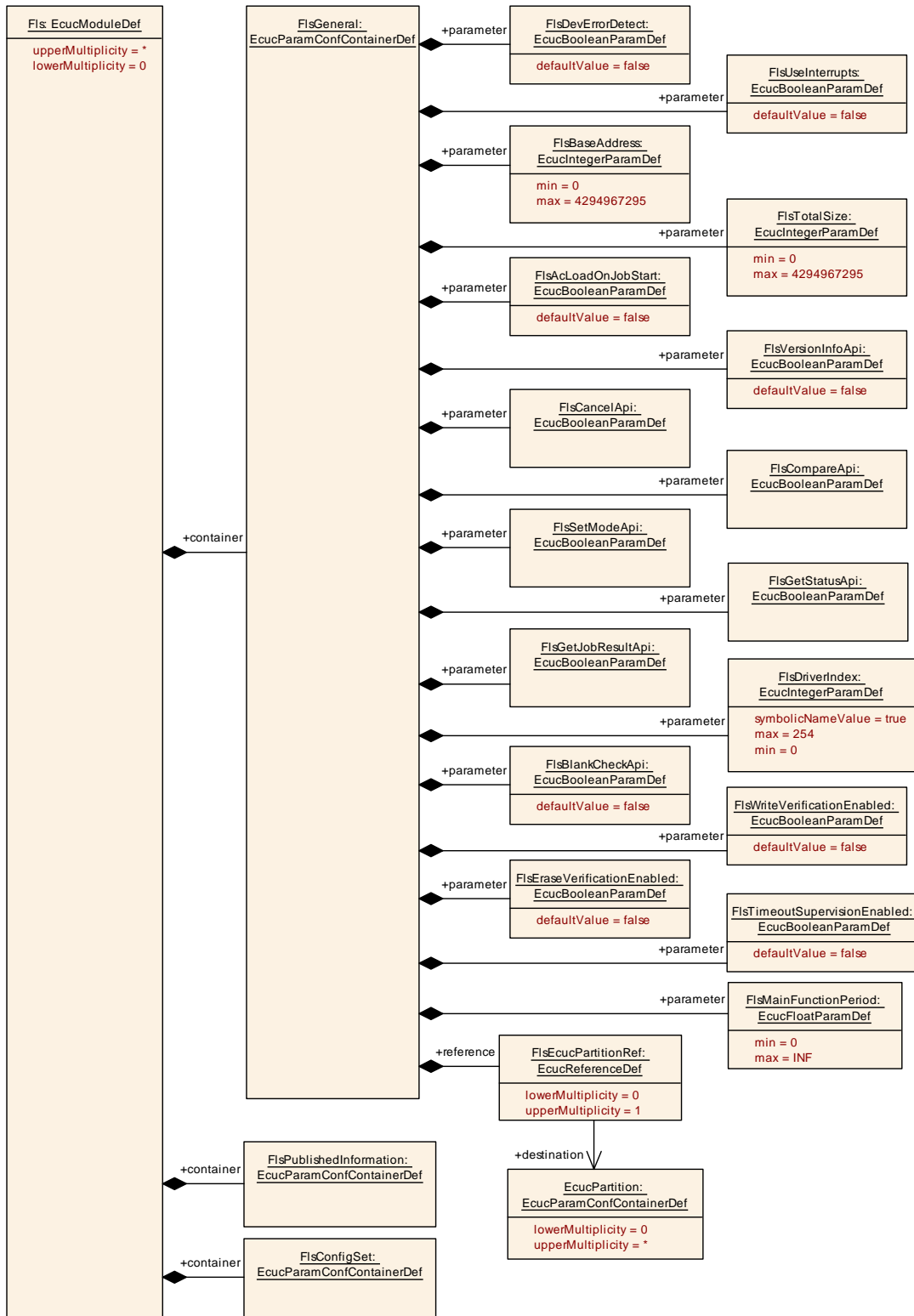
Figure 10.1: Configuration of the Fls

10.2.2 FlsGeneral

SWS Item	[ECUC_Fls_00172]
Container Name	FlsGeneral
Parent Container	Fls
Description	Container for general parameters of the flash driver. These parameters are always pre-compile.
Configuration Parameters	

SWS Item	[ECUC_Fls_00284]		
Parameter Name	FlsAcLoadOnJobStart		
Parent Container	FlsGeneral		
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		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00169]		
Parameter Name	FlsBaseAddress		
Parent Container	FlsGeneral		
Description	The flash memory start address (see also SWS_Fls_00208 and SWS_Fls_00209). This parameter defines the lower boundary for read / write / erase and compare jobs.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00319]
Parameter Name	FlsBlankCheckApi
Parent Container	FlsGeneral
Description	Compile switch to enable/disable the Fls_BlankCheck function. true: API supported / function provided. false: API not supported / function not provided
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false

▽

△

Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00285]		
Parameter Name	FlsCancelApi		
Parent Container	FlsGeneral		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00286]		
Parameter Name	FlsCompareApi		
Parent Container	FlsGeneral		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00287]		
Parameter Name	FlsDevErrorDetect		
Parent Container	FlsGeneral		
Description	Switches the development error detection and notification on or off.		
	• true: detection and notification is enabled.		
	• false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	

▽

$\triangle$

Scope / Dependency	scope: local		

SWS Item	**[ECUC_Fls_00288]**		
Parameter Name	FlsDriverIndex		
Parent Container	FlsGeneral		
Description	Index of the driver, used by FEE.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 254		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_Fls_00321]**		
Parameter Name	FlsEraseVerificationEnabled		
Parent Container	FlsGeneral		
Description	Compile switch to enable erase verification. true: memory region is checked to be erased. false: memory region is not checked to be erased.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Fls_00289]**		
Parameter Name	FlsGetJobResultApi		
Parent Container	FlsGeneral		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00290]		
Parameter Name	FlsGetStatusApi		
Parent Container	FlsGeneral		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00306]		
Parameter Name	FlsMainFunctionPeriod		
Parent Container	FlsGeneral		
Description	Cycle time of calls of the flash driver's main function (in seconds).		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: Only relevant if deadline monitoring for internal functionality has to be done in software (e.g. erase / write timings)		

SWS Item	[ECUC_Fls_00291]		
Parameter Name	FlsSetModeApi		
Parent Container	FlsGeneral		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00322]
Parameter Name	FlsTimeoutSupervisionEnabled
Parent Container	FlsGeneral

▽

△

Description	Compile switch to enable timeout supervision.
	true: timeout supervision for read/erase/write/compare jobs enabled. false: timeout supervision for read/erase/write/compare jobs disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Post-Build Variant Value	false
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local

SWS Item	[ECUC_Fls_00170]
Parameter Name	FlsTotalSize
Parent Container	FlsGeneral
Description	The total amount of flash memory in bytes (see also SWS_Fls_00208 and SWS_Fls_00209).
	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	–
Post-Build Variant Value	false
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local

SWS Item	[ECUC_Fls_00292]
Parameter Name	FlsUseInterrupts
Parent Container	FlsGeneral
Description	Job processing triggered by hardware interrupt. true: Job processing triggered by interrupt (hardware controlled). false: Job processing not triggered by interrupt (software controlled) or the underlying hardware does not support interrupt mode for flash operations.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Post-Build Variant Value	false
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local
	dependency: Only available if supported by underlying flash hardware

SWS Item	[ECUC_Fls_00293]		
Parameter Name	FlsVersionInfoApi		
Parent Container	FlsGeneral		
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	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00320]		
Parameter Name	FlsWriteVerificationEnabled		
Parent Container	FlsGeneral		
Description	Compile switch to enable write verification. true: written data is compared directly after write. false: written date is not compared directly after write.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00323]		
Parameter Name	FlsEcucPartitionRef		
Parent Container	FlsGeneral		
Description	Maps the Flash driver to zero or one ECUC partition to make the driver API available in this partition.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.3 FlsConfigSet

SWS Item	[ECUC_Fls_00174]
Container Name	FlsConfigSet
Parent Container	Fls
Description	Container for runtime configuration parameters of the flash driver. Implementation Type: Fls_ConfigType.
Configuration Parameters	

SWS Item	[ECUC_Fls_00270]		
Parameter Name	FlsAcErase		
Parent Container	FlsConfigSet		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00305]		
Parameter Name	FlsAcWrite		
Parent Container	FlsConfigSet		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00318]	
Parameter Name	FlsDefaultMode	
Parent Container	FlsConfigSet	
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 value	MEMIF_MODE_SLOW		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Fls_00307]**		
Parameter Name	FlsJobEndNotification		
Parent Container	FlsConfigSet		
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	–		
Regular Expression	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Fls_00274]**		
Parameter Name	FlsJobErrorNotification		
Parent Container	FlsConfigSet		
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	–		
Regular Expression	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00275]		
Parameter Name	FlsMaxReadFastMode		
Parent Container	FlsConfigSet		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		
	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	[ECUC_Fls_00276]		
Parameter Name	FlsMaxReadNormalMode		
Parent Container	FlsConfigSet		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		
	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	[ECUC_Fls_00277]		
Parameter Name	FlsMaxWriteFastMode		
Parent Container	FlsConfigSet		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD

▽

△

Scope / Dependency	scope: local
	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	[ECUC_Fls_00278]		
Parameter Name	FlsMaxWriteNormalMode		
Parent Container	FlsConfigSet		
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	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		
	dependency: 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	[ECUC_Fls_00279]		
Parameter Name	FlsProtection		
Parent Container	FlsConfigSet		
Description	Erase/write protection settings. Only relevant if supported by hardware.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		
	dependency: Only relevant if supported by hardware.		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
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.

[SWS_Fls_00352] ⌈The table above specifies the parameters that shall be located in an external data structure of type `Fls_ConfigType`.⌋*(SRS_BSW_00438, SRS_BSW_00388)*

[SWS_Fls_00353] ⌈The organization and location of the data structure `Fls_Config-Type` shall be up to the implementer.⌋ *(SRS_BSW_00438)*

[SWS_Fls_00355] ⌈Hardware or implementation specific parameters can be added to `Fls_ConfigType` if necessary.⌋ *(SRS_BSW_00438)*
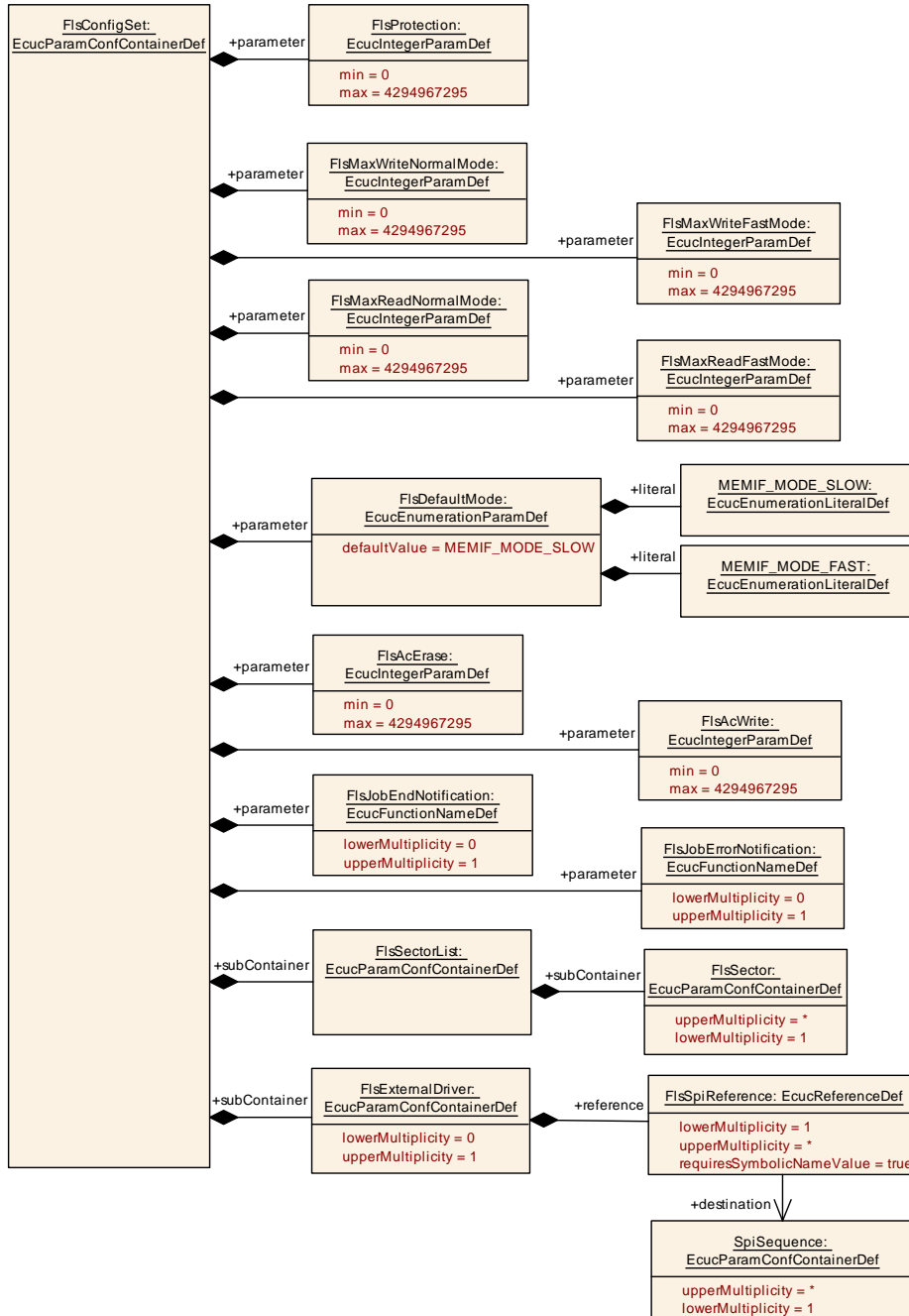

Figure 10.2: Runtime Configuration Parameters

10.2.4 FlsExternalDriver

SWS Item	[ECUC_Fls_00316]
Container Name	FlsExternalDriver
Parent Container	FlsConfigSet
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	[ECUC_Fls_00317]		
Parameter Name	FlsSpiReference		
Parent Container	FlsExternalDriver		
Description	Reference to SPI sequence (required for external Flash drivers).		
Multiplicity	1..*		
Type	Symbolic name reference to SpiSequence		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.5 FlsSectorList

SWS Item	[ECUC_Fls_00201]
Container Name	FlsSectorList
Parent Container	FlsConfigSet
Description	List of flashable sectors and pages.
Configuration Parameters	

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

10.2.6 FlsSector

SWS Item	[ECUC_Fls_00202]
Container Name	FlsSector
Parent Container	FlsSectorList
Description	Configuration description of a flashable sector
Configuration Parameters	

SWS Item	[ECUC_Fls_00280]		
Parameter Name	FlsNumberOfSectors		
Parent Container	FlsSector		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00281]		
Parameter Name	FlsPageSize		
Parent Container	FlsSector		
Description	Size of one page of this sector. Implementation Type: Fls_LengthType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: The sector size has to be an integer multiple of the page size.		

SWS Item	[ECUC_Fls_00282]
Parameter Name	FlsSectorSize
Parent Container	FlsSector
Description	Size of this sector. Implementation Type: Fls_LengthType.
Multiplicity	1
Type	EcucIntegerParamDef
Range	0 .. 4294967295

▽

△

Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		
	dependency: The sector size has to be an integer multiple of the page size.		

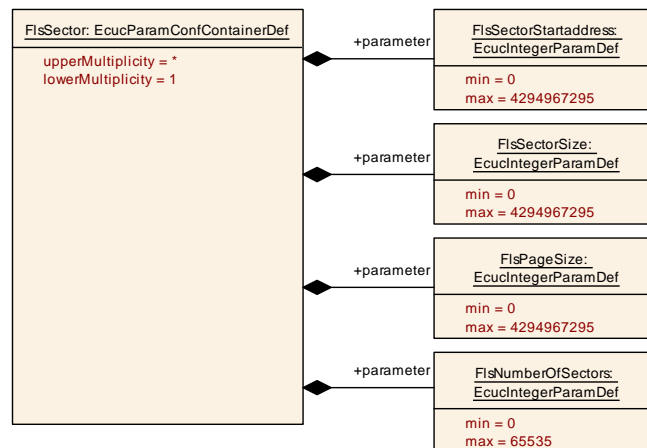| SWS Item	[ECUC_Fls_00283]		
Parameter Name	FlsSectorStartaddress		
Parent Container	FlsSector		
Description	Start address of this sector.		
	Implementation Type: Fls_AddressType.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

Figure 10.3: Sector Parameters

10.3 Published Information

For details refer to the chapter 10.3 "Published Information" in SWS_BSWGeneral [4].

SWS Item	[ECUC_Fls_00178]
Container Name	FlsPublishedInformation
Parent Container	Fls
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	[ECUC_Fls_00294]		
Parameter Name	FlsAcLocationErase		
Parent Container	FlsPublishedInformation		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00295]		
Parameter Name	FlsAcLocationWrite		
Parent Container	FlsPublishedInformation		
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	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00296]		
Parameter Name	FlsAcSizeErase		
Parent Container	FlsPublishedInformation		
Description	Number of bytes in RAM needed for the erase flash access code.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00297]		
Parameter Name	FlsAcSizeWrite		
Parent Container	FlsPublishedInformation		
Description	Number of bytes in RAM needed for the write flash access code.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00299]		
Parameter Name	FlsErasedValue		
Parent Container	FlsPublishedInformation		
Description	The contents of an erased flash memory cell.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00298]		
Parameter Name	FlsEraseTime		
Parent Container	FlsPublishedInformation		
Description	Maximum time to erase one complete flash sector.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	[ECUC_Fls_00300]		
Parameter Name	FlsExpectedHwId		
Parent Container	FlsPublishedInformation		
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	–		
Regular Expression	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Fls_00198]**		
Parameter Name	FlsSpecifiedEraseCycles		
Parent Container	FlsPublishedInformation		
Description	Number of erase cycles specified for the flash device (usually given in the device data sheet). 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	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

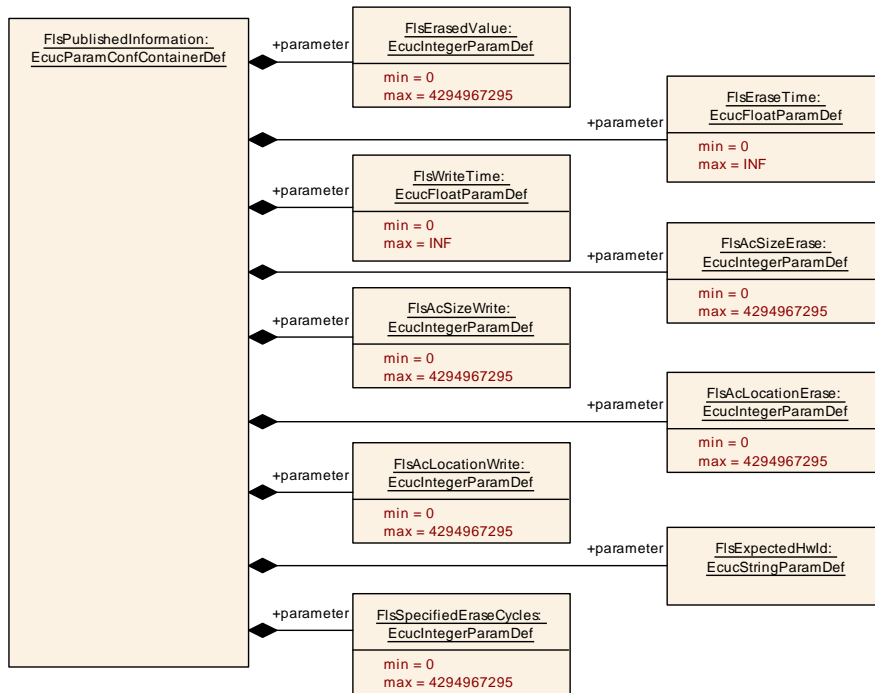| SWS Item	**[ECUC_Fls_00301]**		
Parameter Name	FlsWriteTime		
Parent Container	FlsPublishedInformation		
Description	Maximum time to program one complete flash page.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	Published Information	X	All Variants
Scope / Dependency	scope: local		

No Included Containers

Figure 10.4: Additional Published Parameters

A Not applicable requirements

[SWS_Fls_NA_00366] ⌈These requirements are not applicable to this specification.⌋ *(SRS_BSW_00344, SRS_BSW_00170, SRS_BSW_00398, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_-BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_-BSW_00433, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00422, SRS_-BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_-BSW_00415, SRS_BSW_00342, SRS_BSW_00160, SRS_BSW_00007, SRS_-BSW_00300, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00314, SRS_-BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00302, SRS_-BSW_00328, SRS_BSW_00312, SRS_BSW_00006, SRS_BSW_00304, SRS_-BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_-BSW_00371, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00330, SRS_-BSW_00009, SRS_BSW_00401, SRS_BSW_00172, SRS_BSW_00010, SRS_-BSW_00333, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334, SRS_-SPAL_12267, SRS_SPAL_12163, SRS_SPAL_12462, SRS_SPAL_12463, SRS_-SPAL_12069, SRS_SPAL_12063, SRS_SPAL_12064, SRS_SPAL_12067, SRS_-SPAL_12078, SRS_Fls_12149)*