<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.06.2006</td>
<td>2.0.1</td>
<td>AUTOSAR Administration</td>
<td>Layout Adaptations</td>
</tr>
<tr>
<td>10.04.2006</td>
<td>2.0.0</td>
<td>AUTOSAR Administration</td>
<td>Document structure adapted to common Release 2.0 SWS Template</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- new functionality: Read, Compare and SetMode functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- scalability: functionality can be configured (on/off)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- adapted to new MemHwA architecture</td>
</tr>
<tr>
<td>10.07.2004</td>
<td>1.0.0</td>
<td>AUTOSAR Administration</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
Disclaimer

This specification as released by the AUTOSAR Development Partnership is intended for the purpose of information only. The use of material contained in this specification requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of this Specification.

Following the completion of the development of the AUTOSAR Specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2006 AUTOSAR Development Partnership. All rights reserved.

Advice to users of AUTOSAR Specification Documents:

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

1. Introduction and functional overview ................................................................. 5
2. Acronyms and abbreviations .............................................................................. 6
3. Related documentation ...................................................................................... 7
   3.1 AUTOSAR deliverables .................................................................................. 7
   3.2 Related standards and norms ......................................................................... 7
4. Constraints and assumptions ............................................................................. 8
   4.1 Limitations .................................................................................................... 8
   4.2 Applicability to car domains ......................................................................... 8
5. Dependencies to other modules ......................................................................... 9
   5.1 File structure ................................................................................................ 9
      5.1.1 Code file structure ................................................................................ 9
      5.1.2 Header file structure ............................................................................. 9
   5.2 System clock ............................................................................................... 10
   5.3 Communication or I/O drivers .................................................................. 10
6. Requirements traceability ................................................................................. 11
7. Functional specification .................................................................................... 18
   7.1 General design rules .................................................................................. 18
   7.2 Error classification .................................................................................... 18
   7.3 Error detection .......................................................................................... 19
   7.4 Error notification ...................................................................................... 19
   7.5 External flash driver .................................................................................. 20
   7.6 Loading, executing and removing the flash access code ......................... 20
8. API specification .............................................................................................. 21
   8.1 Imported types ............................................................................................ 21
      8.1.1 Standard types .................................................................................... 21
      8.1.2 MemIf types ........................................................................................ 21
   8.2 Type definitions .......................................................................................... 21
      8.2.1 Fls_ConfigType ................................................................................... 21
   8.3 Function definitions .................................................................................... 21
      8.3.1 Fls_Init ............................................................................................... 21
      8.3.2 Fls_Erase ............................................................................................ 22
      8.3.3 Fls_Write .............................................................................................. 23
      8.3.4 Fls_Cancel .......................................................................................... 25
      8.3.5 Fls_GetStatus ..................................................................................... 25
      8.3.6 Fls_GetJobResult ................................................................................. 26
      8.3.7 Fls_MainFunction ............................................................................... 26
      8.3.8 Fls_Read ............................................................................................. 28
      8.3.9 Fls_Compare ....................................................................................... 29
      8.3.10 Fls_SetMode ..................................................................................... 30
      8.3.11 Fls_GetVersionInfo ......................................................................... 30
   8.4 Call-back notifications ................................................................................. 31
1 Introduction and functional overview

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Abbreviation / Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DET</td>
<td>Development Error Tracer – module to which development errors are reported.</td>
</tr>
<tr>
<td>DEM</td>
<td>Diagnostic Event Manager – module to which production relevant errors are reported.</td>
</tr>
<tr>
<td>AC</td>
<td>(Flash) access code – abbreviation introduced to keep the names of the configuration parameters reasonably short.</td>
</tr>
</tbody>
</table>

Further definitions of terms used throughout this document:

<table>
<thead>
<tr>
<th>Term:</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash sector</td>
<td>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.</td>
</tr>
<tr>
<td>Flash page</td>
<td>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.</td>
</tr>
<tr>
<td>Flash access code</td>
<td>Internal flash driver routines called by the main function (job processing function) to erase or write the flash hardware.</td>
</tr>
</tbody>
</table>
3 Related documentation

3.1 AUTOSAR deliverables

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

[2] Layered Software Architecture, 
AUTOSAR_LayeredSoftwareArchitecture.pdf

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

[4] General Requirements on SPAL, 
AUTOSAR_SRS_SPAL_General.pdf

[5] Requirements on Flash Driver 
AUTOSAR_SRS_Flash_Driver.pdf

[6] Requirements on Memory Hardware Abstraction Layer, 
AUTOSAR_SRS_MemHW_AbstractionLayer.pdf

AUTOSAR_ECU_Configuration.pdf

3.2 Related standards and norms

[8] HIS Flash Driver Specification 
HIS flash driver v130.pdf on 
http://www.automotive-his.de/download/
4 Constraints and assumptions

4.1 Limitations

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

4.2 Applicability to car domains

No restrictions.
5 Dependencies to other modules

5.1 File structure

5.1.1 Code file structure

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

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

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

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

5.1.2 Header file structure

**FLS107:** The file include structure shall be as follows:

![File Include Structure Diagram](image)

*Figure 1: File include structure*
- Fls.h shall include MemIf_Types.h, Dem.h and Det.h
- Fls.h shall include Fls_Cfg.h
- Fls.c shall include Fls.h
- Fls_PBcfg.c shall include Fls.h
- In case of a driver for an external SPI flash memory, Fls.h shall include Spi.h

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

### 5.2 System clock

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

### 5.3 Communication or I/O drivers

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

**Document:** General Requirements on Basic Software Modules

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Satisfied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BSW00344] Reference to link-time configuration</td>
<td>Not applicable (this module does not provide any link-time parameters)</td>
</tr>
<tr>
<td>[BSW00404] Reference to post build time configuration</td>
<td>FLS014, FLS173, FLS174</td>
</tr>
<tr>
<td>[BSW00405] Reference to multiple configuration sets</td>
<td>FLS014, FLS173, FLS174</td>
</tr>
<tr>
<td>[BSW00345] Pre-compile time configuration</td>
<td>FLS171, FLS172</td>
</tr>
<tr>
<td>[BSW159] Tool-based configuration</td>
<td>FLS179</td>
</tr>
<tr>
<td>[BSW167] Static configuration checking</td>
<td>FLS053</td>
</tr>
<tr>
<td>[BSW171] Configurability of optional functionality</td>
<td>FLS172, FLS183, FLS184, FLS185, FLS186, FLS187, FLS188</td>
</tr>
<tr>
<td>[BSW170] Data for reconfiguration of AUTOSAR SW-components</td>
<td>Not applicable (this module does not depend on faults, signal qualities, …)</td>
</tr>
<tr>
<td>[BSW00380] Separate C-File for configuration parameters</td>
<td>FLS159, FLS179</td>
</tr>
<tr>
<td>[BSW00419] Separate C-Files for pre-compile time configuration parameters</td>
<td>FLS179</td>
</tr>
<tr>
<td>[BSW00381] Separate configuration header file for pre-compile time parameters</td>
<td>FLS107</td>
</tr>
<tr>
<td>[BSW00412] Separate H-File for configuration parameters</td>
<td>FLS107</td>
</tr>
<tr>
<td>[BSW00383] List dependencies of configuration files</td>
<td>FLS189</td>
</tr>
<tr>
<td>[BSW00384] List dependencies to other modules</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>[BSW00387] Specify the configuration class of callback function</td>
<td>Not applicable (this module does not provide any callback routines)</td>
</tr>
<tr>
<td>[BSW00388] Introduce containers</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00389] Containers shall have names</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00390] Parameter content shall be unique within the module</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00391] Parameter shall have unique names</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00392] Parameters shall have a type</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00393] Parameters shall have a range</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00394] Specify the scope of the parameters</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00395] List the required parameters (per parameter)</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00396] Configuration classes</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00397] Pre-compile-time parameters</td>
<td>Chapter 10.2.2, Chapter 10.2.3</td>
</tr>
<tr>
<td>[BSW00398] Link-time parameters</td>
<td>Not applicable (this module does not provide any link-time parameters)</td>
</tr>
<tr>
<td>[BSW00399] Loadable Post-build time parameters</td>
<td>Chapter 10.2.3</td>
</tr>
<tr>
<td>[BSW00400] Selectable Post-build time parameters</td>
<td>Chapter 10.2.3</td>
</tr>
<tr>
<td>[BSW00402] Published information</td>
<td>Chapter 10.3</td>
</tr>
<tr>
<td>[BSW00375] Notification of wake-up reason</td>
<td>Not applicable (this module does not wake up the ECU)</td>
</tr>
<tr>
<td>[BSW101] Initialization interface</td>
<td>FLS014</td>
</tr>
<tr>
<td>Requirement</td>
<td>Satisfied by</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[BSW00416] Sequence of Initialization</td>
<td>Not applicable (requirement on system architecture, not on a single module)</td>
</tr>
<tr>
<td>[BSW00406] Check module initialization</td>
<td>FLS017</td>
</tr>
<tr>
<td>[BSW168] Diagnostic Interface of SW components</td>
<td>Not applicable (no use case)</td>
</tr>
<tr>
<td>[BSW00407] Function to read out published parameters</td>
<td>Chapter 8.3.11</td>
</tr>
<tr>
<td>[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces</td>
<td>Not applicable (this module does not provide an AUTOSAR interface)</td>
</tr>
<tr>
<td>[BSW00424] BSW main processing function task allocation</td>
<td>Not applicable (requirement on system design, not on a single module)</td>
</tr>
<tr>
<td>[BSW00425] Trigger conditions for schedulable objects</td>
<td>Chapter 8.5</td>
</tr>
<tr>
<td>[BSW00426] Exclusive areas in BSW modules</td>
<td>Not applicable (this module does not provide any exclusive areas)</td>
</tr>
<tr>
<td>[BSW00427] ISR description for BSW modules</td>
<td>Not applicable (no ISR’s defined for this module, usage of interrupts is implementation specific)</td>
</tr>
<tr>
<td>[BSW00428] Execution order dependencies of main processing functions</td>
<td>Not applicable (this module does provide only one main processing function)</td>
</tr>
<tr>
<td>[BSW00429] Restricted BSW OS functionality access</td>
<td>Not applicable (requirement on the implementation, not for the specification)</td>
</tr>
<tr>
<td>[BSW00431] The BSW Scheduler module implements task bodies</td>
<td>Not applicable (requirement on the BSW scheduler module)</td>
</tr>
<tr>
<td>[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path</td>
<td>FLS190</td>
</tr>
<tr>
<td>[BSW00433] Calling of main processing functions</td>
<td>Not applicable (requirement on system design, not on a single module)</td>
</tr>
<tr>
<td>[BSW00434] The Schedule Module shall provide an API for exclusive areas</td>
<td>Not applicable (this module does not provide any exclusive areas)</td>
</tr>
<tr>
<td>[BSW00336] Shutdown interface</td>
<td>Not applicable (no use case).</td>
</tr>
<tr>
<td>[BSW00337] Classification of errors</td>
<td>FLS004, FLS007</td>
</tr>
<tr>
<td>[BSW00338] Detection and Reporting of development errors</td>
<td>FLS077</td>
</tr>
<tr>
<td>[BSW00369] Do not return development error codes via API</td>
<td>FLS164, FLS006</td>
</tr>
<tr>
<td>[BSW00339] Reporting of production relevant error status</td>
<td>Not applicable (this module only provides production relevant error events, no error status)</td>
</tr>
<tr>
<td>[BSW00421] Reporting of production relevant error events</td>
<td>FLS006, FLS104, FLS105, FLS106, FLS154</td>
</tr>
<tr>
<td>[BSW00422] Debouncing of production relevant error status</td>
<td>Not applicable (requirement on the DEM)</td>
</tr>
<tr>
<td>[BSW00420] Production relevant error event rate detection</td>
<td>Not applicable (requirement on the DEM)</td>
</tr>
<tr>
<td>[BSW00417] Reporting of Error Events by Non-Basic Software</td>
<td>Not applicable (this is a BSW module)</td>
</tr>
<tr>
<td>Requirement</td>
<td>Satisfied by</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>[BSW00323] API parameter checking</td>
<td>FLS015, FLS020, FLS021, FLS026, FLS027, FLS097, FLS098</td>
</tr>
<tr>
<td>[BSW004] Version check</td>
<td>FLS053</td>
</tr>
<tr>
<td>[BSW00409] Header files for production code</td>
<td>FLS160, FLS107</td>
</tr>
<tr>
<td>error IDs</td>
<td></td>
</tr>
<tr>
<td>[BSW00385] List possible error notifications</td>
<td>FLS077, FLS167, FLS163, FLS172</td>
</tr>
<tr>
<td>[BSW161] Microcontroller abstraction</td>
<td>Not applicable (requirement on AUTOSAR architecture, not a single module)</td>
</tr>
<tr>
<td>[BSW162] ECU layout abstraction</td>
<td>Not applicable (requirement on AUTOSAR architecture, not a single module)</td>
</tr>
<tr>
<td>[BSW00324] Do not use HIS I/O Library</td>
<td>Not applicable (architecture decision)</td>
</tr>
<tr>
<td>[BSW005] No hard coded horizontal interfaces within MCAL</td>
<td>Not applicable (requirement on AUTOSAR architecture, not a single module)</td>
</tr>
<tr>
<td>[BSW00415] User dependent include files</td>
<td>Not applicable (only one user for this module)</td>
</tr>
<tr>
<td>[BSW164] Implementation of interrupt service routines</td>
<td>FLS193</td>
</tr>
<tr>
<td>[BSW00325] Runtime of interrupt service routines</td>
<td>FLS193</td>
</tr>
<tr>
<td>[BSW00326] Transition from ISRs to OS tasks</td>
<td>Not applicable (requirement on implementation, not on specification)</td>
</tr>
<tr>
<td>[BSW00342] Usage of source code and object code</td>
<td>Not applicable (requirement on AUTOSAR architecture, not a single module)</td>
</tr>
<tr>
<td>[BSW00343] Specification and configuration of time</td>
<td>FLS178</td>
</tr>
<tr>
<td>[BSW160] Human-readable configuration data</td>
<td>Not applicable (requirement on documentation, not on specification)</td>
</tr>
<tr>
<td>[BSW007] HIS MISRA C</td>
<td>Not applicable (requirement on implementation, not on specification)</td>
</tr>
<tr>
<td>[BSW00300] Module naming convention</td>
<td>Not applicable (requirement on implementation, not on specification)</td>
</tr>
<tr>
<td>[BSW00413] Accessing instances of BSW modules</td>
<td>Conflict: This is currently not reflected in the driver’s specification. This requirement will have impact on almost all BSW modules, therefore it can not be implemented within the Release 2.0 timeframe.</td>
</tr>
<tr>
<td>[BSW00347] Naming separation of different instances of BSW drivers</td>
<td>Not applicable (requirement on the implementation, not on the specification)</td>
</tr>
<tr>
<td>[BSW00305] Self-defined data types naming convention</td>
<td>Chapter 8.2</td>
</tr>
<tr>
<td>[BSW00307] Global variables naming convention</td>
<td>Not applicable (requirement on the implementation, not on the specification)</td>
</tr>
<tr>
<td>[BSW00310] API naming convention</td>
<td>Chapter 8.3</td>
</tr>
<tr>
<td>[BSW00373] Main processing function naming convention</td>
<td>Chapter 8.3.7</td>
</tr>
<tr>
<td>[BSW00327] Error values naming convention</td>
<td>FLS004, FLS007</td>
</tr>
<tr>
<td>Requirement</td>
<td>Satisfied by</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>[BSW00335] Status values naming convention</td>
<td>Chapter 8.1.2</td>
</tr>
<tr>
<td>[BSW00350] Development error detection keyword</td>
<td>FLS077, FLS162, FLS172</td>
</tr>
<tr>
<td>[BSW00408] Configuration parameter naming convention</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00410] Compiler switches shall have defined values</td>
<td>Chapter 10.2</td>
</tr>
<tr>
<td>[BSW00411] Get version info keyword</td>
<td>Chapter 10.2.2</td>
</tr>
<tr>
<td>[BSW00346] Basic set of module files</td>
<td>FLS107</td>
</tr>
<tr>
<td>[BSW158] Separation of configuration from implementation</td>
<td>FLS107</td>
</tr>
<tr>
<td>[BSW00314] Separation of interrupt frames and service routines</td>
<td>Not applicable (this module does not implement any ISRs)</td>
</tr>
<tr>
<td>[BSW00370] Separation of callback interface from API</td>
<td>Not applicable (this module does not provide any callback routines)</td>
</tr>
<tr>
<td>[BSW00348] Standard type header</td>
<td>Not applicable (standard header files included via interface header file)</td>
</tr>
<tr>
<td>[BSW00353] Platform specific type header</td>
<td>Not applicable (standard header files included via interface header file)</td>
</tr>
<tr>
<td>[BSW00361] Compiler specific language extension header</td>
<td>Not applicable (standard header files included via interface header file)</td>
</tr>
<tr>
<td>[BSW00301] Limit imported information</td>
<td>FLS107</td>
</tr>
<tr>
<td>[BSW00302] Limit exported information</td>
<td>Not applicable (requirement on the implementation, not on the specification)</td>
</tr>
<tr>
<td>[BSW00328] Avoid duplication of code</td>
<td>Not applicable (requirement on the implementation, not on the specification)</td>
</tr>
<tr>
<td>[BSW00312] Shared code shall be reentrant</td>
<td>Not applicable (requirement on the implementation, not on the specification)</td>
</tr>
<tr>
<td>[BSW006] Platform independency</td>
<td>Not applicable (this is a module of the microcontroller abstraction layer)</td>
</tr>
<tr>
<td>[BSW00357] Standard API return type</td>
<td>Chapter 8.3.2, Chapter 8.3.3, Chapter 8.3.8, Chapter 8.3.9</td>
</tr>
<tr>
<td>[BSW00377] Module specific API return types</td>
<td>Chapter 8.3.5, Chapter 8.3.6</td>
</tr>
<tr>
<td>[BSW00304] AUTOSAR integer data types</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00355] Do not redefine AUTOSAR integer data types</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00378] AUTOSAR boolean type</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00306] Avoid direct use of compiler and platform specific keywords</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00308] Definition of global data</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>Requirement</td>
<td>Satisfied by</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>[BSW00309] Global data with read-only constraint</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00371] Do not pass function pointers via API</td>
<td>Not applicable (no function pointers in this specification)</td>
</tr>
<tr>
<td>[BSW00358] Return type of init() functions</td>
<td>Chapter 8.3.1</td>
</tr>
<tr>
<td>[BSW00414] Parameter of init function</td>
<td>Chapter 8.3.1, FLS194</td>
</tr>
<tr>
<td>[BSW00376] Return type and parameters of main processing functions</td>
<td>Chapter 8.3.7</td>
</tr>
<tr>
<td>[BSW00359] Return type of callback functions</td>
<td>Not applicable (this module does not provide any callback routines)</td>
</tr>
<tr>
<td>[BSW00360] Parameters of callback functions</td>
<td>Not applicable (this module does not provide any callback routines)</td>
</tr>
<tr>
<td>[BSW00329] Avoidance of generic interfaces</td>
<td>Chapter 8.3  (explicit interfaces defined)</td>
</tr>
<tr>
<td>[BSW00330] Usage of macros / inline functions instead of functions</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00331] Separation of error and status values</td>
<td>FLS004, FLS164, FLS006</td>
</tr>
<tr>
<td>[BSW0009] Module User Documentation</td>
<td>Not applicable (requirement on documentation, not on specification)</td>
</tr>
<tr>
<td>[BSW00401] Documentation of multiple instances of configuration parameters</td>
<td>Not applicable (all configuration parameters are single instance only)</td>
</tr>
<tr>
<td>[BSW172] Compatibility and documentation of scheduling strategy</td>
<td>Not applicable (no internal scheduling policy)</td>
</tr>
<tr>
<td>[BSW010] Memory resource documentation</td>
<td>Not applicable (requirement on documentation, not on specification)</td>
</tr>
<tr>
<td>[BSW00333] Documentation of callback function context</td>
<td>Not applicable (requirement on documentation, not for specification)</td>
</tr>
<tr>
<td>[BSW00374] Module vendor identification</td>
<td>FLS178</td>
</tr>
<tr>
<td>[BSW00379] Module identification</td>
<td>FLS178</td>
</tr>
<tr>
<td>[BSW003] Version identification</td>
<td>FLS178</td>
</tr>
<tr>
<td>[BSW00318] Format of module version numbers</td>
<td>FLS178</td>
</tr>
<tr>
<td>[BSW00321] Enumeration of module version numbers</td>
<td>Not applicable (requirement on implementation, not for specification)</td>
</tr>
<tr>
<td>[BSW00341] Microcontroller compatibility documentation</td>
<td>Not applicable (requirement on documentation, not on specification)</td>
</tr>
<tr>
<td>[BSW00334] Provision of XML file</td>
<td>Not applicable (requirement on documentation, not on specification)</td>
</tr>
</tbody>
</table>
### Requirement Satisfied by

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

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Satisfied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BSW12132] Flash driver static</td>
<td>FLS048, FLS171</td>
</tr>
<tr>
<td>configuration</td>
<td></td>
</tr>
<tr>
<td>[BSW12133] Publication of flash</td>
<td>FLS177, FLS178</td>
</tr>
<tr>
<td>properties</td>
<td></td>
</tr>
<tr>
<td>[BSW12134] Flash read function</td>
<td>FLS095, FLS096, FLS097, FLS098</td>
</tr>
<tr>
<td>[BSW12135] Flash write function</td>
<td>FLS024, FLS025, FLS026, FLS027</td>
</tr>
<tr>
<td>[BSW12136] Flash erase function</td>
<td>FLS018, FLS019, FLS020, FLS021</td>
</tr>
<tr>
<td>[BSW123301] Flash compare function</td>
<td>FLS148, FLS149, FLS150, FLS151, FLS152, FLS153, FLS186</td>
</tr>
<tr>
<td>[BSW12137] Flash cancel function</td>
<td>FLS031, FLS183</td>
</tr>
<tr>
<td>[BSW12138] Flash driver status function</td>
<td>FLS034, FLS184</td>
</tr>
<tr>
<td>[BSW121302] Flash driver mode selection function</td>
<td>FLS155, FLS156, FLS187</td>
</tr>
<tr>
<td>[BSW12159] Flash address check</td>
<td>FLS020, FLS021, FLS026, FLS027, FLS097, FLS098</td>
</tr>
<tr>
<td>[BSW12158] Flash blank check</td>
<td>FLS055</td>
</tr>
<tr>
<td>[BSW12141] Flash write verification</td>
<td>FLS056</td>
</tr>
<tr>
<td>[BSW12160] Flash erase verification</td>
<td>FLS022</td>
</tr>
<tr>
<td>[BSW12143] Flash driver job management</td>
<td>FLS016, FLS017, FLS023, FLS030, FLS032, FLS100</td>
</tr>
<tr>
<td>[BSW12144] Flash driver job processing function</td>
<td>FLS037, FLS038, FLS039, FLS190</td>
</tr>
<tr>
<td>[BSW123303] Job processing – normal mode</td>
<td>FLS040</td>
</tr>
<tr>
<td>[BSW123304] Job processing – fast mode</td>
<td>FLS040</td>
</tr>
<tr>
<td>[BSW12193] Load flash access code to RAM on job start</td>
<td>FLS140, FLS141</td>
</tr>
<tr>
<td>[BSW12194] Execute flash access code from RAM</td>
<td>FLS142</td>
</tr>
<tr>
<td>[BSW121300] Remove flash access code from RAM</td>
<td>FLS143</td>
</tr>
<tr>
<td>[BSW12147] Functional scope</td>
<td>FLS088</td>
</tr>
<tr>
<td>[BSW12182] External flash driver static configuration</td>
<td>FLS174</td>
</tr>
<tr>
<td>[BSW12107] Check Flash type</td>
<td>FLS144</td>
</tr>
<tr>
<td>[BSW12145] Flash driver job processing execution time</td>
<td>FLS040, FLS175, FLS181, FLS176, FLS182</td>
</tr>
<tr>
<td>[BSW12083] Use HIS specification as basis</td>
<td>Not applicable (the module provides comparable functionality but different API and different design rules)</td>
</tr>
<tr>
<td>[BSW12184] Limit read access blocking times</td>
<td>FLS040</td>
</tr>
<tr>
<td>[BSW12148] Common Flash API</td>
<td>FLS088</td>
</tr>
<tr>
<td>[BSW12149] Microcontroller independency</td>
<td>Not applicable (requirement on implementation, not on specification)</td>
</tr>
</tbody>
</table>
7 Functional specification

7.1 General design rules

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

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

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

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

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

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

7.2 Error classification

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

**FLS161:** Development error values are of type `uint8`.

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

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

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

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

### 7.4 Error notification

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

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

**FLS007**: Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the flash driver’s
implementation documentation. The classification and enumeration shall be compatible with the errors listed above [FLS004].

### 7.5 External flash driver

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

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

### 7.6 Loading, executing and removing the flash access code

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

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

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

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

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

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

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

8.1 Imported types

8.1.1 Standard types

In this chapter all types included from the following files are listed:
- Std_Types.h
  - Std_ReturnType
  - Std_VersionInfoType

8.1.2 MemIf types

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

- MemIf_ModeType
- MemIf_StatusType
- MemIf_JobResultType
- MemIf_AddressType
- MemIf_LengthType

8.2 Type definitions

8.2.1 Fls_ConfigType

<table>
<thead>
<tr>
<th>Type:</th>
<th>Struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range:</td>
<td>Hardware dependend structure</td>
</tr>
<tr>
<td>Description:</td>
<td>Structure to hold the flash driver configuration set. The contents of the initialisation data structure are specific to the flash memory hardware.</td>
</tr>
<tr>
<td>Description:</td>
<td>A pointer to such a structure is provided to the flash driver initialization routine for configuration of the driver and flash memory hardware.</td>
</tr>
</tbody>
</table>

8.3 Function definitions

8.3.1 Fls_Init

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Fls_Init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void Fls_Init</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>const Fls_ConfigType *ConfigPtr</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Service ID [hex]:</td>
<td>0x00</td>
</tr>
<tr>
<td>Behaviour:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Re-entrancy</td>
<td>Non re-entrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>ConfigPtr Pointer to flash driver configuration set.</td>
</tr>
</tbody>
</table>
Parameters (out): None
Return value: None

Description:
FLS014: This function initializes the flash driver (software) and all flash memory relevant registers (hardware) with parameters provided in the given configuration set.

FLS191: This routine shall store the pointer to the given configuration set in a local variable in order to allow the module access to the configuration set contents during runtime.

FLS086: The routine shall initialize all module global variables and those controller registers that are needed for controlling the flash device and that do not influence or depend on other (hardware) modules. Registers that can influence or depend on other modules shall be initialized by a common system module.

FLS015: If development error detection is enabled the (hardware specific) contents of the given configuration set shall be checked for being within the allowed range. Errors shall be reported to the Debug Error Tracer (DET) with the error value FLS_E_PARAM_CONFIG.

FLS016: After having finished the module initialization the flash driver state shall be set to MEMIF_IDLE and the flash job result shall be set to MEMIF_JOB_OK.

FLS017: If development error detection is enabled the routine shall check that the flash driver and hardware are not yet initialized (driver state MEMIF_UNINIT) or the driver is currently busy (MEMIF_BUSY). Errors shall be reported to the Development Error Tracer (DET) with the corresponding error value (FLS_E_UNINIT respectively FLS_E_BUSY).

FLS048: If supported by hardware, this routine sets the flash memory erase/write protection as provided in the configuration set.

Caveats: None
Configuration: None

8.3.2 Fls_Erase

Service name: Fls_Erase
Syntax: Std_ReturnType Fls_Erase
          (MemIf_AddressType TargetAddress,
           MemIf_LengthType Length)

Service ID [hex]: 0x01
Behaviour: Asynchronous
Re-entrancy: Non re-entrant

Parameters (in):
TargetAddress Target address in flash memory. This address offset will be added to the flash memory base address.
Min.: 0
Max.: FLS_SIZE – 1

Length Number of bytes to erase
Min.: 1
Max.: FLS_SIZE – TargetAddress

Parameters (out): None
Return value: E_OK: erase command has been accepted
              E_NOT_OK: erase command has not been accepted
**Description:**

| FL0S18: | Service for erasing one or more complete flash sectors. This service shall copy the given parameters to driver internal variables, initiate an erase job, set the driver status to MEMIF_E_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK. |
| FL0S19: | The erase job shall be executed asynchronously within the flash driver’s main function. A flash memory block starting from FLS_BASE_ADDRESS + TargetAddress of size Length shall be erased. Note: Length will be rounded up to the next full sector boundary, since only complete flash sectors can be erased. |
| FL0S20: | If development error detection is enabled the routine shall check that the erase start address (FLS_BASE_ADDRESS + TargetAddress) is aligned to a flash sector boundary and that it lies within the specified lower and upper flash address boundaries. If not, the erase job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_ADDRESS. |
| FL0S21: | If development error detection is enabled the routine shall check that the erase length is greater than 0 and that the erase end address (erase start address + length) is aligned to a flash sector boundary and that it lies within the specified upper flash address boundary. If not, the erase job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_LENGTH. |
| FL0S65: | If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the erase request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_UNINIT. |
| FL0S23: | If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the erase request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_BUSY. |
| FL0S145: | If possible, e.g. with interrupt controlled implementations, the first round of the erase job shall be started directly form this routine to reduce overall runtime. |

**Caveats:**

- The flash driver shall have been initialized before this service is called.
- Only one read, write, erase or compare job can be accepted at a time.

**Configuration:**

None

### 8.3.3 Fls_Write

| **Service name:** | Fls_Write |
| **Syntax:** | Std_ReturnType Fls_Write  
  (  
    MemIf_AddressType TargetAddress,  
    const uint8 *SourceAddressPtr,  
    MemIf_LengthType Length  
  ) |
| **Service ID [hex]:** | 0x02 |
| **Behaviour:** | Asynchronous |
| **Re-entrancy:** | Non re-entrant |
The Specification of Flash Driver

**Parameters (in):**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetAddress</td>
<td>Target address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: $\text{FLS_SIZE} - 1$</td>
</tr>
<tr>
<td>SourceAddressPtr</td>
<td>Pointer to source data buffer</td>
</tr>
<tr>
<td>Length</td>
<td>Number of bytes to write Min.: 1 Max.: $\text{FLS_SIZE} - \text{TargetAddress}$</td>
</tr>
</tbody>
</table>

**Parameters (out):**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>write command has been accepted</td>
</tr>
<tr>
<td>E_NOT_OK</td>
<td>write command has not been accepted</td>
</tr>
</tbody>
</table>

**Description:**

**FLS024:** Service for writing one or more complete flash pages. This service shall copy the given parameters to driver internal variables, initiate a write job, set the driver status to MEMIF_E_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.

**FLS025:** The write job shall be executed asynchronously within the flash driver’s main function. A flash memory block starting from $\text{FLS\_BASE\_ADDRESS} + \text{TargetAddress}$ of size Length shall be programmed with the data provided via SourceAddressPtr.

**FLS026:** If development error detection is enabled the routine shall check that the write start address ($\text{FLS\_BASE\_ADDRESS} + \text{TargetAddress}$) is aligned to a flash page boundary and that it lies within the specified lower and upper flash address boundaries. If not, the write job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_ADDRESS.

**FLS027:** If development error detection is enabled the routine shall check that the write length is greater than 0 and that the write end address (write start address + length) is aligned to a flash page boundary and that it lies within the specified upper flash address boundary. If not, the write job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_LENGTH.

**FLS066:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the write request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_UNINIT.

**FLS030:** If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the write request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_BUSY.

**FLS157:** If development error detection is enabled, the routine shall check the given data buffer pointer for not being a NULL pointer. If this error is encountered the write request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_DATA.

**FLS146:** If possible, e.g. with interrupt controlled implementations, the first round of the write job shall be started directly form this routine to reduce overall runtime.

**Caveats:**

- The flash driver shall have been initialized before this service is called.
- Only one read, write, erase or compare job can be accepted at a time.
- The flash memory area shall have been erased before.

**Configuration:**

None
### 8.3.4 Fls_Cancel

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

### 8.3.5 Fls_GetStatus

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

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

8.3.7 Fls_MainFunction

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Fls_MainFunction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void Fls_MainFunction (void)</td>
</tr>
<tr>
<td>Service ID [hex]:</td>
<td>0x06</td>
</tr>
<tr>
<td>Behaviour:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Re-entrancy:</td>
<td>Non re-entrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>None --</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None --</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>FLS037: This function shall perform the processing of the flash read, write, erase and compare jobs. FLS038: When a job has been initiated, this function has to be called cyclically until the job is finished. FLS039: The function might also be called cyclically if no job is pending. In this case the function shall return without any action. FLS040: The flash driver main function shall only process as much data in one call cycle as statically configured for the current job type (read, write, erase or compare) and the current operating mode (normal, fas). FLS104: If a flash erase job fails due to a hardware error, the job result shall be set to MEMIF_JOB_FAILED and an error shall be reported to the DEM with the error code FLS_E_ERASE_FAILED. FLS105: If a flash write job fails due to a hardware error, the job result shall be</td>
</tr>
</tbody>
</table>
set to MEMIF_JOB_FAILED and an error shall be reported to the DEM with the error code FLS_E_WRITE_FAILED.

**FLS106:** If a flash read job fails due to a hardware error, the job result shall be set to MEMIF_JOB_FAILED and an error shall be reported to the DEM with the error code FLS_E_READ_FAILED.

**FLS154:** If a flash compare job fails due to a hardware error, the job result shall be set to MEMIF_JOB_FAILED and an error shall be reported to the DEM with the error code FLS_E_COMPARE_FAILED.

**FLS022:** After erasing a flash block, if development error detection is enabled, the routine shall compare the contents of the addressed memory area against the value of an erased flash cell to check whether the block has been completely erased. If not, the job result shall be set to MEMIF_JOB_FAILED and an error shall be reported to the DET with the error code FLS_E_VERIFY_ERASE_FAILED.

**FLS055:** Before writing a flash block, if development error detection is enabled, the routine shall compare the contents of the addressed memory area against the value of an erased flash cell to check whether the block has been completely erased. If not, the write job shall be aborted, the job result shall be set to MEMIF_JOB_FAILED and the error shall be reported to the DET with the error code FLS_E_VERIFY_ERASE_FAILED.

**FLS056:** After writing a flash block, if development error detection is enabled, the routine shall compare the contents of the reprogrammed memory area against the contents of the provided application buffer to check whether the block has been completely reprogrammed. If not, the job result shall be set to MEMIF_JOB_FAILED and an error shall be reported to the DET with the error code FLS_E_VERIFY_WRITE_FAILED.

**FLS052:** After a read, erase, write or compare job has been finished, the function shall set the job result to MEMIF_JOB_OK if it is currently MEMIF_JOB_PENDING else it shall leave the result unchanged. Also it shall set the driver state to MEMIF_IDLE and call the job end notification function if configured (FLS103).

**FLS072:** The configuration parameter FLS_USE_INTERRUPTS shall switch between interrupt and polling controlled job processing if this is supported by the flash memory hardware. If interrupt controlled job processing is supported and enabled, the interrupt service routine located in Fls_Irq.c shall reset the interrupt flag, check for errors reported by the underlying hardware, reload the hardware finite state machine for the next round of the pending job or call the appropriate notification routine if the job is finished or aborted. The function Fls_MainFunction is still required for processing of jobs without hardware interrupt support (e.g. read jobs) and for timeout supervision.

**FLS117:** If development error detection is enabled, the routine shall check if the driver has been initialized. If not an error shall be reported to the DET with the error code FLS_E_UNINIT.

<table>
<thead>
<tr>
<th>Caveats</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>None</td>
</tr>
</tbody>
</table>
### 8.3.8 Fls_Read

<table>
<thead>
<tr>
<th>Service name</th>
<th>Fls_Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>Std_ReturnType Fls_Read</code></td>
</tr>
<tr>
<td>(MemIf_AddressType SourceAddress, uint8 *TargetAddressPtr, MemIf_LengthType Length)</td>
<td></td>
</tr>
<tr>
<td>Service ID [hex]</td>
<td>0x07</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Re-entrancy</td>
<td>Non re-entrant</td>
</tr>
<tr>
<td>Parameters (in)</td>
<td>SourceAddress</td>
</tr>
<tr>
<td>Parameters (out)</td>
<td>None --</td>
</tr>
<tr>
<td>Return value</td>
<td><code>E_OK</code>: read command has been accepted <code>E_NOT_OK</code>: read command has not been accepted</td>
</tr>
</tbody>
</table>
| Description        | FLS095: Service for reading from flash memory. This service shall copy the given parameters to driver internal variables, initiate a read job, set the driver status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with `E_OK`.  
FLS096: The read job shall be executed asynchronously within the flash driver's main function. A continuous flash memory block starting from `FLS_BASE_ADDRESS` + SourceAddress of size Length shall be copied to the buffer pointed to by TargetAddressPtr.  
FLS097: If development error detection is enabled the routine shall check that the read start address (`FLS_BASE_ADDRESS` + SourceAddress) lies within the specified lower and upper flash address boundaries. If not, the read job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_ADDRESS`.  
FLS098: If development error detection is enabled the routine shall check that the read length is greater than 0 and that the read end address (read start address + length) lies within the specified upper flash address boundary. If not, the read job shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_PARAM_LENGTH`.  
FLS099: If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the read request shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_UNINIT`.  
FLS100: If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the read request shall be rejected with the return value `E_NOT_OK` and the error shall be reported to the DET with the error code `FLS_E_BUSY`.  
FLS158: If development error detection is enabled, the routine shall check the given data buffer pointer for not being a NULL pointer. If this error is encountered the write request shall be rejected with the return value `E_NOT_OK` and the error... |
Caveats:
- The flash driver shall have been initialized before this service is called.
- Only one read, write or erase job can be accepted at a time.

Configuration:
None

8.3.9 Fls_Compare

Service name: Fls_Compare

Syntax:
Std_ReturnType Fls_Compare
{
    MemIf_AddressType SourceAddress,
    uint8 *TargetAddressPtr,
    MemIf_LengthType Length
}

Service ID [hex]: 0x08

Behaviour: Asynchronous

Re-entrancy: Non re-entrant

Parameters (in):
SourceAddress Source address in flash memory. This address offset will be added to the flash memory base address.
Min.: 0
Max.: FLS_SIZE – 1

TargetAddressPtr Pointer to target data buffer

Length Number of bytes to compare
Min.: 1
Max.: FLS_SIZE – SourceAddress

Parameters (out): None

Return value:
E_OK: compare command has been accepted
E_NOT_OK: compare command has not been accepted

Description:

FLS148: Service for comparing the contents of an area of flash memory with that of an application data buffer. This service shall copy the given parameters to driver internal variables, initiate a compare job, set the driver status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.

FLS149: The compare job shall be executed asynchronously within the flash driver’s main function. A continuous flash memory block starting from FLS_BASE_ADDRESS + SourceAddress of size Length shall be compared to the buffer pointed to by TargetAddressPtr.

FLS150: If development error detection is enabled the routine shall check that the compare start address (FLS_BASE_ADDRESS + SourceAddress) lies within the specified lower and upper flash address boundaries. If not, the compare job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_ADDRESS.

FLS151: If development error detection is enabled the routine shall check that the given length is greater than 0 and that the compare end address (compare start address + length) lies within the specified upper flash address boundary. If not, the compare job shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_PARAM_LENGTH.

FLS152: If development error detection is enabled, the routine shall check if the driver has been initialized. If not, the compare request shall be rejected with the return value E_NOT_OK and the error shall be reported to the DET with the error code FLS_E_UNINIT.
8.3.10 Fls_SetMode

Service name: Fls_SetMode
Syntax: void Fls_SetMode
              (      
                MemIf_ModeType Mode
              )
Service ID [hex]: 0x09
Sync/Async: Synchronous
Reentrancy: non reentrant
Parameters (in): Mode
                MEMIF_MODE_NORMAL: Normal read access / normal SPI access.
                MEMIF_MODE_FAST: Fast read access / SPI burst access.
Parameters (out): None --
Return value: None --
Description: FLS155: This service shall set the flash driver’s operation mode to the given “Mode” parameter.

FLS156: If development error detection is enabled, the routine shall check if the driver is currently busy. If so, the mode change request shall be rejected and the error shall be reported to the DET with the error code FLS_E_BUSY.

Caveats: This service shall not be called during a running operation.
Configuration: FLS187: This function shall be configurable at pre-compile time using the parameter FLS_SET_MODE_API

8.3.11 Fls_GetVersionInfo

Service name: Fls_GetVersionInfo
Syntax: void Fls_GetVersionInfo
              {      
                Std_VersionInfoType *VersioninfoPtr
              }
Service ID [hex]: 0x10
Sync/Async: Synchronous
Re-entrancy: Non re-entrant
Parameters (in): none --
Parameters (out): Versioninfo Pointer to where to store the version information of this module.
Return value: none --
Description: FLS165: This service returns the version information of this module. The version information includes:
8.4 Call-back notifications

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

8.5 Scheduled functions

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

See chapter 8.3.7

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

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

<table>
<thead>
<tr>
<th>API function</th>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dem_ReportError</td>
<td>Dem</td>
<td>Routine to report production relevant error events by event ID.</td>
</tr>
</tbody>
</table>
8.6.2 Optional Interfaces

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

<table>
<thead>
<tr>
<th>API function</th>
<th>Module</th>
<th>Description</th>
<th>Configuration parameter (description see chapter 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det_ReportError</td>
<td>Det</td>
<td>Development error notification</td>
<td>FLS_DEV_ERROR_DETECT</td>
</tr>
</tbody>
</table>

8.6.3 Configurable interfaces

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

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

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

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

**Fee_JobEndNotification**

<table>
<thead>
<tr>
<th>Name:</th>
<th>Fee_JobEndNotification</th>
</tr>
</thead>
</table>
| Syntax:           | Void Fee_JobEndNotification  
|                   | { void } |
| Reentrancy:       | Don’t care             |
| Parameters (in):  | None                   |
| Parameters (out): | None                   |
| Return value:     | None                   |
| Description:      | FLS167: This callback function shall be called when a job has been completed with a positive result:  
|                   | - Read job finished & OK  
|                   | - Write job finished & OK  
|                   | - Erase job finished & OK  
|                   | - Compare job finished & memory blocks are the same |
| Caveats:          | None                   |
| Configuration:    | None                   |

**Fee_JobErrorNotification**

<table>
<thead>
<tr>
<th>Name:</th>
<th>Fee_JobErrorNotification</th>
</tr>
</thead>
</table>
| Syntax:           | Void Fee_JobErrorNotification  
|                   | { void } |
| Reentrancy:       | Don’t care             |
| Parameters (in):  | None                   |
| Parameters (out): | None                   |
| Return value:     | None                   |


<table>
<thead>
<tr>
<th>Return value</th>
<th>None --</th>
</tr>
</thead>
</table>
| Description: | **FLS168:** This callback function shall be called when a job has been cancelled or finished with negative result:  
|              | • Read job aborted or failed  
|              | • Write job aborted or failed  
|              | • Erase job aborted or failed  
|              | • Compare job aborted or failed  
|              | • Compare job finished and memory blocks differ |
| Caveats:     | None |
| Configuration: | None |
9 Sequence diagrams

9.1 Initialization

![Figure 2: Flash driver initialization sequence](image)

9.2 Synchronous functions

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

![Figure 3: Fls_GetJobResult](image)
9.3 Asynchronous functions

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

![Sequence diagram](image)

Figure 4: Flash write sequence, flash access code loaded on job start
9.4 Canceling a running job

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

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

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

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

Chapter 10.3 specifies published information of the module Flash Driver.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:
- AUTOSAR Layered Software Architecture [2]

This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

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

10.1.1 Configuration and configuration parameters

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

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

10.1.2 Containers

Containers structure the set of configuration parameters. This means:
- all configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.
10.1.3 Specification template for configuration parameters

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

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

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The configuration parameter shall be of configuration class Pre-compile time.</td>
</tr>
<tr>
<td>--</td>
<td>The configuration parameter shall never be of configuration class Pre-compile time.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The configuration parameter shall be of configuration class Link time.</td>
</tr>
<tr>
<td>--</td>
<td>The configuration parameter shall never be of configuration class Link time.</td>
</tr>
</tbody>
</table>

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

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

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in Chapters 10.2.2 and 10.2.3

10.2.1 Variants

Variant PC: Only pre-compile time parameters

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

FLS194: The initialization function of this module shall always have a pointer as a parameter, even though for Variant PC no configuration set shall be given. Instead a NULL pointer shall be passed to the initialization function. This means that, in
contradiction to BSW00414 only one interface for initialization shall be implemented and it shall not depend on the modules configuration, which interface the calling software module shall use.

10.2.2 Fls_ModuleConfiguration

FLS171 The following table specifies parameters that shall be configured during system generation. These parameters shall be located in the file Fls_Cfg.h. Further hardware or implementation specific parameters can be added if necessary.

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>FLS172:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>Fls_ModuleConfiguration</td>
</tr>
<tr>
<td>Description</td>
<td>Configuration parameters for module flash driver.</td>
</tr>
</tbody>
</table>

**Configuration Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_DEV_ERROR_DETECT</td>
<td>Pre-processor switch for enabling the development error detection and reporting (see FLS077).</td>
<td>#define</td>
<td>--</td>
<td>ON</td>
<td>Development error detection and reporting enabled</td>
<td>Module</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OFF</td>
<td>Development error detection and reporting disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pre-compile X</td>
<td>All variants</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Link time --</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Post Build --</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_USE_INTERRUPTS</td>
<td>Job processing triggered by hardware interrupt</td>
<td>#define</td>
<td>--</td>
<td>ON</td>
<td>Job processing triggered by interrupt (hardware controlled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OFF</td>
<td>Job processing not triggered by interrupt (software controlled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pre-compile X</td>
<td>All variants</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Link time --</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Post Build --</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

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

**Scope**: Module  
**Dependency**: None

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_TOTAL_SIZE</td>
<td>The total amount of flash memory in bytes (see also FLS118). FLS170: This parameter in conjunction with FLS_BASE_ADDRESS defines the upper boundary for read / write / erase and compare jobs.</td>
<td>#define</td>
<td></td>
<td>--</td>
<td>Pre-compile</td>
</tr>
</tbody>
</table>

|                      |                                                                                                                                          |        |      | --              | Link time          |
|                      |                                                                                                                                          |        |      | --              | Post Build         |

| Scope   | Module                                                                                                                                  |        |      | --              |                        |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------|        |      | --              |                        |

### FLS_AC_LOAD_ON_JOB_START

**Scope**: Module  
**Dependency**: None

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_AC_LOAD_ON_JOB_START</td>
<td>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.</td>
<td>#define</td>
<td></td>
<td>--</td>
<td>Pre-compile</td>
</tr>
</tbody>
</table>

|                      |                                                                                                                                          |        |      | --              | Link time          |
|                      |                                                                                                                                          |        |      | --              | Post Build         |

| Scope   | Module                                                                                                                                  |        |      | --              |                        |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------|        |      | --              |                        |

### FLS_VERSION_INFO_API

**Scope**: Module  
**Dependency**: None

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_VERSION_INFO_API</td>
<td>Compile switch to enable / disable the version information function.</td>
<td>#define</td>
<td></td>
<td>--</td>
<td>Pre-compile</td>
</tr>
</tbody>
</table>

|                      |                                                                                                                                          |        |      | --              | Link time          |
|                      |                                                                                                                                          |        |      | --              | Post Build         |

| Scope   | Module                                                                                                                                  |        |      | --              |                        |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------|        |      | --              |                        |

### FLS_CANCEL_API

**Scope**: Module  
**Dependency**: None

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_CANCEL_API</td>
<td>Compile switch to enable / disable the Fls_Cancel function.</td>
<td>#define</td>
<td></td>
<td>--</td>
<td>Pre-compile</td>
</tr>
</tbody>
</table>

|                      |                                                                                                                                          |        |      | --              | Link time          |
|                      |                                                                                                                                          |        |      | --              | Post Build         |

| Scope   | Module                                                                                                                                  |        |      | --              |                        |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------|        |      | --              |                        |

### FLS_COMPARE_API

**Scope**: Module  
**Dependency**: None

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_COMPARE_API</td>
<td>Compile switch to enable / disable the Fls_Compare function.</td>
<td>#define</td>
<td></td>
<td>--</td>
<td>Pre-compile</td>
</tr>
</tbody>
</table>

|                      |                                                                                                                                          |        |      | --              | Link time          |
|                      |                                                                                                                                          |        |      | --              | Post Build         |

| Scope   | Module                                                                                                                                  |        |      | --              |                        |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------|        |      | --              |                        |
### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fls_ConfigSet</td>
<td>1..*</td>
<td>Module initialization</td>
</tr>
</tbody>
</table>

### 10.2.3 Fls_ConfigSet

**FLS173:** The following table specifies the parameters that shall be located in an external data structure of type Fls_ConfigType. The organization and location of this data structure shall be up to the implementer. The type declaration shall be located in
the file Fls.h. Further hardware or implementation specific parameters can be added if necessary.

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>FLS174:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>Fls_ConfigSet</td>
</tr>
<tr>
<td>Description</td>
<td>Configuration parameters for module initialization.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Unit</td>
</tr>
<tr>
<td>Range</td>
</tr>
<tr>
<td>Configuration Class</td>
</tr>
<tr>
<td>Link time</td>
</tr>
<tr>
<td>Post Build</td>
</tr>
<tr>
<td>Scope</td>
</tr>
<tr>
<td>Dependency</td>
</tr>
</tbody>
</table>

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

| Name | FLS_MAX_ERASE_NORMAL_MODE |
| Description | The maximum number of bytes to erase in one cycle of the flash driver’s job processing function in normal mode. |
| Type | integer |
| Unit | -- |
| Range | -- Implementation specific |
| Configuration Class | Pre-compile X Variant PC |
| Link time | -- -- |
| Post Build | X Variant PB |
| Scope | Module |
| Dependency | FLS175: This value has to correspond to the settings in FLS_SECTOR_LIST. The minimum number is defined by the size of one flash sector and therefore depends on the underlying flash device. |

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

**FLS181**: This value has to correspond to the settings in FLS\_SECTOR\_LIST. The minimum number is defined by the size of one flash sector and therefore depends on the underlying flash device.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
<th>Scope</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>FLS_MAX_WRITE_NORMAL_MODE</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The maximum number of bytes to write in one cycle of the flash driver’s job processing function in normal mode.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unit</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Configuration Class</strong></td>
<td><strong>Pre-compile</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PC</strong></td>
<td><strong>Post Build</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Post Build</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PB</strong></td>
<td></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td><strong>Module</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td><strong>FLS176</strong>: 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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
<th>Scope</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>FLS_MAX_WRITE_FAST_MODE</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The maximum number of bytes to write in one cycle of the flash driver’s job processing function in fast mode.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unit</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Configuration Class</strong></td>
<td><strong>Pre-compile</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PC</strong></td>
<td><strong>Post Build</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Post Build</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PB</strong></td>
<td></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td><strong>Module</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td><strong>FLS176</strong>: 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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Dependency

**FLS\_MAX\_READ\_NORMAL\_MODE**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
<th>Scope</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>FLS_MAX_READ_NORMAL_MODE</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The maximum number of bytes to read or compare in one cycle of the flash driver’s job processing function in normal mode.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unit</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Configuration Class</strong></td>
<td><strong>Pre-compile</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PC</strong></td>
<td><strong>Post Build</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Post Build</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PB</strong></td>
<td></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td><strong>Module</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td>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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Dependency

**FLS\_MAX\_READ\_FAST\_MODE**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Range</th>
<th>Configuration Class</th>
<th>Scope</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>FLS_MAX_READ_FAST_MODE</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The maximum number of bytes to read or compare in one cycle of the flash driver’s job processing function in fast mode.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unit</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Configuration Class</strong></td>
<td><strong>Pre-compile</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PC</strong></td>
<td><strong>Post Build</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Link time</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Post Build</strong></td>
<td><strong>X</strong></td>
<td><strong>Variant PB</strong></td>
<td></td>
<td><strong>--</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td><strong>Module</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td>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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Name: FLS_AC_ERASE

**Description:**
Address offset in RAM to which the erase flash access code shall be loaded. Used as function pointer to access the erase flash access code.

**Type:** Hardware specific

**Unit:** --

**Range:** -- --

**Configuration Class:**
- Pre-compile: X Variant PC
- Link time: -- --
- Post Build: X Variant PB

**Scope:** Module

**Dependency:** None

### Name: FLS_AC_WRITE

**Description:**
Address offset in RAM to which the write flash access code shall be loaded. Used as function pointer to access the write flash access code.

**Type:** Hardware specific

**Unit:** --

**Range:** -- --

**Configuration Class:**
- Pre-compile: X Variant PC
- Link time: -- --
- Post Build: X Variant PB

**Scope:** Module

**Dependency:** None

### Name: FLS_JOB_END_NOTIFICATION

**Description:**
Mapped to the job end notification routine provided by the upper layer module, typically the Flash EEPROM Emulation module.

**Type:** Function pointer

**Unit:** --

**Range:** Implementation specific

**Configuration Class:**
- Pre-compile: X Variant PC
- Link time: -- --
- Post Build: X Variant PB

**Scope:** Module

**Dependency:** None

### Name: FLS_JOB_ERROR_NOTIFICATION

**Description:**
Mapped to the job error notification routine provided by the upper layer module, typically the Flash EEPROM Emulation module.

**Type:** Function pointer

**Unit:** --

**Range:** Implementation specific

**Configuration Class:**
- Pre-compile: X Variant PC
- Link time: -- --
- Post Build: X Variant PB

**Scope:** Module

**Dependency:** None

### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
10.3 Published Information

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

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

<table>
<thead>
<tr>
<th><strong>SWS Item</strong></th>
<th><strong>FLS178:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Information elements</strong></td>
<td><strong>Type / Range</strong></td>
</tr>
<tr>
<td><strong>Information element name</strong></td>
<td><strong>Information element description</strong></td>
</tr>
<tr>
<td>FLS_VENDOR_ID</td>
<td>#define/uint16 Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list</td>
</tr>
<tr>
<td>FLS_MODULE_ID</td>
<td>#define/uint8 Module ID of this module from Module List</td>
</tr>
<tr>
<td>FLS_AR_MAJORVERSION</td>
<td>#define/uint8 Major version number of AUTOSAR specification on which the appropriate implementation is based on.</td>
</tr>
<tr>
<td>FLS_AR_MINORVERSION</td>
<td>#define/uint8 Minor version number of AUTOSAR specification on which the appropriate implementation is based on.</td>
</tr>
<tr>
<td>FLS_AR_PATCHVERSION</td>
<td>#define/uint8 Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.</td>
</tr>
<tr>
<td>FLS_SW_MAJORVERSION</td>
<td>#define/uint8 Major version number of the vendor specific implementation of the module. The numbering is vendor specific.</td>
</tr>
<tr>
<td>FLS_SW_MINORVERSION</td>
<td>#define/uint8 Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.</td>
</tr>
<tr>
<td>FLS_SW_PATCHVERSION</td>
<td>#define/uint8 Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.</td>
</tr>
<tr>
<td>FLS_PAGE_LIST</td>
<td>-- / -- List of flash pages (smallest writable flash memory units) specifying start address (for checking of address alignment) and size (for checking write job sizes) for all different flash page sizes.</td>
</tr>
<tr>
<td>FLS_SECTOR_LIST</td>
<td>-- / -- List of flash sectors (smallest erasable flash memory units) specifying start address (for checking of address alignment) and size (for checking erase job sizes) for all different flash sector sizes.</td>
</tr>
<tr>
<td>FLS_ERASED_VALUE</td>
<td>Integer / -- The contents of an erased flash memory cell.</td>
</tr>
<tr>
<td>FLS_ERASE_TIME</td>
<td>µsec / -- Maximum time to erase one complete flash sector.</td>
</tr>
<tr>
<td><strong>Information element name</strong></td>
<td><strong>Type / Range</strong></td>
</tr>
<tr>
<td><strong>Information element description</strong></td>
<td></td>
</tr>
<tr>
<td>FLS_WRITE_TIME</td>
<td>µsec / -- Maximum time to program one complete flash page.</td>
</tr>
<tr>
<td>FLS_AC_SIZE_ERASE</td>
<td>Integer / -- Number of bytes in RAM needed for the erase flash access code.</td>
</tr>
<tr>
<td>FLS_AC_SIZE_WRITE</td>
<td>Integer / -- Number of bytes in RAM needed for the write flash access code.</td>
</tr>
<tr>
<td>FLS_AC_LOCATION_ERASE</td>
<td>Integer / -- Position in RAM, to which the erase flash access code has to be loaded. Only relevant if the erase flash access code is not position independent. If this information is not provided it is assumed that the erase flash access code is position independent and that therefore the RAM position can be freely configured.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>FLS_AC_LOCATION_WRITE</td>
<td>Integer / --</td>
</tr>
<tr>
<td>FLSEXPECTED_HW_ID</td>
<td>-- / --</td>
</tr>
</tbody>
</table>
11 Changes to Release 1

11.1 Deleted SWS Items

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS074</td>
<td>New SWS template (didn’t fit with new document structure)</td>
</tr>
<tr>
<td>FLS045</td>
<td>New SWS template (didn’t fit with new document structure)</td>
</tr>
<tr>
<td>FLS115</td>
<td>New SWS template (didn’t fit with new document structure)</td>
</tr>
</tbody>
</table>

11.2 Replaced SWS Items

<table>
<thead>
<tr>
<th>SWS Item of Release 1</th>
<th>replaced by SWS Item</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS112</td>
<td>FLS167</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS113</td>
<td>FLS168</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS043</td>
<td>FLS169</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS044</td>
<td>FLS170</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS085</td>
<td>FLS171</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS103</td>
<td>FLS173</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS050</td>
<td>FLS175</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS051</td>
<td>FLS176</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS119</td>
<td>FLS177</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
<tr>
<td>FLS106</td>
<td>FLS179</td>
<td>New SWS template (copy-paste didn’t work on the tags)</td>
</tr>
</tbody>
</table>

11.3 Changed SWS Items

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS077</td>
<td>New SWS template</td>
</tr>
<tr>
<td>FLS016, FLS017, FLS018, FLS024, FLS033, FLS036, FLS104, FLS105, FLS106, FLS022, FLS055, FLS056, FLS052, FLS095</td>
<td>New memory hardware abstraction architecture (reference to memory abstraction interface instead of flash interface)</td>
</tr>
<tr>
<td>FLS031, FLS036, FLS037, FLS040, FLS052</td>
<td>Added functionality (Read, Compare, SetMode functions)</td>
</tr>
<tr>
<td>FLS015</td>
<td>NULL pointer check removed (see FLS194)</td>
</tr>
</tbody>
</table>
## 11.4 Added SWS Items

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS145, FLS146</td>
<td>Bugzilla entry #4873</td>
</tr>
<tr>
<td>FLS147</td>
<td>Bugzilla entry #4507</td>
</tr>
<tr>
<td>FLS148, FLS149, FLS150, FLS151, FLS152, FLS153, FLS154, FLS155, FLS156, FLS181, FLS182</td>
<td>RFC #6793: Same functionality as EEPROM driver. Compare and SetMode functions added to the flash driver.</td>
</tr>
<tr>
<td>FLS157, FLS158</td>
<td>Bugzilla entry #4621</td>
</tr>
<tr>
<td>FLS159, FLS160, FLS161, FLS162, FLS163, FLS164, FLS167, FLS168, FLS172, FLS174, FLS178</td>
<td>New SWS template</td>
</tr>
<tr>
<td>FLS165, FLS166</td>
<td>New SWS template: GetVersionInfo function</td>
</tr>
<tr>
<td>FLS183, FLS184, FLS185, FLS186, FLS187, FLS188</td>
<td>RFC #6798: Scalability of flash driver</td>
</tr>
<tr>
<td>FLS190</td>
<td>BSW00432</td>
</tr>
<tr>
<td>FLS191</td>
<td>BSW12265</td>
</tr>
<tr>
<td>FLS193</td>
<td>To resolve review issue for BSW00325</td>
</tr>
<tr>
<td>FLS194</td>
<td>Added because of BSW00414</td>
</tr>
</tbody>
</table>